|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 | 6 | Data wykonania |  | Grupa |  |
| Ocena |  | Data oddania |  | Imię Nazwisko |  |
| Nazwa ćwiczenia | Pętla gry, animacja, kolizje, obsługa dotyku | | |

**Cel ćwiczenia laboratoryjnego**

Celem tego ćwiczenia jest stworzenie gry na platformie Android, obejmującej pętlę gry, animacje, tło, obsługę zdarzeń oraz sterowanie postacią. Uczestnik będzie miał okazję zaznajomić się z różnymi aspektami programowania gier mobilnych.

**Zadanie**

Studenci zostaną poproszeni o utworzenie gry, którą będzie rozszerzeniem i połączeniem przykładów zawartych w ćwiczeniu oraz zastosowaniem wiedzy zdobytej na poprzednich zajęciach.

**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 tło do sceny gry, nadając jej atrakcyjny wygląd.
3. Tworzyć animacje, 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 pętle gry należy uzupełnić rozgrywkę inspirując się klasyczną grą Arkanoid. Jest to to gra arkadowa, gdzie gracz steruje paletką, odbijając piłkę w kierunku bloków, próbując zniszczyć je wszystkie. Zadaniem gracza jest utrzymanie piłki w grze, unikając utraty życia poprzez przejście przez dolną krawędź ekranu, a gracz zdobywa punkty za każdy zniszczony blok. Gra oferuje różnorodne poziomy, bonusy i stanowi klasyczne wyzwanie zręcznościowe.

**Zasady gry**

Celem gry Arkanoid jest zniszczenie wszystkich bloków na ekranie za pomocą piłki i paletki. Gracz kontroluje paletkę, poruszając nią w lewo i w prawo po dolnej krawędzi ekranu. Piłka zaczyna grę na paletce i jest odbijana w górę w kierunku bloków. Ekran w grze Arkanoid jest wypełniony różnymi kolorowymi blokami ułożonymi w różnych konfiguracjach. Bloki mają różne właściwości, takie jak trwałość, co oznacza, ile uderzeń piłką są potrzebne do zniszczenia bloku. Gracz musi używać paletki, aby odbijać piłkę i kierować ją w stronę bloków, próbując zniszczyć jak najwięcej bloków. Piłka odbija się od ścian bocznych i górnej krawędzi ekranu. Jeśli piłka dotknie dolnej krawędzi ekranu, gracz traci jedno życie (jeśli ograniczone) lub kończy grę (jeśli życia się skończą). Niektóre bloki mogą zawierać bonusy, takie jak rozszerzenia paletki, dodatkowe piłki, przyspieszenia itp. Gracz może zbierać te bonusy, ułatwiając sobie zadanie w niszczeniu bloków. Gry typu Arkanoid często składają się z kilku poziomów, z różnymi układami bloków na każdym poziomie. Po zniszczeniu wszystkich bloków na danym poziomie, gracz przechodzi do kolejnego. Gracz kończy grę, gdy straci wszystkie życia, czyli gdy piłka przeleci dolną krawędź ekranu, a gracz nie zdąży jej odbić. Celem jest uzyskanie jak najwyższego wyniku poprzez zniszczenie jak największej liczby bloków.

**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.
* 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.

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

Oto 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.

**Petal 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 prostego środowiska gry:

**GameView.java**

Najpierw tworzysz klasę GameView, która rozszerza SurfaceView i implementuje Runnable:

package com.example.simplegameloop;  
import android.content.Context;  
import android.graphics.Canvas;  
import android.graphics.Color;  
import android.graphics.Paint;  
import android.view.SurfaceHolder;  
import android.view.SurfaceView;  
import androidx.core.content.ContextCompat;

public class GameView extends SurfaceView implements Runnable {  
  
 private Thread gameThread = null;  
 private SurfaceHolder surfaceHolder;  
 private boolean running = false;  
 private Canvas canvas;  
 private Paint paint;  
 private Context context;  
  
 public GameView(Context context) {  
 super(context);  
 this.context = context;  
 surfaceHolder = getHolder();  
 paint = new Paint();  
 }  
 @Override  
 public void run() {  
 while (running) {  
 if (surfaceHolder.getSurface().isValid()) {  
 canvas = surfaceHolder.lockCanvas();  
 canvas.drawColor(Color.*BLACK*); *// Tło ekranu  
 // Tutaj możesz dodać rysowanie elementów gry* Paint paint = new Paint();  
 int color = ContextCompat.*getColor*(context, R.color.*white*);  
 paint.setColor(color);  
 paint.setTextSize(50);  
 canvas.drawText("Game status " + running, 100, 100, paint);  
  
 surfaceHolder.unlockCanvasAndPost(canvas);  
 }  
 try {  
 Thread.*sleep*(16); *// Około 60 klatek na sekundę* } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 public void pause() {  
 running = false;  
 try {  
 gameThread.join();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void resume() {  
 running = true;  
 gameThread = new Thread(this);  
 gameThread.start();  
 }  
}

**MainActivity.java**

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

package com.example.simplegameloop;  
import androidx.appcompat.app.AppCompatActivity;  
import android.os.Bundle;  
public class MainActivity extends AppCompatActivity {  
 private GameView gameView;  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 gameView = new GameView(this);  
 setContentView(gameView);  
 }  
 @Override  
 protected void onPause() {  
 super.onPause();  
 gameView.pause();  
 }  
 @Override  
 protected void onResume() {  
 super.onResume();  
 gameView.resume();  
 }  
}

Aby dodać uprawnienie WAKE\_LOCK do Twojej aplikacji Android, 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 rozwagą, 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.

Obraz zawierający zrzut ekranu, tekst

Opis wygenerowany automatycznie

Spróbuj zmieniać status aplikacji za pomocą funkcji telefonu i obserwuj, jak zmienia się aplikacja.

Następny przykład zilustruje jak do powyższej pętli gry dodać poruszający się obiekt (piłka), który będzie się odbijał od ścian.

package com.example.simplegameloop;  
  
import android.content.Context;  
import android.graphics.Canvas;  
import android.graphics.Color;  
import android.graphics.Paint;  
import android.view.SurfaceHolder;  
import android.view.SurfaceView;  
  
import androidx.core.content.ContextCompat;  
  
public class GameView extends SurfaceView implements Runnable, SurfaceHolder.Callback {  
  
 private Thread gameThread = null;  
 private SurfaceHolder surfaceHolder;  
 private boolean running = false;  
 private Canvas canvas;  
 private Paint paint;  
 private Context context;  
  
 private float ballX, ballY; *// Położenie piłki* private float ballSpeedX, ballSpeedY; *// Prędkość piłki* private float ballRadius = 30; *// Promień piłki* public GameView(Context context) {  
 super(context);  
 this.context = context;  
 surfaceHolder = getHolder();  
 paint = new Paint();  
  
 *// Dodaj callback do obsługi zdarzeń powierzchni* surfaceHolder.addCallback(this);  
 }  
  
 @Override  
 public void surfaceCreated(SurfaceHolder holder) {  
 *// Inicjalizacja położenia i prędkości piłki po utworzeniu powierzchni* ballX = getWidth() / 2f;  
 ballY = getHeight() / 2f;  
 ballSpeedX = 8f;  
 ballSpeedY = 8f;  
  
 *// Rozpocznij wątek po utworzeniu powierzchni* resume();  
 }  
  
 @Override  
 public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {  
 *// Zmiana rozmiaru powierzchni - aktualizacja położenia piłki* ballX = width / 2f;  
 ballY = height / 2f;  
 }  
  
 @Override  
 public void surfaceDestroyed(SurfaceHolder holder) {  
 *// Opcjonalnie zatrzymaj animację lub inne działania przed zniszczeniem powierzchni* pause();  
 }  
  
 @Override  
 public void run() {  
 while (running) {  
 if (surfaceHolder.getSurface().isValid()) {  
 canvas = surfaceHolder.lockCanvas();  
 canvas.drawColor(Color.*BLACK*); *// Tło ekranu  
  
 // Rysuj piłkę* paint.setColor(ContextCompat.*getColor*(context, R.color.*white*));  
 canvas.drawCircle(ballX, ballY, ballRadius, paint);  
  
 *// Aktualizuj położenie piłki* updateBallPosition();  
  
 surfaceHolder.unlockCanvasAndPost(canvas);  
 }  
  
 try {  
 Thread.*sleep*(16); *// Około 60 klatek na sekundę* } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 private void updateBallPosition() {  
 *// Aktualizuj położenie piłki na podstawie prędkości* ballX += ballSpeedX;  
 ballY += ballSpeedY;  
  
 *// Odbij piłkę od ścian ekranu* if (ballX - ballRadius < 0 || ballX + ballRadius > getWidth()) {  
 ballSpeedX = -ballSpeedX;  
 }  
 if (ballY - ballRadius < 0 || ballY + ballRadius > getHeight()) {  
 ballSpeedY = -ballSpeedY;  
 }  
 }  
  
 public void pause() {  
 running = false;  
 try {  
 gameThread.join();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void resume() {  
 running = true;  
 gameThread = new Thread(this);  
 gameThread.start();  
 }  
}

Teraz, u dołu można dodać paletkę i stworzyć grę typu Arkanoid, oczywiście jeszcze w bardzo podstawowej wersji:

package com.example.simplegameloop;  
  
import android.content.Context;  
import android.graphics.Canvas;  
import android.graphics.Color;  
import android.graphics.Paint;  
import android.view.MotionEvent;  
import android.view.SurfaceHolder;  
import android.view.SurfaceView;  
  
import androidx.core.content.ContextCompat;  
  
public class GameView extends SurfaceView implements Runnable, SurfaceHolder.Callback {  
  
 private Thread gameThread = null;  
 private SurfaceHolder surfaceHolder;  
 private boolean running = false;  
 private Canvas canvas;  
 private Paint paint;  
 private Context context;  
  
 private float ballX, ballY; *// Położenie piłki* private float ballSpeedX, ballSpeedY; *// Prędkość piłki* private float ballRadius = 30; *// Promień piłki* private float paddleX, paddleY; *// Położenie paletki* private float paddleWidth = 200; *// Szerokość paletki* private float paddleHeight = 20; *// Wysokość paletki* private float paddleSpeed = 10; *// Prędkość paletki* public GameView(Context context) {  
 super(context);  
 this.context = context;  
 surfaceHolder = getHolder();  
 paint = new Paint();  
  
 *// Dodaj callback do obsługi zdarzeń powierzchni* surfaceHolder.addCallback(this);  
 }  
  
 @Override  
 public void surfaceCreated(SurfaceHolder holder) {  
 *// Inicjalizacja położenia i prędkości piłki oraz paletki* ballX = getWidth() / 2f;  
 ballY = getHeight() / 2f;  
 ballSpeedX = 5f;  
 ballSpeedY = 5f;  
  
 paddleX = (getWidth() - paddleWidth) / 2f;  
 paddleY = getHeight() - 2 \* paddleHeight; *// Umieść paletkę na dole ekranu  
  
 // Rozpocznij wątek po utworzeniu powierzchni* resume();  
 }  
  
 @Override  
 public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {  
 *// Zmiana rozmiaru powierzchni - aktualizacja położenia piłki i paletki* ballX = width / 2f;  
 ballY = height / 2f;  
  
 paddleX = (width - paddleWidth) / 2f;  
 paddleY = height - 2 \* paddleHeight; *// Umieść paletkę na dole ekranu* }  
  
 @Override  
 public void surfaceDestroyed(SurfaceHolder holder) {  
 *// Opcjonalnie zatrzymaj animację lub inne działania przed zniszczeniem powierzchni* pause();  
 }  
  
 @Override  
 public void run() {  
 while (running) {  
 if (surfaceHolder.getSurface().isValid()) {  
 canvas = surfaceHolder.lockCanvas();  
 canvas.drawColor(Color.*BLACK*); *// Tło ekranu  
  
 // Rysuj paletkę* paint.setColor(ContextCompat.*getColor*(context, R.color.*white*));  
 canvas.drawRect(paddleX, paddleY, paddleX + paddleWidth, paddleY + paddleHeight, paint);  
  
 *// Rysuj piłkę* paint.setColor(ContextCompat.*getColor*(context, R.color.*white*));  
 canvas.drawCircle(ballX, ballY, ballRadius, paint);  
  
 *// Aktualizuj położenie piłki i paletki* updateBallPosition();  
  
 surfaceHolder.unlockCanvasAndPost(canvas);  
 }  
  
 try {  
 Thread.*sleep*(16); *// Około 60 klatek na sekundę* } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 private void updateBallPosition() {  
 *// Aktualizuj położenie piłki na podstawie prędkości* ballX += ballSpeedX;  
 ballY += ballSpeedY;  
  
 *// Odbij piłkę od ścian ekranu* if (ballX - ballRadius < 0 || ballX + ballRadius > getWidth()) {  
 ballSpeedX = -ballSpeedX;  
 }  
 if (ballY - ballRadius < 0 || ballY + ballRadius > getHeight()) {  
 ballSpeedY = -ballSpeedY;  
 }  
  
 *// Odbij piłkę od paletki* if (ballY + ballRadius > paddleY && ballX > paddleX && ballX < paddleX + paddleWidth) {  
 ballSpeedY = -ballSpeedY;  
 }  
 }  
  
 @Override  
 public boolean onTouchEvent(MotionEvent event) {  
 float touchX = event.getX();  
  
 switch (event.getAction()) {  
 case MotionEvent.*ACTION\_MOVE*:  
 *// Przesuń paletkę w lewo lub w prawo w zależności od dotyku* if (touchX > paddleX && touchX < paddleX + paddleWidth) {  
 paddleX = touchX - paddleWidth / 2;  
 }  
 break;  
 }  
  
 return true;  
 }  
  
 public void pause() {  
 running = false;  
 try {  
 gameThread.join();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void resume() {  
 running = true;  
 gameThread = new Thread(this);  
 gameThread.start();  
 }  
}

Obraz zawierający zrzut ekranu, tekst

Opis wygenerowany automatycznie

Omówiony wyżej projekt obsługi gry dołączony jest do instrukcji ćwiczenia (Simple Game Loop). Zaprezentowana i dołączona do ćwiczenia gra nie obsługuje warunku zwycięstwa i wymaga implementacji tej funkcjonalności. Należy połączyć szkielet obsługi gry (zadanie z poprzedniego ćwiczenia), dodać dźwięki, menu, wyniki, wybór poziomu trudności i inne elementy, które występują w znanej grze Arkanoid.