ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Лабораторная работа №3

«Жуки»

Выполнил: студент группы ИП-712

Алексеев С.В.

Работу принял: ассистент кафедры ПМиК

Павлова У.В.

Новосибирск 2020

Оглавление

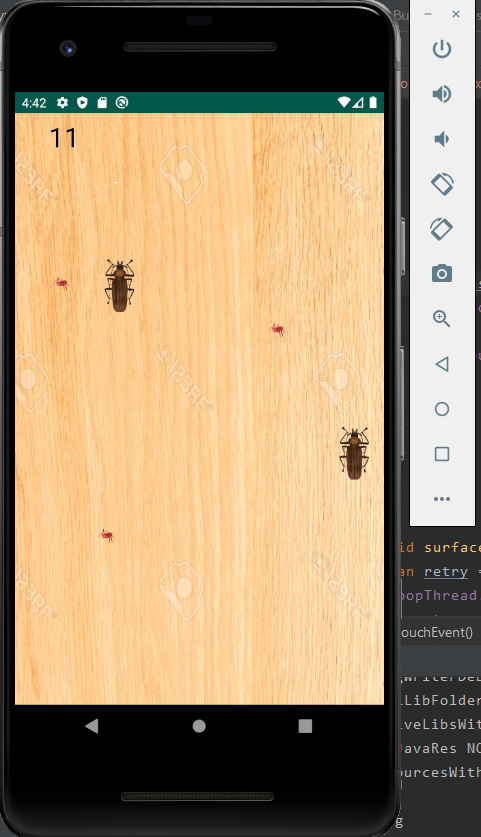
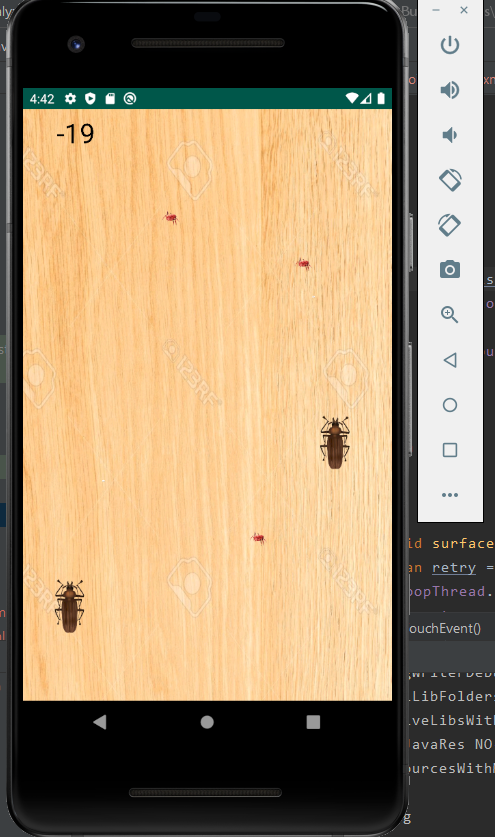
[Цель работы 4](#_Toc40522043)

[Описание программы и главных функций 5](#_Toc40522044)

[Листинг программы 7](#_Toc40522045)

[Скрины 19](#_Toc40522046)

[20](#_Toc40522047)



# Цель работы

Разработать приложение «жуки», где игроку предлагается при помощи touchScreen-a уничтожить как можно большее число жуков.

Реализовано:

* Обработка отдельного жука производится в отдельном потоке.
* Движение объектов(жуков) по экрану
* Объекты имеют разные свойства: ( размер, скорость, количество очков )
* Наличие звука в игре ( попадание, промах )
* Штраф, за промах по объекту
* Подсчет очков

Описание программы и главных функций.

Пользователю необходимо уничтожить как можно большее число движущихся объектов (жуков). За каждый промах по объекту предусмотрен штраф, в виде потери части очков. При попадании на объект – жук умирает, генерируется новый.

Обработка отдельного объекта производится в отдельном потоке:

new Thread(bigbug).start();

new Thread(fastbug).start();

new Thread(fatbug).start();

SurfaceView унаследован от View и является элементом управления, предоставляющим область для рисования (Surface).

класс GameView который будет производить отрисовку нашей графики на сцене и унаследуем его от класса SurfaceView, для того что бы производить.

Цикл игры происходит в отдельном потоке. В одном потоке мы создаем обновления и рисование, а в основном потоке мы обрабатываем события. Для этого создали файл GameManager.java

Работа с изображениями через класс android.graphics.Bitmap

Класс Bitmap не имеет своих методов для рисования, для этого используем метод Canvas.

Задаются позиции(x,y) и параметры ( speed,width,height). Перемещение объекта и его направления реализовано с помощью функции private void update(). Функция void draw(Canvas canvas) отвечает за отрисовку наших спрайтов. Далее с помощью BugGenerator создаем наши объекты указывая к ним параметры .

private static final int FPS = 20; - Наша скорость в м\с = 10

Задаем состояние потока:

private boolean isAlive = false;

void setAlive(boolean statue) {

isAlive = statue;

}

В функции public void run() происходят действия выполняемые в потоке.

public void surfaceDestroyed(SurfaceHolder holder) – уничтожение области рисования.

public void surfaceCreated(SurfaceHolder holder) - Создание области рисования

public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) – изменение области рисования

onTouchEvent() — событие касания к экрану

onTouchEvent(MotionEvent) — вызывается при возникновении события жеста на сенсорном экране.

Использование звука:

Класс SoundPool удобно использовать для проигрывания коротких аудиоклипов.

В нашем случае присутствует два коротких аудиоклипа.

soundPool.load(context, R.raw.*wood1*, 0);  
soundPool.load(context, R.raw.*death*, 0);

Для проигрывания сэмпла используется метод play.

Который имеет параметры: ( soundID, leftVolume, rightVolume, priority, loop, rate)

soundID - переменная с номером сэмпла. Этот номер возвращается в результате выполнения метода load.

В нашем случае soundID 1 soundID 2

Счетчик очков:

Класс Paint содержит информацию о стиле и цвете для рисования геометрии, текста и растровых изображений.

canvas.drawText() — используем для рисования текста на канвасе.

Задаем тип, и наполняем необходимыми свойствами для корректной работы.

canvas.drawText(String.valueOf(view.getScore()), 100, 100, paint);

getScore() получаем значение для счетчика очков.

Score – переменная, отвечающая за сами очки.

Получаем очки за каждый вид объекта(жука) по-разному.

Каждый объект содержит в себе определенное value.

Bigbug value = 1

Fastbug value = 5

Middlesizebug value =3

Предусмотрен штраф:

За каждый промах будет теряться по 10 очков:

if (!isCatched) {

score -=10; … }

# Листинг программы

Bug.java

package com.example.bugs;  
  
import android.graphics.Bitmap;  
import android.graphics.BitmapFactory;  
import android.graphics.Canvas;  
import java.util.Random;  
import android.graphics.Rect;  
import android.media.MediaPlayer;  
  
public class Bug implements Runnable{  
  
 private GameView gameView;  
  
 private Bitmap bmp;  
  
 private int x;  
 private int y;  
  
 private int xSpeed;  
 private int ySpeed;  
  
 private int width;  
 private int height;  
  
 private int value;  
  
 private boolean isAlive = false;  
  
 Bug(GameView gameView, int resource, double scale, int speed, int value)  
 {  
 Bitmap bmp = BitmapFactory.*decodeResource*(gameView.getResources(), resource);  
 bmp = Bitmap.*createScaledBitmap*(bmp, (int)(bmp.getWidth() \* scale), (int)(bmp.getHeight() \* scale), true);  
  
 this.gameView = gameView;  
 this.bmp = bmp;  
 this.width = bmp.getWidth();  
 this.height = bmp.getHeight();  
 this.value = value;  
  
 Random rnd = new Random();  
  
 switch (rnd.nextInt(2)) {  
 case 0:  
 xSpeed = speed;  
 break;  
  
 case 1:  
 xSpeed = -speed;  
 break;  
 }  
  
 switch (rnd.nextInt(2)) {  
 case 0:  
 ySpeed = speed;  
 break;  
  
 case 1:  
 ySpeed = -speed;  
 break;  
 }  
  
 switch (rnd.nextInt(4)) {  
 case 0:  
 x = -width;  
 y = rnd.nextInt(gameView.getHeight() - height);  
 break;  
  
 case 1:  
 x = rnd.nextInt(gameView.getWidth() - width);  
 y = -height;  
 break;  
  
 case 2:  
 x = gameView.getWidth();  
 y = rnd.nextInt(gameView.getHeight() - height);  
 break;  
  
 case 3:  
 x = rnd.nextInt(gameView.getWidth() - width);  
 y = gameView.getHeight();  
 break;  
 }  
 }  
  
 private void update()  
 {  
 if (x >= gameView.getWidth() - width) {  
 if (xSpeed > 0) {  
 xSpeed = -xSpeed;  
 }  
 }  
 else {  
 if (x <= 0) {  
 if (xSpeed < 0) {  
 xSpeed = -xSpeed;  
 }  
 }  
 }  
  
 x = x + xSpeed;  
  
 if (y >= gameView.getHeight() - height) {  
 if (ySpeed > 0) {  
 ySpeed = -ySpeed;  
 }  
 }  
 else {  
 if (y <= 0) {  
 if (ySpeed < 0) {  
 ySpeed = -ySpeed;  
 }  
 }  
 }  
  
 y = y + ySpeed;  
 }  
  
 boolean isCollision(float x2, float y2) {  
 return x2 > x && x2 < x + width && y2 > y && y2 < y + height;  
 }  
  
 void draw(Canvas canvas)  
 {  
 int srcX = 0;  
 int srcY = 0;  
 Rect src = new Rect(0, 0, srcX + width, srcY + height);  
 Rect dst = new Rect(x, y, x + width, y + height);  
 canvas.drawBitmap(bmp, src, dst, null);  
 }  
  
 boolean isAlive() {  
 return isAlive;  
 }  
  
 void setAlive(boolean statue) {  
 isAlive = statue;  
 }  
  
 int getValue() {  
 return value;  
 }  
  
 @Override  
 public void run() {  
 isAlive = true;  
  
 while (isAlive) {  
 update();  
  
 try {  
 Thread.*sleep*(1000 / 30);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

BugGenerator.java

package com.example.bugs;  
  
import java.util.Random;  
  
public class BugGenerator extends Thread {  
 private boolean isAlive = false;  
  
 private GameView view;  
 private final Random rnd = new Random();  
  
 BugGenerator(GameView view) {  
 this.view = view;  
 }  
  
 @Override  
 public void run() {  
 while (isAlive) {  
 if (view.getCountBugs() < 7) {  
 switch (rnd.nextInt(3)) {  
 case 0:  
 Bug bigbug = new Bug(view, R.drawable.*bigbug*, 0.1, 5, 1);  
 new Thread(bigbug).start();  
 view.bugs.add(bigbug);  
  
 break;  
  
 case 1:  
 Bug fastbug = new Bug(view, R.drawable.*fastbug*, 0.02, 15, 5);  
 new Thread(fastbug).start();  
 view.bugs.add(fastbug);  
  
 break;  
  
 case 2:  
 Bug middlesizebug = new Bug(view, R.drawable.*middlesizebug*, 0.1, 10, 3);  
 new Thread(middlesizebug).start();  
 view.bugs.add(middlesizebug);  
  
 break;  
 }  
 }  
  
 try {  
 Thread.*sleep*(1000 / (8 - view.getCountBugs()));  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
  
 for (Bug bug : view.bugs) {  
 bug.setAlive(false);  
 }  
  
 view.bugs.clear();  
 }  
  
 void setAlive(boolean statue) {  
 isAlive = statue;  
 }  
}

GameManager.java

package com.example.bugs;  
  
import android.graphics.Bitmap;  
import android.graphics.BitmapFactory;  
import android.graphics.Canvas;  
import android.graphics.Color;  
import android.graphics.Paint;  
  
import java.util.List;  
  
public class GameManager extends Thread  
{  
 private static final int *FPS* = 30;  
  
 private GameView view;  
  
 private boolean isAlive = false;  
  
 private Bitmap bmp;  
  
 GameManager(GameView view) {  
 this.view = view;  
  
 bmp = BitmapFactory.*decodeResource*(view.getResources(), R.drawable.*table*);  
 bmp = Bitmap.*createScaledBitmap*(bmp,1080, 2160, true);  
 }  
  
 void setAlive(boolean statue) {  
 isAlive = statue;  
 }  
  
 private void draw(Canvas canvas) {  
 List<Bug> bugs = view.getBugs();  
  
 for(int i = 0; i < bugs.size(); ++i) {  
 if (bugs.get(i) != null && bugs.get(i).isAlive()) {  
 bugs.get(i).draw(canvas);  
 }  
 }  
  
 Paint paint = new Paint();  
 paint.setColor(Color.*WHITE*);  
 paint.setTextSize(80);  
 canvas.drawText(String.*valueOf*(view.getScore()), 100, 100, paint);  
 }  
  
 @Override  
 public void run() {  
 long ticksPS = 1000 / *FPS*;  
 long startTime;  
 long sleepTime;  
  
 while (isAlive) {  
 Canvas c = null;  
 startTime = System.*currentTimeMillis*();  
  
 try {  
 c = view.getHolder().lockCanvas();  
 synchronized (view.getHolder()) {  
 c.drawBitmap(bmp, 0, 0, null);  
  
 draw(c);  
 }  
 } finally {  
 if (c != null) {  
 view.getHolder().unlockCanvasAndPost(c);  
 }  
 }  
  
 sleepTime = ticksPS - (System.*currentTimeMillis*() - startTime);  
  
 try {  
 if (sleepTime > 0) {  
 *sleep*(sleepTime);  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

GameView.java

package com.example.bugs;  
  
import android.content.Context;  
import android.media.AudioAttributes;  
import android.media.SoundPool;  
import android.view.MotionEvent;  
import android.view.SurfaceHolder;  
import android.view.SurfaceView;  
  
import java.util.\*;  
  
public class GameView extends SurfaceView implements  
 SurfaceHolder.Callback {  
  
 private SoundPool soundPool;  
 private volatile int score;  
  
 private GameManager gameLoopThread;  
 private BugGenerator bugGenerator;  
  
  
 public volatile List<Bug> bugs = new ArrayList<>();  
  
 public GameView(Context context) {  
 super(context);  
 gameLoopThread = new GameManager(this);  
 bugGenerator = new BugGenerator(this);  
  
 getHolder().addCallback(this);  
  
 AudioAttributes attributes = new AudioAttributes.Builder()  
 .setUsage(AudioAttributes.*USAGE\_GAME*)  
 .setContentType(AudioAttributes.*CONTENT\_TYPE\_SONIFICATION*)  
 .build();  
  
 soundPool = new SoundPool.Builder().setMaxStreams(7).setAudioAttributes(attributes).build();  
 soundPool.load(context, R.raw.*wood1*, 0);  
 soundPool.load(context, R.raw.*death*, 0);  
  
 score = 0;  
 }  
  
 public List<Bug> getBugs() {  
 return bugs;  
 }  
  
 public int getCountBugs() {  
 return bugs.size();  
 }  
  
 public boolean onTouchEvent(MotionEvent event) {  
 float x = event.getX();  
 float y = event.getY();  
  
 if (event.getAction() == MotionEvent.*ACTION\_DOWN*)  
 synchronized (getHolder())  
 {  
 boolean isCatched = false;  
 for (Bug bug : bugs)  
 {  
 if (bug.isCollision(x, y))  
 {  
 isCatched = true;  
 score += bug.getValue();  
  
 soundPool.play(2, 100, 100, 0, 0, 1);  
  
 bug.setAlive(false);  
 bugs.remove(bug);  
  
 break;  
 }  
 }  
  
 if (!isCatched) {  
 score -=10;  
  
 soundPool.play(1, 100, 100, 0, 0, 1);  
 }  
 }  
  
 return true;  
 }  
  
  
 public void surfaceDestroyed(SurfaceHolder holder) {  
 boolean retry = true;  
 gameLoopThread.setAlive(false);  
 bugGenerator.setAlive(false);  
  
 while (retry)  
 {  
 try  
 {  
 gameLoopThread.join();  
 bugGenerator.join();  
 retry = false;  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
  
 soundPool.release();  
 soundPool = null;  
 }  
  
  
 public void surfaceCreated(SurfaceHolder holder) {  
 gameLoopThread.setAlive(true);  
 gameLoopThread.start();  
  
 bugGenerator.setAlive(true);  
 bugGenerator.start();  
 }  
  
 public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {  
 }  
  
 public int getScore() {  
 return score;  
 }  
}

FullscreenActivity.java

package com.example.bugs;  
  
import android.annotation.SuppressLint;  
import android.content.pm.ActivityInfo;  
import android.support.v7.app.ActionBar;  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
import android.os.Handler;  
import android.view.MotionEvent;  
import android.view.View;  
  
*/\*\*  
 \* An example full-screen activity that shows and hides the system UI (i.e.  
 \* status bar and navigation/system bar) with user interaction.  
 \*/*public class FullscreenActivity extends AppCompatActivity {  
 */\*\*  
 \* Whether or not the system UI should be auto-hidden after  
 \* {****@link*** *#AUTO\_HIDE\_DELAY\_MILLIS} milliseconds.  
 \*/* private static final boolean *AUTO\_HIDE* = true;  
  
 */\*\*  
 \* If {****@link*** *#AUTO\_HIDE} is set, the number of milliseconds to wait after  
 \* user interaction before hiding the system UI.  
 \*/* private static final int *AUTO\_HIDE\_DELAY\_MILLIS* = 3000;  
  
 */\*\*  
 \* Some older devices needs a small delay between UI widget updates  
 \* and a change of the status and navigation bar.  
 \*/* private static final int *UI\_ANIMATION\_DELAY* = 300;  
  
 private View mContentView;  
 private View mControlsView;  
 private boolean mVisible;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
  
 setContentView(R.layout.*activity\_fullscreen*);  
  
 mVisible = true;  
 mControlsView = findViewById(R.id.*fullscreen\_content\_controls*);  
 mContentView = findViewById(R.id.*fullscreen\_content*);  
  
  
 // Set up the user interaction to manually show or hide the system UI.  
 mContentView.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 toggle();  
 }  
 });  
  
  
 setRequestedOrientation(ActivityInfo.*SCREEN\_ORIENTATION\_PORTRAIT*);  
  
 setContentView(new GameView(this));  
 }  
  
 @Override  
 protected void onPostCreate(Bundle savedInstanceState) {  
 super.onPostCreate(savedInstanceState);  
  
 delayedHide(100);  
 }  
  
 */\*\*  
 \* Touch listener to use for in-layout UI controls to delay hiding the  
 \* system UI. This is to prevent the jarring behavior of controls going away  
 \* while interacting with activity UI.  
 \*/* private final View.OnTouchListener mDelayHideTouchListener = new View.OnTouchListener() {  
 @Override  
 public boolean onTouch(View view, MotionEvent motionEvent) {  
 if (*AUTO\_HIDE*) {  
 delayedHide(*AUTO\_HIDE\_DELAY\_MILLIS*);  
 }  
  
 return false;  
 }  
 };  
  
 private void toggle() {  
 if (mVisible) {  
 hide();  
 } else {  
 show();  
 }  
 }  
  
 private void hide() {  
 // Hide UI first  
 ActionBar actionBar = getSupportActionBar();  
 if (actionBar != null) {  
 actionBar.hide();  
 }  
 mControlsView.setVisibility(View.*GONE*);  
 mVisible = false;  
  
 // Schedule a runnable to remove the status and navigation bar after a delay  
 mHideHandler.removeCallbacks(mShowPart2Runnable);  
 mHideHandler.postDelayed(mHidePart2Runnable, *UI\_ANIMATION\_DELAY*);  
 }  
  
 private final Runnable mHidePart2Runnable = new Runnable() {  
 @SuppressLint("InlinedApi")  
 @Override  
 public void run() {  
 // Delayed removal of status and navigation bar  
  
 // Note that some of these constants are new as of API 16 (Jelly Bean)  
 // and API 19 (KitKat). It is safe to use them, as they are inlined  
 // at compile-time and do nothing on earlier devices.  
 mContentView.setSystemUiVisibility(View.*SYSTEM\_UI\_FLAG\_LOW\_PROFILE* | View.*SYSTEM\_UI\_FLAG\_FULLSCREEN* | View.*SYSTEM\_UI\_FLAG\_LAYOUT\_STABLE* | View.*SYSTEM\_UI\_FLAG\_IMMERSIVE\_STICKY* | View.*SYSTEM\_UI\_FLAG\_LAYOUT\_HIDE\_NAVIGATION* | View.*SYSTEM\_UI\_FLAG\_HIDE\_NAVIGATION*);  
 }  
 };  
  
 @SuppressLint("InlinedApi")  
 private void show() {  
 // Show the system bar  
 mContentView.setSystemUiVisibility(View.*SYSTEM\_UI\_FLAG\_LAYOUT\_FULLSCREEN* | View.*SYSTEM\_UI\_FLAG\_LAYOUT\_HIDE\_NAVIGATION*);  
 mVisible = true;  
  
 // Schedule a runnable to display UI elements after a delay  
 mHideHandler.removeCallbacks(mHidePart2Runnable);  
 mHideHandler.postDelayed(mShowPart2Runnable, *UI\_ANIMATION\_DELAY*);  
 }  
  
 private final Runnable mShowPart2Runnable = new Runnable() {  
 @Override  
 public void run() {  
 // Delayed display of UI elements  
 ActionBar actionBar = getSupportActionBar();  
 if (actionBar != null) {  
 actionBar.show();  
 }  
 mControlsView.setVisibility(View.*VISIBLE*);  
 }  
 };  
  
 private final Handler mHideHandler = new Handler();  
 private final Runnable mHideRunnable = new Runnable() {  
 @Override  
 public void run() {  
 hide();  
 }  
 };  
  
 */\*\*  
 \* Schedules a call to hide() in delay milliseconds, canceling any  
 \* previously scheduled calls.  
 \*/* private void delayedHide(int delayMillis) {  
 mHideHandler.removeCallbacks(mHideRunnable);  
 mHideHandler.postDelayed(mHideRunnable, delayMillis);  
 }  
}

# Скрины

# 

