Федеральное агентство связи Сибирский государственный университет телекоммуникаций и информатики Кафедра прикладной математики и кибернетики (ПМ и К)

Лабораторная работа «Жуки»

по дисциплине "Программирование для мобильных устройств"

Выполнил:

студентка ИВТ, группы ИП-813

Захарова К.Ю.

Проверила:

Ассистент кафедры ПМиК

Павлова У.В.

Новосибирск, 2021 г.

Оглавление

[Задание 3](#_Toc67656391)

[Теория 3](#_Toc67656392)

[Реализация приложения 4](#_Toc67656393)

[Код программы 4](#_Toc67656394)

# Задание

Создайте игру "ЖУК". Жуки бегают по экрану. Игроку предлагается при помощи touchScreen-a уничтожить как можно большее число жуков. Обработка отдельного жука должна производиться в отдельном потоке. За каждый промах игроку начисляется штраф. Предусмотреть несколько видов насекомых. Попадание и промах должны иметь звуковое сопровождение. По окончанию игры выводятся результаты.

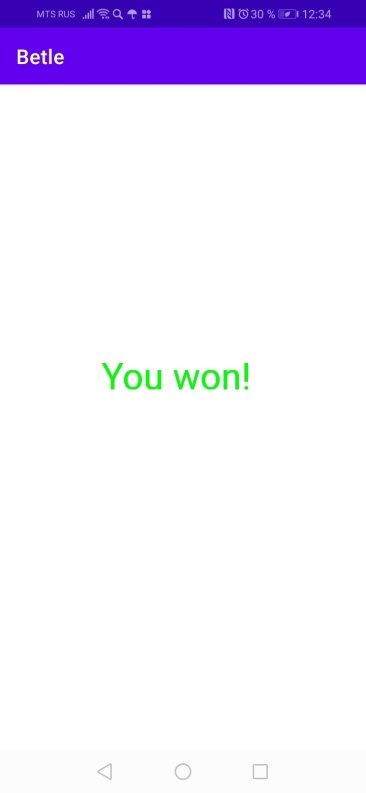
# Теория

При работе с 2D графикой в Android отрисовку можно выполнять используя Canvas. Проще всего это сделать с помощью своего класса, унаследованного от View. Необходимо просто описать метод onDraw(), и использовать предоставленный в качестве параметра canvas для выполнения всех необходимых действий. Однако этот подход имеет свои недостатки. Метод onDraw() вызывается системой. Вручную же можно использовать метод invalidate(), говорящий системе о необходимости перепрорисовки. Но вызов invalidate() не гарантирует незамедлительного вызова метода onDraw(). Поэтому, если нам необходимо постоянно делать отрисовку (например для какой-либо игры), вышеописанный способ вряд ли стоит считать подходящим.

Особенность класса SurfaceView заключается в том, что он предоставляет отдельную область для рисования, действия с которой должны быть вынесены в отдельный поток приложения. Таким образом, приложению не нужно ждать, пока система будет готова к отрисовке всей иерархии view-элементов. Вспомогательный поток может использовать canvas нашего SurfaceView для отрисовки с той скоростью, которая необходима.  
  
Вся реализация сводится к двум основным моментам:

1. Создание класса, унаследованного от SurfaceView и реализующего интерфейс SurfaceHolder.Callback
2. Создание потока, который будет управлять отрисовкой.

# Реализация приложения



# Код программы

MainActivity.java

**package** com.example.betle.activity;  
  
**import** androidx.appcompat.app.AppCompatActivity;  
**import** androidx.annotation.RequiresApi;  
**import** android.os.Bundle;  
**import** android.os.Build;  
  
**import** com.example.betle.core.GameView;  
  
**public class** MainActivity **extends** AppCompatActivity {  
 *//аннотированный элемент должен вызываться только  
 // на заданном уровне APi или выше* @RequiresApi(api = Build.VERSION\_CODES.***N***)  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(**new** GameView (**this**));  
 }  
}

GameMainThread.java

**package** com.example.betle.core;  
  
**import** android.graphics.Canvas;  
**import** android.os.Build;  
**import** android.view.SurfaceHolder;  
**import** androidx.annotation.RequiresApi;  
**import** java.util.Objects;  
  
@RequiresApi(api = Build.VERSION\_CODES.***N***)  
**public class** GameMainThread **extends** Thread {  
 **private** GameView **view**;  
 **private final long maxSleepTime**;*// чтобы не перенагревался телефон* **public** GameMainThread(GameView view) {  
 **this**(view, 30);  
 }  
  
 **public** GameMainThread(GameView view, **int** framesPerSecond) {  
 **super**();  
 **this**.**view** = Objects.*requireNonNull*(view);*//ограждаем, чтобы не было нулевых элементов* **if** (framesPerSecond > 120) {*//условия если бы нажимаем сразу на несколько* **throw new** RuntimeException(**"Too high FPS for Game Thread"**);  
 } **else if** (framesPerSecond < 1) {  
 **throw new** RuntimeException(**"FPS can`t be zero or negative"**);  
 }  
 **this**.**maxSleepTime** = 1000 / framesPerSecond;  
 }  
  
 @Override  
 **public void** run() {  
 **long** startTime, sleepTime;  
 Canvas canvas = **null**;  
 SurfaceHolder holder = **null**;  
  
 **while**(**true**) {  
 *//bpvthztv dhtvz dsgjkytybz* startTime = System.*currentTimeMillis*();  
 *//вызываем добавление жуков* **view**.refillEntities();  
  
 **try** {  
 holder = **view**.getHolder();  
 canvas = holder.lockCanvas();  
 **synchronized** (holder) {  
 **view**.onDraw(canvas);  
 }  
 } **finally** {  
 **if** (canvas != **null**) {  
 holder.unlockCanvasAndPost(canvas);  
 }  
 }  
  
  
 sleepTime = **maxSleepTime** - (System.*currentTimeMillis*() - startTime);  
 **try** {  
 **if** (sleepTime < 10) {  
 *sleep*(10);  
 } **else** {  
 *sleep*(sleepTime);  
 }  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
}

GameView.java

**package** com.example.betle.core;  
  
**import** android.content.Context;  
**import** android.content.res.Resources;  
**import** android.graphics.Canvas;  
**import** android.graphics.Color;  
**import** android.graphics.Paint;  
**import** android.media.AudioManager;  
**import** android.media.SoundPool;  
**import** android.os.Build;  
**import** android.view.MotionEvent;  
**import** android.view.SurfaceHolder;  
**import** android.view.SurfaceView;  
  
**import** androidx.annotation.NonNull;  
**import** androidx.annotation.RequiresApi;  
  
**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.Objects;  
**import** java.util.stream.Collectors;  
**import** java.util.stream.IntStream;  
  
**import** com.example.betle.R;  
**import** com.example.betle.core.entity.TapEntity;  
**import** com.example.betle.util.GameStage;  
**import** com.example.betle.util.ScoreCalculator;  
**import** com.example.betle.util.TapEntityBitmapFactory;  
  
@RequiresApi(api = Build.VERSION\_CODES.***N***)  
*//SurfaceView-отдельная область для рисования  
//SurfaceHolder.Callback - создание области, её изменение и разрушении***public class** GameView **extends** SurfaceView **implements** SurfaceHolder.Callback {  
 *//максимальное число картинок на экране 8* **private static final int *MAX\_ENTITIES*** = 8;  
  
 **private final** Thread **thread**;  
 **private final** Resources **resources**;  
  
 **private final** Paint **defaultPaint** = **new** Paint();  
 **private final** Paint **wonPaint** = **new** Paint();  
 **private final** Paint **lostPaint** = **new** Paint();  
  
 **private final** SurfaceHolder **surfaceHolder**;  
 **private final** List<TapEntity> **tapEntities**;  
 **private** ScoreCalculator **scoreCalculator** = **new** ScoreCalculator(0, 2000, -100);  
 **private** GameStage **currentGameStage** = GameStage.***IN\_PROGRESS***;  
 **private** SoundPool **soundPool**;  
 **private int failSound**, **successSound**;  
  
  
 **public** GameView(Context context) {  
 **super**(context);  
 *//список жуков* **tapEntities** = **new** ArrayList<>();  
 *//работа с полотном для рисования.Именно этот объект будет предоставлять нам canvas  
 // для отрисовки* **surfaceHolder** = Objects.*requireNonNull*(getHolder());  
 **surfaceHolder**.addCallback(**this**);  
 *//синхронизация с потоком отрисовки* **thread** = **new** GameMainThread(**this**, 30);  
 *//чтобы наши ресурсы были не ноль* **resources** = Objects.*requireNonNull*(getResources());  
  
 **defaultPaint**.setColor(Color.***BLUE***);  
 **defaultPaint**.setTextSize(64);  
  
 **wonPaint**.setColor(Color.***GREEN***);  
 **wonPaint**.setTextSize(72);  
  
 **lostPaint**.setColor(Color.***RED***);  
 **lostPaint**.setTextSize(72);  
 *//подключение звука* **soundPool** = **new** SoundPool(5, AudioManager.***STREAM\_MUSIC***, 0);  
 **successSound** = **soundPool**.load(context, R.raw.***two***, 0);  
 **failSound** = **soundPool**.load(context, R.raw.***one***, 0);  
  
 setFocusable(**true**);  
 }  
 *//добавление жуков* **private void** addTapEntity() {  
 **tapEntities**.add(**new** TapEntity(  
 TapEntityBitmapFactory.*getRandom*(**resources**), getWidth(), getHeight()));  
 }  
 *//при игре добавление* **public void** refillEntities() {  
 **while** (**tapEntities**.size() < ***MAX\_ENTITIES***) addTapEntity();  
 }  
  
 **private void** initTapEntities() {  
 IntStream.*range*(0, ***MAX\_ENTITIES***).forEach(i -> addTapEntity());  
 }  
 *//рисовка полотна игры* @Override  
 **protected void** onDraw(Canvas canvas) {  
 canvas.drawColor(Color.***WHITE***);  
  
 **switch** (**currentGameStage**) {  
 **case *IN\_PROGRESS***: {  
 canvas.drawText(String.*format*(**"Score: %d"**, **scoreCalculator**.getScore()),  
 150, 100, **defaultPaint**);  
  
 **tapEntities**.forEach(entity -> {  
 entity.update();  
 entity.draw(canvas);  
 });  
 **break**;  
 }  
 **case *WON***: {  
 canvas.drawText(**"You won!"**, 200, 600, **wonPaint**);  
 **break**;  
 }  
 **case *LOST***: {  
 canvas.drawText(**"You lost..."**, 200, 600, **lostPaint**);  
 **break**;  
 }  
 }  
 }  
 *//создание области для отрисовки* @Override  
 **public void** surfaceCreated(@NonNull SurfaceHolder holder) {  
 initTapEntities();  
 **thread**.start();  
 }  
 *//изменение области* @Override  
 **public void** surfaceChanged(@NonNull SurfaceHolder holder, **int** format, **int** width, **int** height) {  
  
 }  
 *//разрушение области* @Override  
 **public void** surfaceDestroyed(@NonNull SurfaceHolder holder) {  
 **thread**.interrupt();  
 }  
 *//обработка касания* @Override  
 **public boolean** onTouchEvent(MotionEvent event) {  
 **float** x = event.getX();  
 **float** y = event.getY();  
 **if** (event.getAction() == MotionEvent.***ACTION\_DOWN***) {  
 **synchronized** (**surfaceHolder**) {  
 List<TapEntity> tappedEntities = **tapEntities**.stream()  
 .filter(entity -> entity.hasCollision(x, y)).collect(Collectors.*toList*());  
 tappedEntities.forEach(**tapEntities**::remove);  
  
 **if** (tappedEntities.size() > 0) {  
 **soundPool**.play(**successSound**, 1f, 1f, 1, 0, 1);  
 **scoreCalculator**.increment(100 \* tappedEntities.size());  
 } **else** {  
 **soundPool**.play(**failSound**, 1f, 1f, 1, 0, 1);  
 **scoreCalculator**.decrement(50);  
 }  
  
 **if** (**scoreCalculator**.isLose()) {  
 **currentGameStage** = GameStage.***LOST***;  
 } **else if** (**scoreCalculator**.isWin()) {  
 **currentGameStage** = GameStage.***WON***;  
 }  
 }  
 }  
  
 **return true**;  
 }  
  
}

GameStage.java

**package** com.example.betle.util;  
  
**public enum** GameStage {  
 ***IN\_PROGRESS***,  
 ***WON***,  
 ***LOST***}

ScoreCalculator.java

**package** com.example.betle.util;  
  
**public class** ScoreCalculator {  
  
 **private int winScore**;  
 **private int loseScore**;  
 **private volatile int currentScore**;  
  
 **public** ScoreCalculator(**int** initialScore, **int** winScore, **int** loseScore) {  
 **if** (initialScore <= loseScore && initialScore >= winScore) {  
 **throw new** IllegalArgumentException(**"Illegal initial score value"**);  
 }  
 **currentScore** = initialScore;  
 **this**.**winScore** = winScore;  
 **this**.**loseScore** = loseScore;  
 }  
  
 **public void** increment(**int** value) {  
 **currentScore** += value;  
 }  
  
 **public void** decrement(**int** value) {  
 **currentScore** -= value;  
 }  
  
 **public boolean** isWin() {  
 **return currentScore** >= **winScore**;  
 }  
  
 **public boolean** isLose() {  
 **return currentScore** <= **loseScore**;  
 }  
  
 **public int** getScore() {  
 **return currentScore**;  
 }  
  
}

TapEntityBitmapEnum.java

**package** com.example.betle.util;  
  
**public enum** TapEntityBitmapEnum {  
 ***ARBUZ***,  
 ***BURGER***,  
 ***PECHECIE***,  
 ***PICCA***,  
 ***SUP***,  
 ***PIROJN***}

TapEntityBitmapFactory.java

**package** com.example.betle.util;  
  
**import** android.content.res.Resources;  
**import** android.graphics.Bitmap;  
**import** android.graphics.BitmapFactory;  
  
**import** com.example.betle.R;  
  
**import** java.util.Random;  
  
**public class** TapEntityBitmapFactory {  
  
 **private static final** Random ***random*** = **new** Random();  
  
 **public static** Bitmap getRandom(Resources resources) {  
 **return** *get*(TapEntityBitmapEnum.*values*()[***random***.nextInt(TapEntityBitmapEnum.*values*().**length**)], resources);  
 }  
  
 **public static** Bitmap get(TapEntityBitmapEnum element, Resources resources) {  
 **switch** (element) {  
 **case *ARBUZ***: {  
 **return** BitmapFactory.*decodeResource*(resources, R.drawable.***arbuz***);  
 }  
 **case *BURGER***: {  
 **return** BitmapFactory.*decodeResource*(resources, R.drawable.***burger***);  
 }  
 **case *PECHECIE***: {  
 **return** BitmapFactory.*decodeResource*(resources, R.drawable.***pechenie***);  
 }  
 **case *PICCA***: {  
 **return** BitmapFactory.*decodeResource*(resources, R.drawable.***picca***);  
 }  
 **case *SUP***: {  
 **return** BitmapFactory.*decodeResource*(resources, R.drawable.***sup***);  
 }  
 **case *PIROJN***: {  
 **return** BitmapFactory.*decodeResource*(resources, R.drawable.***pirojn***);  
 }  
 **default**:  
 **throw new** IllegalStateException(**"Unexpected value: "** + element);  
 }  
 }  
  
}