


Politechnika Bydgoska im. Jana i Jędrzeja Śniadeckich Wydział Telekomunikacji, Informatyki i Elektrotechniki al. prof. S. Kaliskiego 7, 85-796 Bydgoszcz				 <b>POLITECHNIKA BYDGOSKA</b> Wydział Telekomunikacji, Informatyki i Elektrotechniki	
Przedmiot	Programowanie urządzeń mobilnych			Kierunek/Tryb	
Nr laboratorium	10	Data wykonania		Grupa	
Ocena		Data oddania		Imię	
Nazwa ćwiczenia	Pętla gry, animacja postaci, kolizje, obsługa dotyku			Nazwisko	

### Cel ćwiczenia laboratoryjnego

Celem tego ćwiczenia jest stworzenie gry inspirowanej na klasycznej grze Boulderdash, realizacja zadania obejmuje pętlę gry, animacje postaci i przeciwników, tło, dźwięki, obsługę zdarzeń oraz sterowanie postacią. Uczestnik będzie miał okazję zaznajomić się z różnymi aspektami programowania aplikacji mobilnych.

### Zadanie

Studenci zostaną poproszeni o utworzenie gry, którą będzie rozszerzeniem i połączeniem przykładów zawartych w poprzednich ćwiczeniach oraz zastosowaniem wiedzy zawartej w poniższych przykładach.

### Opcjonalne rozszerzenia

- Możesz dodać bardziej rozbudowany interfejs użytkownika, który pozwoli graczowi wybierać poziom trudności.
- Możesz dodać zapis wyników gry i wyników graczy.

### Oczekiwane wyniki

Po ukończeniu ćwiczenia, uczestnik powinien być w stanie:

1. Utworzyć działającą pętlę gry, zapewniającą płynność działania.
2. Dodawać zmieniające się tło do sceny gry, nadając jej atrakcyjny wygląd.
3. Tworzyć animacje postaci, wzbogacając interakcje w grze.
4. Dodawać i sterować dźwiękami.
5. Implementować interaktywne sterowanie postacią, co pozwoli graczowi aktywnie uczestniczyć w rozgrywce.

### Zadanie

Na podstawie przedstawionego poniżej szkieletu aplikacji implementującego pętlę gry należy uzupełnić rozgrywkę inspirowaną klasyczną grą Boulderdash (lista na końcu instrukcji).

### Zasady gry

Boulderdash to klasyczna gra komputerowa z gatunku łamigłówek, która została wydana w latach 80. Gra polega na eksplorowaniu jaskiń w poszukiwaniu diamentów, jednocześnie unikając groźnych przeszkód i zbierając punkty.

Cel gry: Celem gracza jest zebranie wszystkich diamentów dostępnych na poziomie oraz dotarcie do wyjścia z jaskini.

Jaskinie: Gra składa się z wielu poziomów, z których każdy jest jaskinią, zbudowaną z różnych rodzajów bloków, takich jak ziemia, skały, diamenty, a także niebezpieczne obiekty jak kamienie, pułapki i moby.

**Gracz:** Gracz kontroluje postać o imieniu Rockford, która może kopać dziury w ziemi i przesuwając bloki w poziomie. Gracz musi sprawnie manewrować Rockfordem, aby unikać niebezpieczeństw i zdobywać diamenty.

**Zasady ruchu:** Rockford może poruszać się w czterech kierunkach: w lewo, w prawo, do góry i w dół. Gracz może także kopać dziury w ziemi, co pozwala na spadanie kamieni i innych bloków.

**Diamenty:** Głównym celem gry jest zebranie wszystkich diamentów rozsianych po poziomie. Zbieranie diamentów zwiększa wynik gracza i pozwala na przejście do następnego poziomu.

**Przeszkody:** Na drodze gracza stoją różnego rodzaju przeszkody, takie jak kamienie, które mogą zablokować drogę, jak również pułapki, takie jak spadające bloki lub moby, które mogą zniszczyć Rockforda.

**Wyjście:** Po zebraniu wszystkich diamentów gracz musi dotrzeć do wyjścia z jaskini, aby ukończyć poziom.

**Czas:** Czasami gra może być ograniczona czasowo, przez co gracz musi działać szybko, aby ukończyć poziom przed upływem czasu.

**Punkty:** Gracz zdobywa punkty za zbieranie diamentów oraz może zdobyć dodatkowe punkty za wykonywanie zadań specjalnych, takich jak wyeliminowanie wszystkich kamieni z danego obszaru.

**Poziomy trudności:** Wraz z postępem w grze, poziomy stają się coraz trudniejsze, wprowadzając nowe rodzaje przeszkód i bardziej skomplikowane układy.

## Sprawozdanie

Sprawozdanie z ćwiczenia w ramach nauki programowania aplikacji mobilnych powinno zawierać istotne informacje i dokumentację dotyczącą zadania oraz jego realizacji. Oto kilka kluczowych 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.
- Charakterystyka 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.

Przykład do wykorzystania w ćwiczeniu i opis propozycji rozwiązania zadania

Przykład prostego projektu aplikacji Android z pętlą gry. W tym przykładzie używamy Android Studio, języka Java i bibliotek Android.

Utworzenie nowego projektu w Android Studio

Uruchom Android Studio.

Wybierz opcję "New Project".

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

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

Kliknij "Finish", aby utworzyć projekt.

Pętla gry - GameLoop

Pętla gry (GameLoop) jest kluczowym elementem każdej gry, także tych tworzonych na platformę Android w środowisku Android Studio, używając języka Java. To jest przykładowy opis, jak można zaimplementować GameLoop:

Inicjalizacja Pętli Gry (GameLoop)

W Androidzie GameLoop zwykle implementuje się w klasie pochodnej od SurfaceView i implementującej interfejs Runnable.

Tworzysz wątek (Thread), który będzie odpowiedzialny za wywoływanie aktualizacji i rysowania w określonym interwale czasu.

#### Metoda run

Metoda run interfejsu Runnable zawiera główną pętlę gry.

W tej pętli kontrolujesz czas (liczysz klatki na sekundę), wykonujesz aktualizacje stanu gry i wywołujesz metody rysujące.

#### Aktualizacja Stanu Gry

W każdej iteracji pętli aktualizujesz stan gry, co może obejmować logikę gry, ruch postaci, detekcję kolizji itp.

#### Rysowanie

Po zaktualizowaniu stanu gry rysujesz nowy stan na ekranie. To obejmuje rysowanie postaci, tła, wyników itd.

#### Kontrola Czasu

Ważne jest, aby kontrolować, jak często pętla jest aktualizowana i rysowana. Zwykle dąży się do osiągnięcia stałej liczby klatek na sekundę (FPS).

#### Zatrzymywanie Pętli

Kiedy gra jest zamykana lub przechodzi w stan pauzy, należy odpowiednio zatrzymać wątek, aby uniknąć przecieków pamięci.

Poniższy kod jest przykładem bardzo podstawowej pętli gry w Android Studio, która zawiera elementy niezbędne do uruchomienia środowiska gry. Kod ten jest częścią aplikacji na system Android, która odpowiada za prostą animację postaci o nazwie "Rockford".

Import: Importowane są potrzebne biblioteki, takie jak AppCompatActivity, Bitmap, Handler, MotionEvent, View, ImageView i RelativeLayout.

Enum: Definiowany jest enum o nazwie RockfordDo, który reprezentuje różne akcje, jakie Rockford może wykonywać: stanie, spoczynku, ruch w lewo, ruch w prawo, pojawienie się i śmierć.

Zmienne: Deklarowane są różne zmienne, takie jak frameCount, frameWidth, imageView, handler, spritesheet\_rockford\_stand, spritesheet\_rockford\_idle itp., aby obsługiwać animację, ruch i zdarzenia dotykowe.

onCreate(): Ta metoda jest wywoływana przy pierwszym tworzeniu aktywności. Inicjalizuje ona układ, ustawia kolor tła, inicjalizuje widoki obrazów, wczytuje arkusze sprite'ów, oblicza szerokość klatki, ustawia nasłuchiwanie dotyku i rozpoczyna animację.

setTouchListeners(): Ustawia nasłuchiwanie dotyku dla obszarów sterowania (góra, dół, lewo, prawo).

moveRockford(): Przesuwa postać Rockforda na podstawie zdarzeń dotyku.

startAnimation(): Rozpoczyna animację, wysyłając Runnable, który regularnie aktualizuje klatkę animacji.

onDestroy(): Czyści zasoby, gdy aktywność jest niszczona, w tym usuwa wywołania zwrotne i recykluje bitmapy.

Kod obsługuje zdarzenia dotykowe w celu sterowania ruchem Rockforda i aktualizuje klatki animacji tak aby stworzyć efekt animacji.

Activity\_main.xml

Widok:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/mainLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#000000" >

    <!-- Rockford -->
    <ImageView
        android:id="@+id/sprite_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true" />

    <!-- Obszar sterowania prawy -->
    <View
        android:id="@+id/controlRight"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_alignParentEnd="true"
        android:layout_alignParentBottom="true"
        android:layout_marginEnd="14dp"
        android:layout_marginBottom="73dp"
        android:background="#66FFFFFF" />

    <!-- Obszar sterowania lewy -->
    <View
        android:id="@+id/controlLeft"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_alignParentBottom="true"
        android:layout_marginEnd="17dp"
        android:layout_marginBottom="73dp"
        android:layout_toStartOf="@+id/controlRight"
        android:background="#66FFFFFF" />

    <!-- Obszar sterowania góra -->
    <View
        android:id="@+id/controlTop"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_above="@+id/controlDown"
        android:layout_alignParentStart="true"
        android:layout_marginStart="15dp"
        android:layout_marginBottom="13dp"
        android:background="#66FFFFFF" />

    <!-- Obszar sterowania dół -->
    <View
        android:id="@+id/controlDown"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_alignParentStart="true"
        android:layout_alignParentBottom="true"
        android:layout_marginStart="15dp"
        android:layout_marginBottom="17dp"
        android:background="#66FFFFFF" />

</RelativeLayout>
```

MainActivity.java

W głównej aktywności aplikacji (MainActivity.java) inicjalizujesz i uruchamiasz GameView:

```
package com.example.simpleanimation1;
```

```
import androidx.appcompat.app.AppCompatActivity;
```

```
import android.graphics.Bitmap;
```

```
import android.graphics.BitmapFactory;
```

```
import android.os.Bundle;
```

```
import android.os.Handler;
```

```
import android.view.MotionEvent;
```

```
import android.view.View;
```

```
import android.widget.ImageView;
```

```
import android.widget.RelativeLayout;
```

```
public class MainActivity extends AppCompatActivity {  
    private enum RockfordDo {stand,idle,left,right,appear,die};
```

```
    private int frameCount = 7; // Ilość klatek w spritesheecie
```

```
    private int frameWidth; // szerokość pojedynczej klatki
```

```
    private ImageView imageView;
```

```
    private int frameIndex = 0;
```

```
    private Handler handler;
```

```
    private Bitmap spritesheet_rockford_stand;
```

```
    private Bitmap spritesheet_rockford_idle;
```

```
    private Bitmap spritesheet_rockford_left;
```

```
    private Bitmap spritesheet_rockford_right;
```

```
    private RockfordDo rockfordDo = RockfordDo.stand;
```

```
    private int idleTimer=0;
```

```
    private int idletime=40;
```

```
    private int scale = 1; // Współczynnik powiększenia
```

```
    private float imageX , imageY =100; // współrzędne Rockforda
```

```
    private boolean movingUp, movingDown, movingLeft, movingRight;
```

```
    private float deltaX = 64;
```

```
    private float deltaY = 64;
```

```
// Definicja obszarów sterowania
```

```
    private View topControl, bottomControl, leftControl, rightControl;
```

```
@Override
```

```
protected void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);
```

```
    setContentView(R.layout.activity_main);
```

```
    RelativeLayout mainLayout = findViewById(R.id.mainLayout); // Pobieramy RelativeLayout z layoutu
```

```
    mainLayout.setBackgroundColor(getResources().getColor(android.R.color.black)); // Ustawiamy czarne tło
```

```
    imageView = findViewById(R.id.sprite_view);
```

```
    handler = new Handler();
```

```
    spritesheet_rockford_stand = BitmapFactory.decodeResource(getResources(),R.drawable.rockford_stand_32x32);
```

```
    spritesheet_rockford_idle = BitmapFactory.decodeResource(getResources(),R.drawable.rockford_idle_32x32);
```

```
    spritesheet_rockford_left = BitmapFactory.decodeResource(getResources(),R.drawable.rockford_left_32x32);
```

```
    spritesheet_rockford_right = BitmapFactory.decodeResource(getResources(),R.drawable.rockford_right_32x32);
```

```
// Obliczamy szerokość pojedynczej klatki
```

```

frameWidth = spritesheet_rockford_stand.getWidth() / frameCount;

// Ustawiamy szerokość ImageView na szerokość pojedynczej klatki, pomnożoną przez współczynnik powiększenia
imageView.getLayoutParams().width = frameWidth * scale;
imageView.getLayoutParams().height = spritesheet_rockford_stand.getHeight() * scale; // Ustawiamy wysokość
proporcjonalnie do powiększenia

imageView.requestLayout();

// Inicjalizacja obszarów sterowania
topControl = findViewById(R.id.controlTop);
bottomControl = findViewById(R.id.controlDown);
leftControl = findViewById(R.id.controlLeft);
rightControl = findViewById(R.id.controlRight);

// Ustawiamy obszary sterowania jako dotykowe
setTouchListeners();

// Uruchamiamy animację
startAnimation();
}

private void setTouchListeners() {
    topControl.setOnTouchListener(new View.OnTouchListener() {
        @Override
        public boolean onTouch(View v, MotionEvent event) {
            switch (event.getAction()) {
                case MotionEvent.ACTION_DOWN:
                    movingUp = true;
                    idleTimer=0;
                    break;
                case MotionEvent.ACTION_UP:
                    movingUp = false;
                    rockfordDo=RockfordDo.stand;
                    break;
            }
            return true;
        }
    });

    bottomControl.setOnTouchListener(new View.OnTouchListener() {
        @Override
        public boolean onTouch(View v, MotionEvent event) {
            switch (event.getAction()) {
                case MotionEvent.ACTION_DOWN:
                    movingDown = true;
                    idleTimer=0;
                    break;
                case MotionEvent.ACTION_UP:
                    movingDown = false;
                    rockfordDo=RockfordDo.stand;
                    break;
            }
            return true;
        }
    });

    leftControl.setOnTouchListener(new View.OnTouchListener() {
        @Override
        public boolean onTouch(View v, MotionEvent event) {
            switch (event.getAction()) {
                case MotionEvent.ACTION_DOWN:
                    movingLeft = true;

```

```

        idleTimer=0;
        break;
    case MotionEvent.ACTION_UP:
        movingLeft = false;
        rockfordDo=RockfordDo.stand;
        break;
    }
    return true;
}
});

rightControl.setOnTouchListener(new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                movingRight = true;
                idleTimer=0;
                break;
            case MotionEvent.ACTION_UP:
                movingRight = false;
                rockfordDo=RockfordDo.stand;
                break;
        }
        return true;
    }
});
}

```

```

private void moveRockford() {
    if (movingUp) {
        // Przesunięcie postaci w górę
        imageY -= deltaY;
        rockfordDo = RockfordDo.left;
    }
    if (movingDown) {
        // Przesunięcie postaci w dół
        imageY += deltaY;
        rockfordDo = RockfordDo.left;
    }
    if (movingLeft) {
        // Przesunięcie postaci w lewo
        imageX -= deltaX;
        rockfordDo = RockfordDo.left;
    }
    if (movingRight) {
        // Przesunięcie postaci w prawo
        imageX += deltaX;
        rockfordDo = RockfordDo.right;
    }
    imageView.setX(imageX);
    imageView.setY(imageY);
}

```

```

private void startAnimation() {
    Runnable animationRunnable = new Runnable() {
        @Override
        public void run() {
            // Ustawiamy kolejną klatkę animacji
            // Wycinamy fragment z spritesheeta odpowiadający danej klatce i ustawiamy w ImageView
            Bitmap frame;
            switch (rockfordDo){
                case stand:

```



```

        frame = Bitmap.createBitmap(spritesheet_rockford_stand, frameIndex * frameWidth, 0, frameWidth,
spritesheet_rockford_stand.getHeight());
        frame = Bitmap.createScaledBitmap(frame, frameWidth * scale, spritesheet_rockford_stand.getHeight() * scale, true);
        moveRockford();
        idleTimer++;
        imageView.setImageBitmap(frame);
        if (idleTimer>20) {rockfordDo=RockfordDo.idle;}
        break;
    case idle:
        frame = Bitmap.createBitmap(spritesheet_rockford_idle, frameIndex * frameWidth, 0, frameWidth,
spritesheet_rockford_idle.getHeight());
        frame = Bitmap.createScaledBitmap(frame, frameWidth * scale, spritesheet_rockford_idle.getHeight() * scale, true);
        moveRockford();
        imageView.setImageBitmap(frame);
        break;
    case right:
        frame = Bitmap.createBitmap(spritesheet_rockford_right, frameIndex * frameWidth, 0, frameWidth,
spritesheet_rockford_right.getHeight());
        frame = Bitmap.createScaledBitmap(frame, frameWidth * scale, spritesheet_rockford_right.getHeight() * scale, true);
        moveRockford();
        imageView.setImageBitmap(frame);
        idleTimer=0;
        break;
    case left:
        frame = Bitmap.createBitmap(spritesheet_rockford_left, frameIndex * frameWidth, 0, frameWidth,
spritesheet_rockford_left.getHeight());
        frame = Bitmap.createScaledBitmap(frame, frameWidth * scale, spritesheet_rockford_left.getHeight() * scale, true);
        moveRockford();
        imageView.setImageBitmap(frame);
        idleTimer=0;
        break;
    }
    // Inkrementujemy indeks klatki
    frameIndex = (frameIndex + 1) % frameCount;

    // Wywołujemy ponownie runnable po pewnym czasie dla następnej klatki
    handler.postDelayed(this, 100); // Ustaw czas opóźnienia między klatkami (tutaj 100ms)
}
};
handler.post(animationRunnable);
}

@Override
protected void onDestroy() {
    super.onDestroy();
    handler.removeCallbacksAndMessages(null);
    spritesheet_rockford_stand.recycle(); // Zwolnij zasoby bitmapy spritesheeta
    spritesheet_rockford_left.recycle(); // Zwolnij zasoby bitmapy spritesheeta
    spritesheet_rockford_right.recycle(); // Zwolnij zasoby bitmapy spritesheeta
}
}

```

Należy pamiętać aby dodać uprawnienie WAKE\_LOCK do Twojej aplikacji Android, w tym celu musisz umieścić odpowiedni wpis w pliku manifestu (AndroidManifest.xml). Wprowadź go między sekcję <manifest> i <application>.

Przykład, jak to może wyglądać w pliku manifestu:

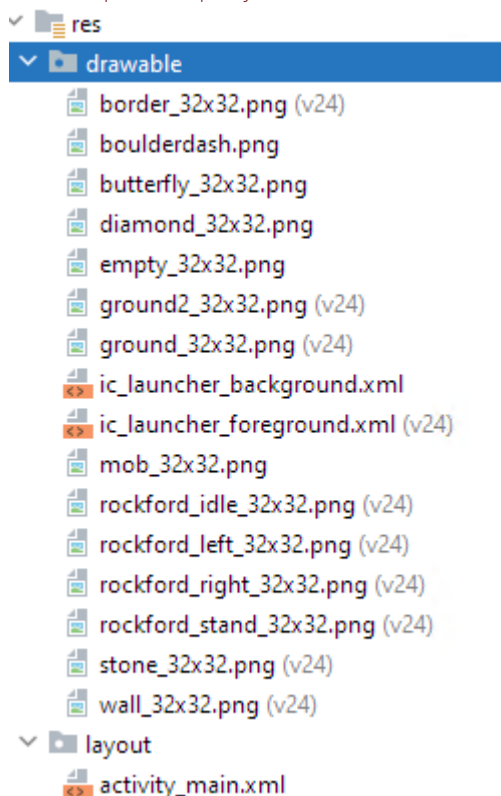
```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.yourpackage">
    <!-- Pozostałe elementy manifestu -->
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <application>
        <!-- Pozostałe elementy aplikacji -->
    
```

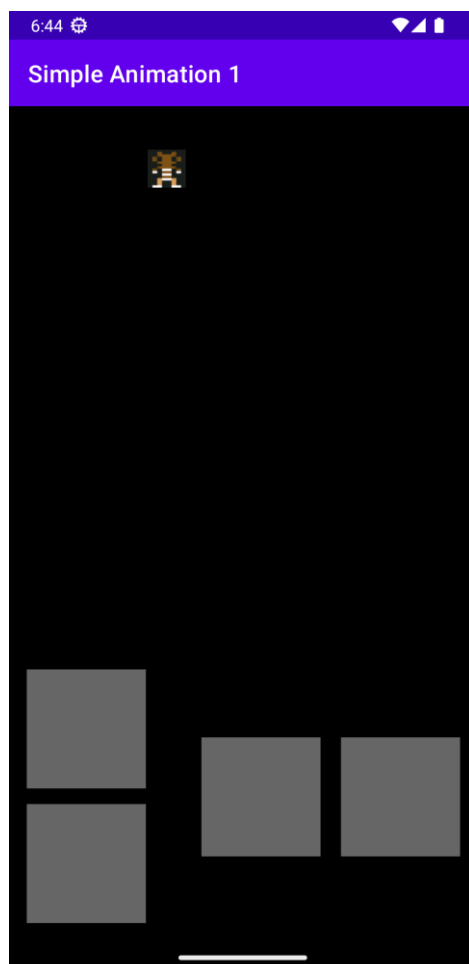
```
</application>
</manifest>
```

Pamiętaj, że zastępujesz `com.example.yourpackage` nazwą pakietu Twojej aplikacji. Pamiętaj, że nadmiarowe uprawnienia powinny być używane z rozważą, ponieważ zbędne uprawnienia mogą zaniepokoić użytkowników i potencjalnie negatywnie wpłynąć na ich zaufanie do Twojej aplikacji. Upewnij się, że uprawnienia, które dodajesz, są związane z rzeczywistymi potrzebami Twojej aplikacji. `android.permission.WAKE_LOCK` to uprawnienie w systemie Android, które pozwala aplikacji utrzymywać urządzenie w stanie czuwania (wakeup) nawet, gdy ekran jest wygaszony. Bez tego uprawnienia urządzenie może wejść w stan uśpienia, co może ograniczać działanie niektórych operacji w tle. Użycie tego uprawnienia jest przydatne w sytuacjach, gdzie chcesz, aby aplikacja działała w tle, nawet gdy ekran jest wygaszony, na przykład podczas odtwarzania muzyki, obsługi powiadomień w czasie rzeczywistym lub monitorowania lokalizacji urządzenia. Jednak należy używać tego uprawnienia z umiarem, ponieważ utrzymywanie urządzenia w stanie czuwania może wpływać na zużycie energii, co z kolei może skutkować skróceniem czasu pracy baterii urządzenia. Dlatego ważne jest, aby sprawdzić, czy rzeczywiście potrzebujesz tego uprawnienia do poprawnego działania Twojej aplikacji.

Widok plików projektu:



Widok aplikacji:



W ten sposób utworzyliśmy animowany element imageView reprezentujący Rockforda, dodatkowo mamy możliwość jego poruszania po planszy (która jeszcze jest pusta) za pomocą zdefiniowanych obszarów reagujących na dotyk gracza.

Następnie dodamy planszę, po której Rockford będzie się mógł poruszać. Otwórz plik `activity_main.xml`, który znajduje się w folderze `res/layout/` Twojego projektu. Dodaj do niego kontener, który będzie reprezentował planszę. W tym przypadku będzie to `GridLayout`. Zmodyfikowany kod XML definiuje układ interfejsu użytkownika w aplikacji Android za pomocą `FrameLayout`.

**FrameLayout:** Jest to kontener, który zawiera wszystkie elementy interfejsu użytkownika. Ma ustawione wymiary na `match_parent`, co oznacza, że będzie zajmować całą dostępną przestrzeń na ekranie.

**View (controlRight, controlLeft, controlTop, controlDown):** Są to widoki typu `View`, które reprezentują obszary sterowania umieszczone w różnych miejscach na ekranie (prawy górny róg, lewy górny róg, górny środek, dolny środek). Każdy z tych widoków ma ustawione wymiary na `100x100 dp` oraz kolor tła ustawiony na przezroczysty biały (`#66FFFFFF`). Te obszary mogą być używane jako przyciski lub elementy interaktywne.

**GridLayout (boardGridLayout):** Jest to siatka, która może zawierać inne widoki. W tym przypadku służy jako plansza gry. Parametry ustawione dla `GridLayout` określają, że ma ona wymiary dopasowane do zawartości, jest umieszczona w lewym górnym rogu (`layout_gravity="top|left"`) i nie ma żadnych marginesów ani wypełnień.

**ImageView (sprite\_view):** Jest to widok `ImageView`, który wyświetla grafikę. W tym przypadku reprezentuje postać w grze. Ma on ustawione wymiary na `0x0 dp`, ale jest wypełniony do szerokości ekranu (`layout_gravity="center_horizontal|fill"`).

Cały układ można dostosować do potrzeb rozwijanej aplikacji, na przykład poprzez zmianę kolorów, wymiarów, dodanie obsługi zdarzeń itp.

Activity\_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout 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:id="@+id/mainLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <!-- Obszar sterowania prawy -->
    <View
        android:id="@+id/controlRight"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_gravity="bottom|right"
        android:layout_marginRight="17dp"
        android:layout_marginBottom="73dp"
        android:background="#66FFFFFF" />

    <!-- Obszar sterowania lewy -->
    <View
        android:id="@+id/controlLeft"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_gravity="bottom|right"
        android:layout_marginRight="127dp"
        android:layout_marginBottom="73dp"
        android:background="#66FFFFFF" />

    <!-- Obszar sterowania góra -->
    <View
        android:id="@+id/controlTop"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_gravity="bottom"
        android:layout_marginStart="15dp"
        android:layout_marginLeft="15dp"
        android:layout_marginBottom="123dp"
        android:background="#66FFFFFF" />

    <!-- Obszar sterowania dół -->
    <View
        android:id="@+id/controlDown"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_gravity="bottom"
        android:layout_marginStart="15dp"
        android:layout_marginLeft="15dp"
        android:layout_marginBottom="17dp"
        android:background="#66FFFFFF" />

    <!-- Plansza -->
    <GridLayout
        android:id="@+id/boardGridLayout"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="top|left"
        android:layout_margin="0dp"
        android:layout_marginLeft="0dp"
        android:layout_marginTop="0dp"
        android:layout_marginRight="0dp"
        android:layout_marginBottom="0dp"
        android:foregroundGravity="top|left"
```

```

        android:padding="0dp"
        android:paddingLeft="0dp"
        android:paddingTop="0dp"
        android:paddingRight="0dp"
        android:paddingBottom="0dp"
        android:visibility="visible"
        tools:visibility="visible" />
<!-- Rockford -->
<ImageView
    android:id="@+id/sprite_view"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:layout_gravity="center_horizontal|fill"
    android:contentDescription="TODO" />
</FrameLayout>

```

Proces tworzenia planszy boardGridLayout w tej aplikacji polega na dynamicznym generowaniu widoków ImageView i ich dodawaniu do GridLayout. Plansza ta jest reprezentowana przez dwuwymiarową tablicę mapa[], gdzie każdy element tablicy odpowiada jednemu polu na planszy, a jego zawartość decyduje o wyświetlanej grafice na danym polu.

Kroki procesu tworzenia planszy boardGridLayout:

1. Inicjalizacja GridLayout: Na początku tworzymy obiekt GridLayout, który będzie reprezentować planszę gry. W tym przypadku jest to dynamicznie tworzony obiekt GridLayout, a nie zdefiniowany w pliku XML.
2. Ustawienie parametrów planszy: Ustawiamy parametry planszy, takie jak liczba kolumn i wierszy, zgodnie z rozmiarem tablicy mapa[][].
3. Tworzenie i dodawanie widoków ImageView: W pętlach iterujemy po elementach tablicy mapa[][]. Dla każdego elementu tworzymy odpowiedni widok ImageView, reprezentujący grafikę odpowiadającą temu polu na planszy. W zależności od zawartości elementu tablicy, ustawiamy odpowiednią grafikę (np. ścianę, ziemię, postać itp.).
4. Ustawianie parametrów widoków: W zależności od potrzeb możemy ustawić dodatkowe parametry widoków ImageView, takie jak rozmiar, marginesy, tło itp.
5. Dodanie widoków do GridLayout: Każdy widok ImageView, reprezentujący pole na planszy, jest dodawany do obiektu GridLayout za pomocą metody addView(). Dzięki temu plansza jest dynamicznie generowana na podstawie danych zawartych w tablicy mapa[][].
6. Dodanie GridLayout do layoutu głównego: Na koniec plansza boardGridLayout jest dodawana do głównego layoutu (FrameLayout) za pomocą metody addView(), co pozwala na jej wyświetlenie na ekranie.

Proces ten pozwala na dynamiczne tworzenie planszy gry na podstawie danych zawartych w tablicy i wyświetlanie jej na ekranie aplikacji. Jest to przydatna technika szczególnie w przypadku gier, gdzie plansze mogą być różnorodne i zmieniają się w trakcie rozgrywki.

## MainActivity.java

```

package com.example.simpleanimation1;

import androidx.appcompat.app.AppCompatActivity;

import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.Bundle;
import android.os.Handler;
import android.view.MotionEvent;
import android.view.View;
import android.widget.FrameLayout;
import android.widget.GridLayout;
import android.widget.ImageView;
import android.widget.RelativeLayout;

public class MainActivity extends AppCompatActivity {
    private enum RockfordBo {stand, idle, left, right, appear, die};

    private int frameCount = 7; // Ilość klatek w spritesheetcie
    private int framewidth; // szerokość pojedynczej klatki

    private GridLayout boardGridLayout; // GridLayout dla planszy
    private Character mapa[][]={

{'b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b'},
{'b','g','s','g','g','s','d','g','w','g','s','g','g','g','s','g','w','g','g','g','e','g','s','w','g','g','e','g','g','b'},
{'b','g','g','g','g','g','g','g','w','g','g','g','g','g','g','g','w','g','g','g','e','g','g','w','g','g','g','g','g','b'},

```

```
{'b','e','e','e','e','e','e','s','e','e','s','e','e','e','e','s','e','s','s','e','e','e','e','e','s','e','e','s','g','b'},
{'b','d','g','g','g','g','g','g','w','g','s','g','g','g','g','s','w','g','s','g','e','g','g','w','g','g','g','s','d','b'},
{'b','g','g','g','g','g','g','m','w','g','s','g','g','g','g','s','w','g','s','g','e','g','g','w','g','g','g','s','d','b'},
{'b','w','w','w','w','w','w','w','w','w','w','w','w','w','w','w','w','g','w','w','e','w','w','w','w','w','w','w','w','w','b'},
{'b','g','s','g','g','s','d','g','w','g','s','g','g','g','g','s','g','w','g','g','g','e','g','s','w','g','g','e','g','g','b'},
{'b','g','g','g','g','g','g','g','w','g','g','g','g','g','g','g','g','w','g','g','g','e','g','g','w','g','g','g','g','g','b'},
{'b','e','e','e','e','e','e','e','e','e','e','e','e','e','e','e','s','e','s','s','e','e','e','e','e','e','s','e','e','s','g','b'},
{'b','m','g','g','g','g','g','g','s','w','g','s','g','g','g','g','s','w','g','s','g','e','g','g','w','g','g','g','s','d','b'},
{'b','g','g','g','d','g','s','s','w','g','s','g','g','g','g','s','w','g','s','g','e','g','g','w','g','g','g','s','d','b'},
{'b','w','w','w','w','w','w','w','w','w','w','w','w','w','w','w','w','g','w','w','e','w','w','w','w','w','w','w','w','w','b'},
{'b','g','s','g','g','s','d','g','w','g','s','g','g','g','g','s','g','w','g','g','g','e','g','s','w','g','g','e','g','g','b'},
{'b','g','g','g','g','g','g','g','w','g','g','g','g','g','g','g','g','w','g','g','g','e','g','g','w','g','g','g','g','g','b'},
{'b','e','e','e','e','e','e','e','e','e','e','e','e','e','e','e','s','e','s','s','e','e','e','e','e','e','s','e','e','s','g','b'},
{'b','g','g','g','g','g','g','g','w','g','s','g','g','g','g','s','w','g','s','g','e','g','g','w','g','g','g','s','d','b'},
{'b','g','g','g','g','g','g','g','m','w','g','s','g','g','g','g','s','w','g','s','g','e','g','g','w','g','g','g','s','d','b'},
{'b','w','w','w','w','w','w','w','w','w','w','w','w','w','w','w','w','g','w','w','e','w','w','w','w','w','w','w','w','w','b'},
{'b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b','b'},
}; //mapa
```

```
private ImageView imageView;
private int frameIndex = 0;
private Handler handler;
private Bitmap spritesheet_rockford_stand;
private Bitmap spritesheet_rockford_idle;
private Bitmap spritesheet_rockford_left;
private Bitmap spritesheet_rockford_right;
private RockfordDo rockfordDo = RockfordDo.stand;
private int idleTimer=0;
private int idleTime=40;
private int scale = 1; // współczynnik powiększenia
private float imageX=88, imageY=88; // współrzędne Rockforda
private boolean movingUp, movingDown, movingLeft, movingRight;
private float deltaX = 88;
private float deltaY = 88;

// Definicja obszarów sterowania
private View topControl, bottomControl, leftControl, rightControl;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    FrameLayout mainLayout = findViewById(R.id.mainLayout); // Pobieramy RelativeLayout z layoutu
    mainLayout.setBackgroundColor(getResources().getColor(android.R.color.black)); // Ustawiamy czarne tło
    // Inicjalizacja planszy w GridLayout
    initBoardGridLayout();

    imageView = findViewById(R.id.sprite_view);
    handler = new Handler();
    spritesheet_rockford_stand = BitmapFactory.decodeResource(getResources(), R.drawable.rockford_stand_32x32);
    spritesheet_rockford_idle = BitmapFactory.decodeResource(getResources(), R.drawable.rockford_idle_32x32);
    spritesheet_rockford_left = BitmapFactory.decodeResource(getResources(), R.drawable.rockford_left_32x32);
    spritesheet_rockford_right = BitmapFactory.decodeResource(getResources(), R.drawable.rockford_right_32x32);

    // obliczamy szerokość pojedynczej klatki
    framewidth = spritesheet_rockford_stand.getWidth() / frameCount;

    // Ustawiamy szerokość ImageView na szerokość pojedynczej klatki, pomnożoną przez współczynnik powiększenia
    imageView.setLayoutParams().width = framewidth * scale;
    imageView.setLayoutParams().height = spritesheet_rockford_stand.getHeight() * scale; // Ustawiamy wysokość
    // proporcjonalnie do powiększenia
    // imageView.requestLayout();
    // Inicjalizacja obszarów sterowania
    topControl = findViewById(R.id.controlTop);
    bottomControl = findViewById(R.id.controlDown);
    leftControl = findViewById(R.id.controlLeft);
    rightControl = findViewById(R.id.controlRight);

    // Ustawiamy obszary sterowania jako dotykowe
    setTouchListeners();

    // Uruchamiamy animację
    startAnimation();
}

private void initBoardGridLayout() {
    // Tworzymy planszę w GridLayout
    boardGridLayout = new GridLayout(this);
    boardGridLayout.setColumnCount(mapa[0].length); // liczba kolumn
    boardGridLayout.setRowCount(mapa.length); // liczba wierszy
    // Tworzymy obiekty ImageView i dodajemy je do planszy
    for (int i = 0; i < mapa.length; i++) {
        for (int j = 0; j < mapa[0].length; j++) {
            ImageView imageView = new ImageView(this);
            GridLayout.LayoutParams params = new GridLayout.LayoutParams();
            //params.height = 64;
            //params.width = 64;
            //imageView.setLayoutParams(params);
            switch (mapa[i][j]){
                case 'b':
                    imageView.setImageResource(R.drawable.border_32x32);
                    break;
                case 'w':

```

```

        imageView.setImageResource(R.drawable.wa11_32x32);
        break;
    case 's':
        imageView.setImageResource(R.drawable.stone_32x32);
        break;
    case 'g':
        imageView.setImageResource(R.drawable.ground_32x32);
        break;
    case 'e':
        imageView.setImageResource(R.drawable.empty_32x32);
        break;
    case 'm':
        imageView.setImageResource(R.drawable.mob_32x32);
        break;
    case 'd':
        imageView.setImageResource(R.drawable.diamond_32x32);
        break;
    case 'x':
        imageView.setImageResource(R.drawable.butterfly_32x32);
        break;
    }
    boardGridLayout.addView(imageView);
}
}

// Pobieramy RelativeLayout z layoutu
FrameLayout mainLayout = findViewById(R.id.mainLayout);

// Dodajemy planszę do RelativeLayout
mainLayout.addView(boardGridLayout,0);
}

private void setTouchListeners() {
    topControl.setOnTouchListener(new View.OnTouchListener() {
        @Override
        public boolean onTouch(View v, MotionEvent event) {
            switch (event.getAction()) {
                case MotionEvent.ACTION_DOWN:
                    movingUp = true;
                    idleTimer=0;
                    break;
                case MotionEvent.ACTION_UP:
                    movingUp = false;
                    rockfordDo=RockfordDo.stand;
                    break;
            }
            return true;
        }
    });

    bottomControl.setOnTouchListener(new View.OnTouchListener() {
        @Override
        public boolean onTouch(View v, MotionEvent event) {
            switch (event.getAction()) {
                case MotionEvent.ACTION_DOWN:
                    movingDown = true;
                    idleTimer=0;
                    break;
                case MotionEvent.ACTION_UP:
                    movingDown = false;
                    rockfordDo=RockfordDo.stand;
                    break;
            }
            return true;
        }
    });

    leftControl.setOnTouchListener(new View.OnTouchListener() {
        @Override
        public boolean onTouch(View v, MotionEvent event) {
            switch (event.getAction()) {
                case MotionEvent.ACTION_DOWN:
                    movingLeft = true;
                    idleTimer=0;
                    break;
                case MotionEvent.ACTION_UP:
                    movingLeft = false;
                    rockfordDo=RockfordDo.stand;
                    break;
            }
            return true;
        }
    });

    rightControl.setOnTouchListener(new View.OnTouchListener() {
        @Override
        public boolean onTouch(View v, MotionEvent event) {
            switch (event.getAction()) {
                case MotionEvent.ACTION_DOWN:
                    movingRight = true;
                    idleTimer=0;
                    break;
                case MotionEvent.ACTION_UP:
                    movingRight = false;
                    rockfordDo=RockfordDo.stand;
                    break;
            }
            return true;
        }
    });
}

private void moveRockford() {
    if (movingUp) {
        // Przesunięcie postaci w górę
        imageY -= deltaY;
        rockfordDo = RockfordDo.left;
    }
    if (movingDown) {
        // Przesunięcie postaci w dół

```

```

        imageY += deltaY;
        rockfordDo = RockfordDo. left;
    }
    if (movingLeft) {
        // Przesunięcie postaci w lewo
        imageX -= deltaX;
        rockfordDo = RockfordDo. left;
    }
    if (movingRight) {
        // Przesunięcie postaci w prawo
        imageX += deltaX;
        rockfordDo = RockfordDo. right;
    }
    imageView.setX(imageX);
    imageView.setY(imageY);
}

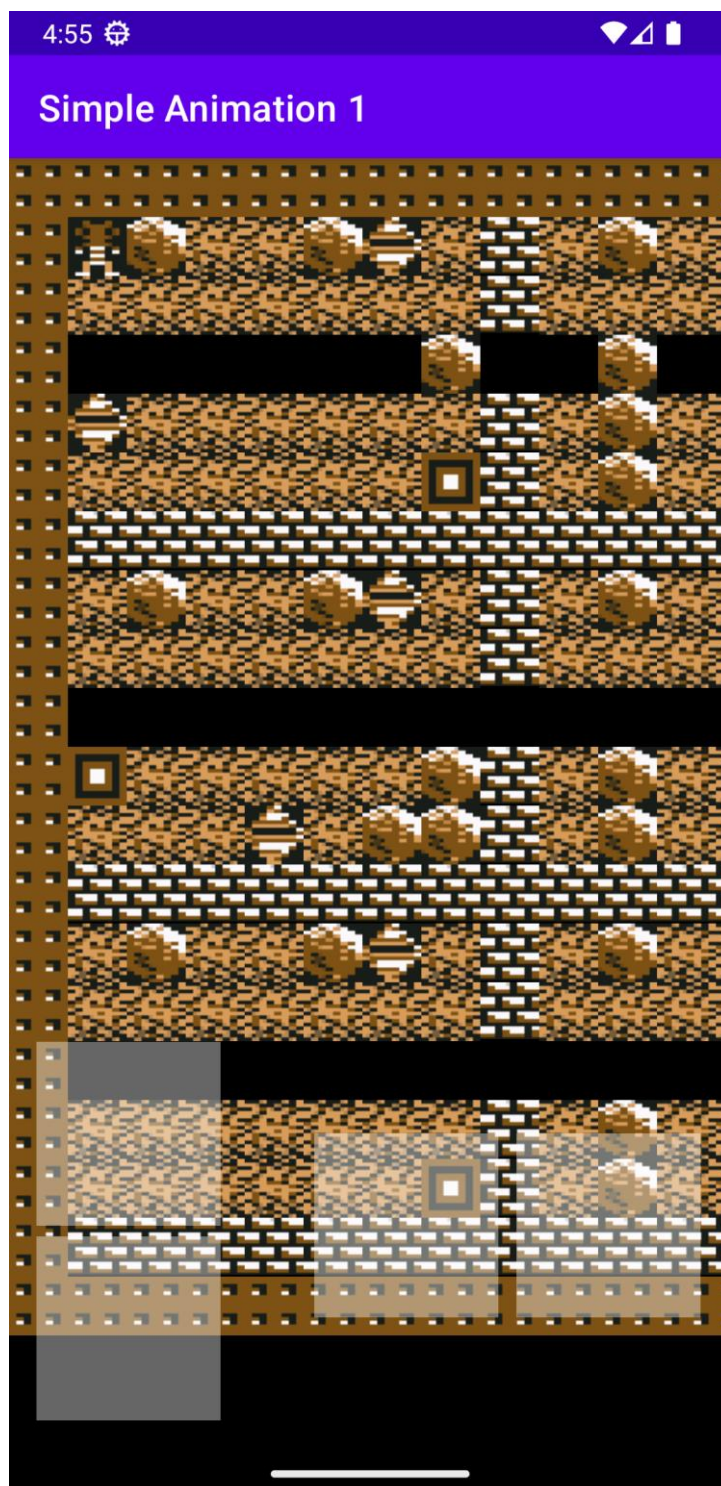
private void startAnimation() {
    Runnable animationRunnable = new Runnable() {
        @Override
        public void run() {
            // ustawiamy kolejną klatkę animacji
            // wycinamy fragment z spritesheeta odpowiadający danej klatce i ustawiamy w ImageView
            Bitmap frame;
            switch (rockfordDo){
                case stand:
                    frame = Bitmap.createBitmap(spritesheet_rockford_stand, frameIndex * framewidth, 0, framewidth,
spritesheet_rockford_stand.getHeight());
                    frame = Bitmap.createScaledBitmap(frame, framewidth * scale, spritesheet_rockford_stand.getHeight() * scale,
true);
                    moveRockford();
                    idleTimer++;
                    imageView.setImageBitmap(frame);
                    if (idleTimer>20) {rockfordDo=RockfordDo.idle;}
                    break;
                case idle:
                    frame = Bitmap.createBitmap(spritesheet_rockford_idle, frameIndex * framewidth, 0, framewidth,
spritesheet_rockford_idle.getHeight());
                    frame = Bitmap.createScaledBitmap(frame, framewidth * scale, spritesheet_rockford_idle.getHeight() * scale,
true);
                    moveRockford();
                    imageView.setImageBitmap(frame);
                    break;
                case right:
                    frame = Bitmap.createBitmap(spritesheet_rockford_right, frameIndex * framewidth, 0, framewidth,
spritesheet_rockford_right.getHeight());
                    frame = Bitmap.createScaledBitmap(frame, framewidth * scale, spritesheet_rockford_right.getHeight() * scale,
true);
                    moveRockford();
                    imageView.setImageBitmap(frame);
                    idleTimer=0;
                    break;
                case left:
                    frame = Bitmap.createBitmap(spritesheet_rockford_left, frameIndex * framewidth, 0, framewidth,
spritesheet_rockford_left.getHeight());
                    frame = Bitmap.createScaledBitmap(frame, framewidth * scale, spritesheet_rockford_left.getHeight() * scale,
true);
                    moveRockford();
                    imageView.setImageBitmap(frame);
                    idleTimer=0;
                    break;
            }
            // Inkrementujemy indeks klatki
            frameIndex = (frameIndex + 1) % frameCount;

            // wywołujemy ponownie runnable po pewnym czasie dla następnej klatki
            handler.postDelayed(this, 100); // Ustaw czas opóźnienia między klatkami (tutaj 100ms)
        }
    };
    handler.post(animationRunnable);
}

@Override
protected void onDestroy() {
    super.onDestroy();
    handler.removeCallbacksAndMessages(null);
    spritesheet_rockford_stand.recycle(); // Zwolnij zasoby bitmapy spritesheeta
    spritesheet_rockford_left.recycle(); // Zwolnij zasoby bitmapy spritesheeta
    spritesheet_rockford_right.recycle(); // Zwolnij zasoby bitmapy spritesheeta
}
}

```





Powyższy przykład to implementacja aktywności w aplikacji Android, która wykorzystuje animację do poruszania postacią na ekranie. Krótkie wyjaśnienie tego, co robi ten kod:

1. Inicjalizacja zmiennych: Zdefiniowane są różne zmienne, takie jak `frameCount` (ilość klatek w `spritesheet`), `frameWidth` (szerokość pojedynczej klatki), oraz zmienne do przechowywania danych o położeniu postaci.
2. Metoda `onCreate`: Metoda ta jest wywoływana podczas tworzenia aktywności. W tej metodzie inicjalizowane są widoki, takie jak plansza gry (`GridLayout`) oraz obszary sterowania (`View`), a także ustawiane są nasłuchiwanie dotyku dla obszarów sterowania.
3. Metoda `initBoardGridLayout`: Ta metoda tworzy planszę gry na podstawie dwuwymiarowej tablicy `mapa[][]`. Każdy element tablicy odpowiada jednemu polu planszy, a jego zawartość decyduje o wyświetlanej grafice.

4. Metoda `setTouchListeners`: Metoda ta ustawia nasłuchiwanie dotyku dla obszarów sterowania. W zależności od akcji dotyku (naciśnięcie lub zwolnienie przycisku), ustawiane są odpowiednie flagi informujące o ruchu postaci.
5. Metoda `moveRockford`: Ta metoda jest odpowiedzialna za przemieszczenie postaci na planszy w zależności od naciśniętych klawiszy.
6. Metoda `startAnimation`: Ta metoda uruchamia animację postaci. Wykorzystuje ona obiekt `Handler`, aby cyklicznie zmieniać klatki animacji i przemieszczać postać na planszy.
7. Metoda `onDestroy`: Metoda ta jest wywoływana, gdy aktywność zostanie zniszczona. W tej metodzie usuwane są wszelkie zadania cykliczne związane z animacją oraz zwalniane są zasoby bitmap.

Zaprezentowany przykład to podstawowa implementacja animacji postaci na planszy w aplikacji Android. Należy go dalej rozwijać, dodając funkcjonalności:

1. Poruszanie po mapie powoduje „zjadanie” ziemi (`ground`), pozostaje czarne pole (`empty`).
2. Kolizje z `border` i `wall` powodują, że `Rockford` nie może dalej się poruszać.
3. Kolizje z `mob` i `butterfly` powodują, że `Rockford` traci życie,
4. Kolizja z diamentem powoduje, że diament znika (liczba punktów się zwiększa).
5. Diament, `mob` i motyl są animowane (można użyć takiej samej metody jak animacja `Rockforda`).
6. Kamienie pod którymi zniknie ziemia spadają (spadający kamień zabija `Rockforda`)
7. Przy zbliżaniu się `Rockforda` do krawędzi ekranu `Plansza` (`GridLayout`) się przesuwają (teraz widać tylko część mapy)
8. Dźwięki:
  - a. poruszanie `Rockfordem` – `krok`, `krok2`
  - b. plansza tytułowa – `main_theme`
  - c. zebranie diamentu – `diamond_collect`
  - d. upadający diament – `diamond_fall`
  - e. zebranie wszystkich diamentów – `exit_open`
  - f. `Rockford` pojawia się na planszy – `rockford_appear`
  - g. Rysowanie planszy z grą – `start_game`
  - h. Upadający kamień – `stone_fall`
  - i. Koniec czasu – `time_up`