Модуль 3. Основы программирования Android приложений

Тема 3.7. Двумерная графика в Android приложениях

2 часа

Оглавление

3.7. Двумерная графика в Android приложениях

3.7.1. Основы графики в Android

Изучение техники рисования под Android начнем с создания нового проекта. Создадим проект как обычно и назовем его **AndroidPaint**.

Работая в среде Android с различными программами, пользователь взаимодействует с ними с помощь кнопок, выпадающих списков и других элементов интерфейса. Эти элементы интерфейса являются объектами, созданными на основе разновидностей класса **View**. Для проведения графических экспериментов в нашем проекте мы создадим новый элемент интерфейса, который будет наследоваться от класса **View**.

Создадим новый класс **MyDraw** и унаследуем его от **View.**

Для того, чтобы можно было рисовать на поверхности нашего элемента, переопределим метод **onDraw** в созданном классе. Вызывать его вручную не нужно. Этот метод относится к методам обратного вызова, его будет вызывать операционная система когда будет нужно. Метод **onDraw** вызывается операционной системой при старте активности чтобы нарисовать элемент интерейса на экране. Нам нужно описать только действия, которые должен выполнять **onDraw –** что и в какой последовательности рисовать на поверхности элемента.

Код получившейся заготовки класса **MyDraw** должен выглядеть так:

```
package ru.samsung.itschool.mydraw;

public class MyDraw extends View{
	public MyDraw (Context context) {
		super(context);
	}

	@Override
	protected void onDraw(Canvas canvas){
		super.onDraw(canvas);
	}
}
```

Здесь мы унаследовали класс **MyDraw** от класса **android.view.View** и переопределили метод **onDraw(Canvas canvas)**. Все строчки, которые будут выделяться волнистой линией нужно проверить, а также импортировать необходимые пакеты самостоятельно, если необходимо. Импорт удобно проводить с помощью сочетания клавиш **Ctrl + Shift + O**.

Далее мы создадим объект - экземпляр класса разработанного элемента **MyDraw** и разместить его на активности. Откроем файл с описанием класса активности и немного изменим код внутри метода **onCreate**:

```
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
```

```
            requestWindowFeature(Window.FEATURE_NO_TITLE);
            // setContentView(R.layout.activity_main);
    }
```

Методы обратного вызова часто используются в програмировании под Android. Например, чтобы разместить наш виджет на Активности (экране приложения) в методе обратного вызова onCreate мы разместили код:

setContentView(new MyDraw(this));

Сам метод onCreate вызывать нигде не нужно. Система его вызовет сама при создании активности.

Обратите внимание, что строчка с установкой макета **activity_main** закомментирована. Для того чтобы избавиться от рамки с заголовком в активности мы также добавили вызов метода **requestWindowFeature(Window.FEATURE_NO_TITLE);** Но это делать вовсе не обязательно.

На этом подготовительные работы завершены. Дальнейший код мы будем писать в созданном классе **MyDraw**.

Взгляните на класс **MyDraw**. При вызове в метод **onDraw** передается объект класса **Canvas** – это холст, на котором мы будет рисовать. На объекте **Canvas** введена декартова система координат, левый верхний угол соответствует точке (0;0), ось Y направлена вниз, ось X – вправо. Размер области рисования можно узнать вызовом методов **getWidth()** и **getHeight().** С помощью различных методов класса мы можем рисовать линии, окружности, дуги и так далее.

Прежде чем что-то рисовать, нужно определить некоторые параметры рисования такие как цвет, толщина и способ рисования (рисование контуров или заливка фигуры). Сам процесс рисования во многом аналогичен рисованию на бумаге – здесь в нашем распоряжении есть программные аналоги цветных карандашей, кисти, линейки, циркуля и т.п. Например, мы можем взять толстый зеленый карандаш и нарисовать им линию, затем взять циркуль с тонким красным карандашом и нарисовать окружность. Эти действия в правильном порядке необходимо прописать в коде при рисовании.

Параметры рисования определяются в объекте класса **Paint**. Именно он содержит стили, цвета и другую графическую информацию для рисования графических объектов. С его помощью можно выбирать способ отображения графических примитивов, которые можно рисовать на объекте **Canvas**. Например, чтобы установить сплошной цвет для рисования линии, нужно вызвать метод **setColor()**. Но для этого нужно сначала создать объект класса **Paint**.

```
Paint paint = new Paint();
paint.setColor(Color.RED); // Выбираем инструмент красного цвета
```

Объект Paint может вести себя как карандаш, рисуя контуры фигуры, или как кисть в графических редакторах – закрашивая фигуры цветом. За это поведение отвечает метод **setStyle ().**

Очертания графического примитива можно рисовать используя вызов **setStyle()** с параметром:

Paint.Style.FILL	рисование с заливкой
Paint.Style.STROKE	рисование только контура
Paint.Style.FILL_AND_STROKE	рисование и контура и заливки

Для рисования качественной графики можно вызывать методы, обеспечивающие сглаживание пикселей:

```
Paint paint = new Paint();
paint.setSubpixelText(true); // Субпиксельное сглаживание для текста
paint.setAntiAlias(true);      // Антиальясинг, сглаживание  линий
```

Для представления цвета в Android обычно используют 32-битное целое число, а не экземпляры класса **Color**. Для задания цвета можно использовать статические константы класса **Color**, например:

```
int blue = Color.BLUE;
Paint paint = new Paint();
paint.setColor(blue);
```

Такие константы описаны только для самых основных цветов. Цвет также можно описать с помощью четырех восьмибитных чисел в формате **ARGB**, по одному для каждого канала (Alpha, Red, Green, Blue). Напомним, что каждое восьмибитное целое беззнаковое число может принимать значения от 0 до 255. Получить нужный цвет из набора цветовых компонентов можно через методы rgb() или argb(), которые возвращают целые значения (код цвета).

```
// Полупрозрачный синий
int myTransparentBlue = Color.argb(127, 0, 0, 255);
```

Метод parseColor() позволяет получить целочисленное значение из шестнадцатеричной формы:

```
int pColor = Color.parseColor("#FF0000FF");
```

Для большей гибкости приложения, цвета помещают в ресурсы. В этом случае их можно будет менять без вмешательства в код программы:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

      <color name="ballColor" >#FF00FFFF
      </color>
```

```
</resources>
```

В коде можно обратиться к цвету следующим образом:

```
int ballColor = getResources().getColor(R.color.ballColor);
```

Имя для цвета в файле ресурсов желательно давать описательное, соответствующее тому, для чего этот цвет применяется. Например, **backgroundColor**. Имена с названиями цветов вроде **red** или **blue** использовать не рекомендуется, в этом случае при изменении кода цвета можно потерять смысл в его имени.

Теперь можно приступить к экспериментам в проекте.

Как было сказано выше, вся работа с графикой происходит в методе onDraw() класса. Для начала установим цвет холста на котором будем рисовать, пусть это будет черный цвет.

Для этого сразу после строчки super.onDraw(canvas) напишем код:

```
Paint paint = new Paint();
// Выбираем кисть
paint.setStyle(Paint.Style.FILL);
// Черный цвет кисти
paint.setColor(Color.BLACK);
// Закрашиваем холст
canvas.drawPaint(paint);
```

Далее мы будем рисовать на этом «подготовленном» холсте. Следуя описанному выше принципу, перед каждым рисованием будем указывать настройки инструмента.

Например, для того, чтобы нарисовать сглаженный красный круг в центре экрана напишем:

```
// Включаем антиальясинг
paint.setAntiAlias(true);
paint.setColor(Color.RED);
// Круг радиусом 100 пикселей в центре экрана
canvas.drawCircle(getWidth() / 2, getHeight() / 2, 100, paint);
```

Для рисования прямоугольника аналогично нужно указать координаты и инструмент:

```
// Синий прямоугольник вверху экрана
paint.setColor(Color.BLUE);
canvas.drawRect(0, 0, getWidth(),200, paint);
```

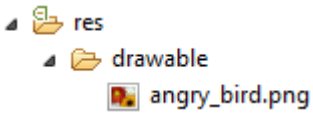
Выведем текст на экран:

```
paint.setColor(Color.WHITE);
paint.setStyle(Paint.Style.FILL);
paint.setTextSize(30);
canvas.drawText("Samsung IT school", 50, 100, paint);
```

Выведем текст под углом:

```
// Расположить текст в этих координатах
int x = 310;
int y = 190;
paint.setColor(Color.YELLOW);
paint.setTextSize(20);
String rotatedText = "Samsung IT school";

// Ограничивающий прямоугольник для наклонного текста
Rect rectangle = new Rect();
paint.getTextBounds(rotatedText, 0, rotatedText.length(), rectangle);

// Вращаем холст по центру текста
canvas.rotate(-45, x + rectangle.exactCenterX(),
      y + rectangle.exactCenterY());

// Выводим текст
paint.setStyle(Paint.Style.FILL);
canvas.drawText(rotatedText, x, y, paint);

// Восстанавливаем нормальное положение холста
canvas.restore();
```

На экране можно рисовать готовые изображения из файлов. Хоть Android и поддерживает распространенные форматы изображений, рекомендуемым является формат PNG. Поместим изображение с именем **angry_bird** в папку **res\drawable** как графический ресурс приложения.




```
int xx = 200, yy = 200;
// Выведем изображение angry_bird на экран под углом 90 градусов
Bitmap image = BitmapFactory.decodeResource(getResources(),
            R.drawable.angry_bird);
canvas.drawBitmap(image, xx, yy, paint);

// вращаем холст относительно центра фигуры
canvas.rotate(-90, xx + image.getWidth() / 2f, yy + image.getHeight()
            / 2f);

// восстанавливаем холст
canvas.restore();
```

3.7.2. Игра “Забавные птички”

Смысл игры

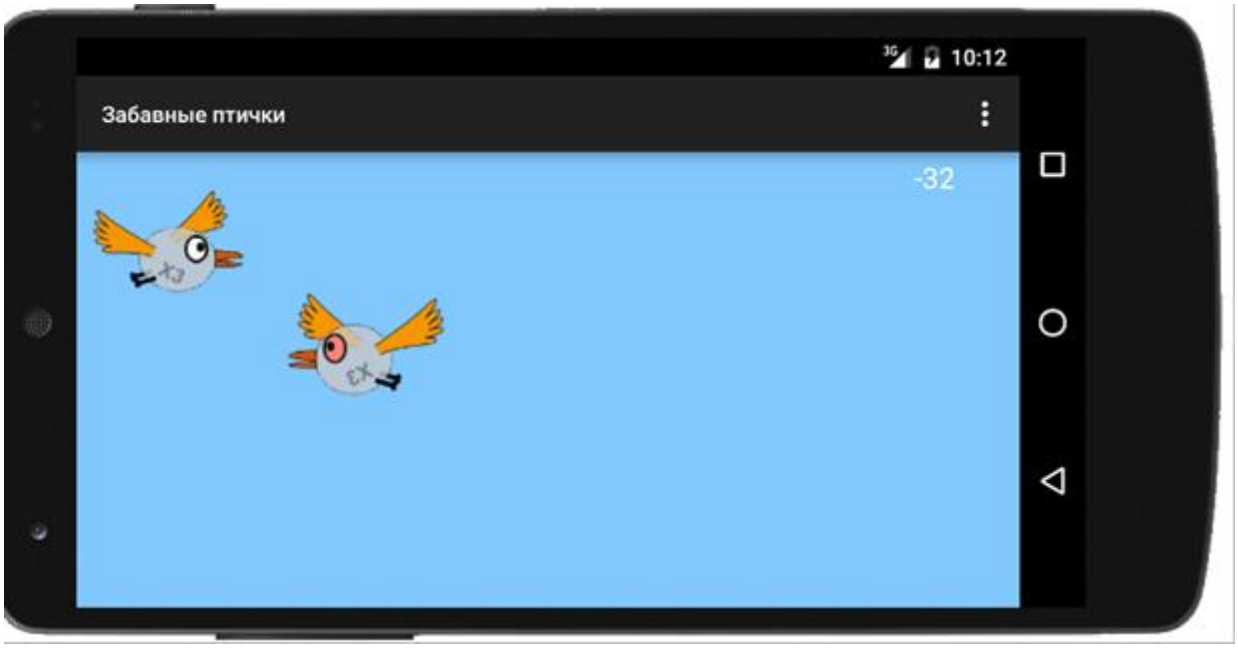
Вы управляете птицей, спешащей домой к себе в гнездо. Ваша задача уворачиваться от пролетающих птиц и набирать очки.

Игровой процесс

Птица игрока может летать вверх или вниз относительно экрана. При соприкосновении со стенками птица отталкивается от них и летит в противоположную сторону. Игрок может управлять птицей, осуществляя прикосновения выше и ниже положения птицы на экране.

За облет птицы противника назначаются очки.

Соприкосновение с птицей противника, а также смена направления полета приводит к уменьшению очков.



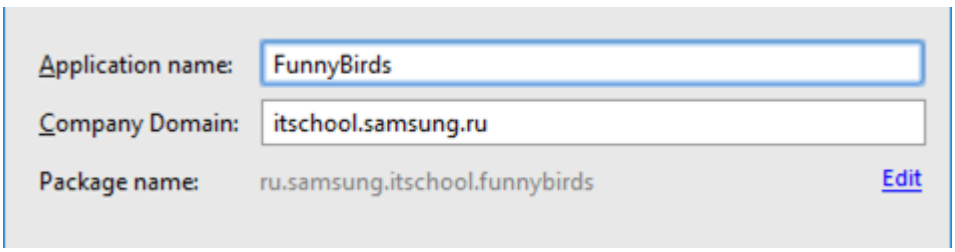
(В игре использованы графические ресурсы со страницы сайта http://www.xojo3d.com/tut015.php)

Создание проекта

1. Создайте новый проект в **Android Studio File | New |New Project…**

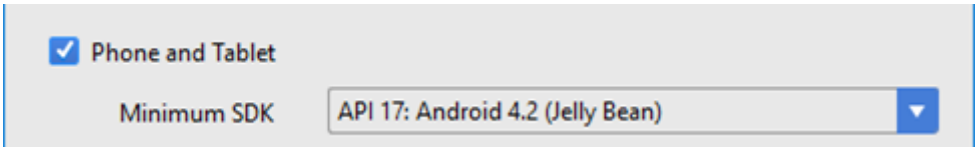
2. Укажите название проекта – **FunnyBirds** и доменное имя **itschool.samsung.ru.**

«Перевернутое» доменное имя **ru.samsung.itschool.funnybirds** будет использоваться для создания пакета, в котором будут находиться классы нашего игрового приложения приложения.



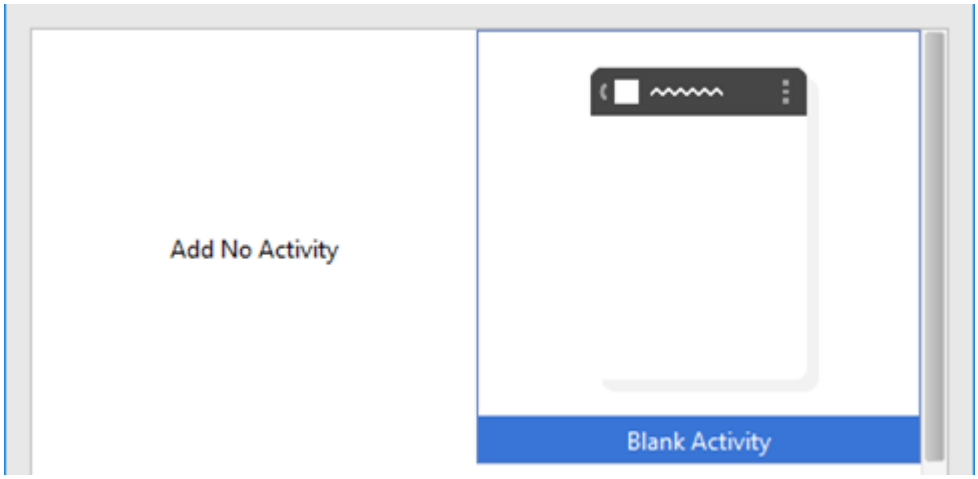
Нажмите **Next**.

3. В следующем окне укажите тип устройств – Phone and Tablet, для которых будем разрабатывать приложение, а также минимальную версию системы, под которой будет работать программа.

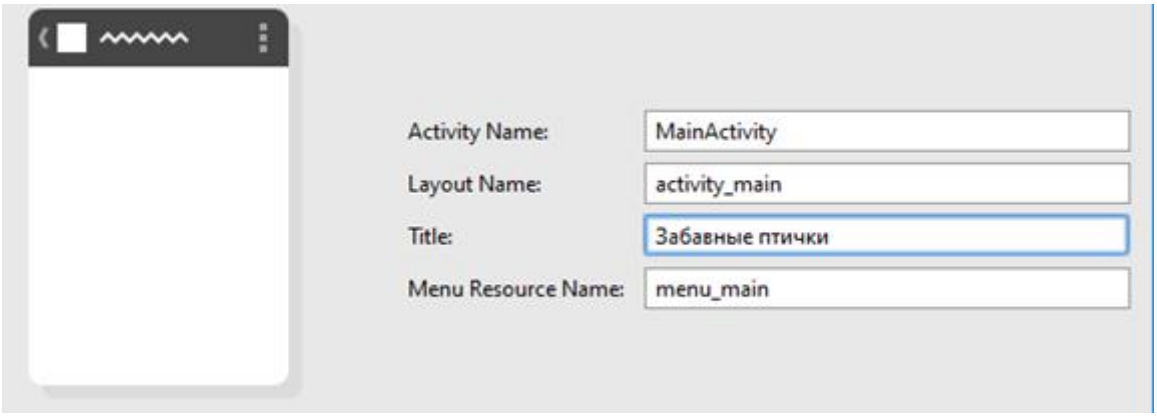


Нажмите **Next**.

4. В окне выбора шаблона активности укажем Blank Activity.



5. В окне настройки активности укажем заголовок активности Забавные птички. Остальные поля можем оставить без изменений.



Нажмите **Finish**.

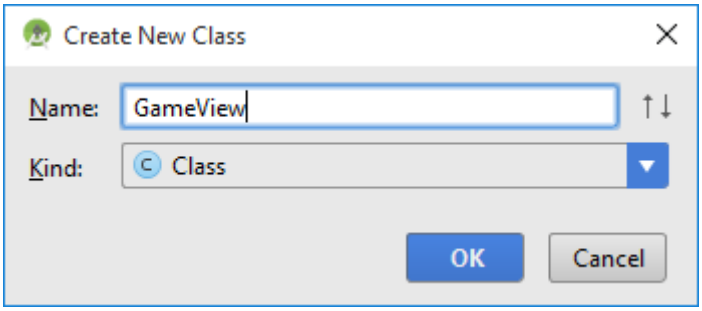
Создание класса GameView

Для управления игрой – реализации игровой логики, отображение игры на экране и взаимодействия с пользователем разработаем класс **GameView**. Объект этого класса будет служить игровым полем для игры **Забавные птички**.

За основу класса **GameView** возьмем стандартный класс **View**, который является базовым классом компонентов интерфейса пользователя в Android.

1. Создание класса GameView

В пакете **ru.samsung.itschool** создайте новый класс с именем **GameView – File |New | Java Class**



2. В качестве суперкласса для созданного GameView укажите View.

Добавьте конструктор, который принимает единственный параметр – контекст.

```
public class GameView extends View{

    public GameView(Context context) {

          super(context);

    }

}
```

3. Переопределите метод onSizeChanged для определения размеров игрового поля:

```
protected void onSizeChanged(int w, int h, int oldw, int oldh) {

      super.onSizeChanged(w, h, oldw, oldh);

      viewWidth = w;
      viewHeight = h;

}
```

Переменные **viewWidth** и **viewHeight** объявите как поля класса **GameView** – они будут содержать актуальные размеры игрового поля:

```
private                              int                              viewWidth;
private int viewHeight;
```

4. Создайте в **GameView** поле **points** для хранения очков, набранных игроком.

```
private int points = 0;
```

5. Чтобы можно было рисовать на поверхности компонента, переопределим метод onDraw() в созданном классе.

Метод onDraw() является методом обратного вызова, его будет вызывать операционная система тогда, когда будет нужно перерисовать компонент (например, при старте активности). Вручную onDraw() вызывать не нужно.

Внутрь метода onDraw() передается объект типа Canvas – холст, на котором можно рисовать, вызывая методы этого объекта. Важно будет определить правильную последовательность рисования.

```
@Override

protected void onDraw(Canvas canvas) {

super.onDraw(canvas);

}
```

6. Отрисуем фон и количество очков на поверхности компонента

```
@Override
protected void onDraw(Canvas canvas) {

    super.onDraw(canvas);

    canvas.drawARGB(250, 127, 199, 255); // заливаем цветом
     Paint p = new Paint();
     p.setAntiAlias(true);
     p.setTextSize(55.0f);
     p.setColor(Color.WHITE);
     canvas.drawText(points+"", viewWidth - 100, 70, p);

}
```

Добавление компонента GameView в активность

Активность не будет содержать разметку, полученную из xml-файла. Единственным элементом, который будет содержаться в активности будет экземпляр класса **GameView** (наше игровое поле).

1. Откройте файл MainActivity с описанием класса активности и немного измените код, внутри метода **onCreate**.

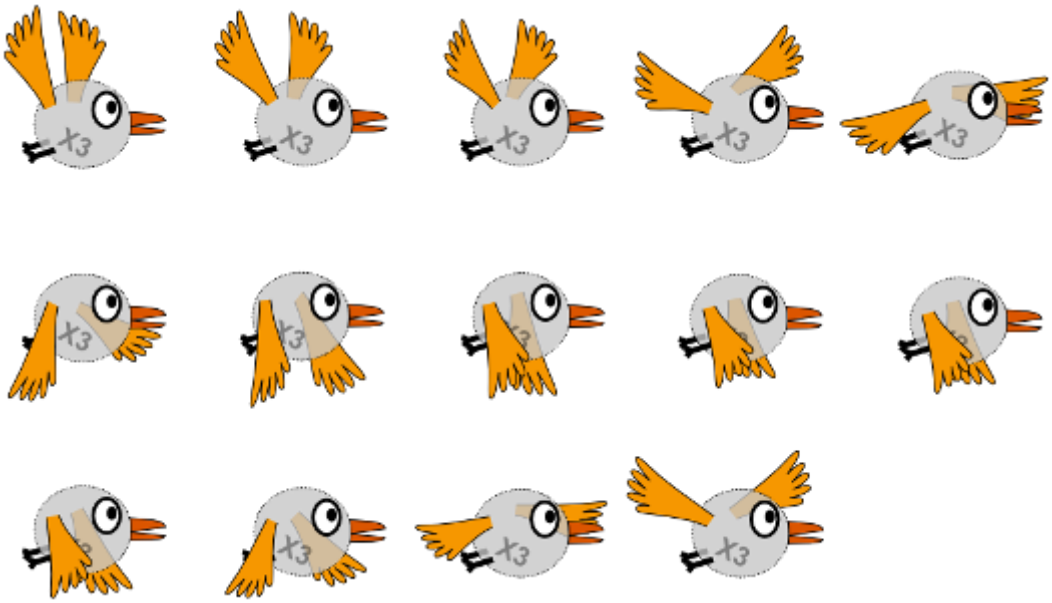
Вместо существующей разметки добавим на активность разработанный компонент **GameView**.

```
protected void onCreate(Bundle savedInstanceState) {

      super.onCreate(savedInstanceState);

      //setContentView(R.layout.activity_main);
      setContentView(new GameView(this));
```

```
}
```

Создание класса Sprite для управления анимацией

Класс **Sprite** будет являться базовым классом для всех игровых объектов. Он будет поддерживать перемещение объектов на экране, покадровую анимацию и определять столкновения объектов. Основой класса является картинка, содержащая множество кадров с различным состоянием игрового объекта (персонажа). Ниже показана картинка, которая будет использоваться для создания анимации главного героя – птички, спешащей домой.



1. Создание нового класса **Sprite**

В пакете **ru.samsung.itschool.funnybirds** создайте новый класс с именем **Sprite – File| New | Java Class**

2. Добавьте в класс следующие поля:

```
    private Bitmap bitmap;
    private List<Rect> frames;
    private int frameWidth;
    private int frameHeight;
    private int currentFrame;
    private double frameTime;
    private double timeForCurrentFrame;

    private double x;
    private double y;

    private double velocityX;
    private double velocityY;

    private int padding;
```

Импортируйте используемые классы **Bitmap** и **Rect – (Alt + Enter).**

Переменная	Назначение
bitmap	Ссылка на изображение с набором кадров.
frames	Коллекция прямоугольных областей на изображении – кадры, которые участвуют в анимационной последовательности.
currentFrame	Номер текущего кадра в коллекции **frames**.
frameTime	Время, отведенное на отображение каждого кадра анимационной последовательности.
timeForCurrentFrame	Текущее время отображения кадра. Номер текущего кадра **currentFrame** меняется на следующий при достижении переменной **timeForCurrentFrame** значения из **frameTime**.
frameWidth и **frameHeight**	Ширина и высоту кадра для отображения на экране.
x и **y**	Местоположение левого верхнего угла спрайта на экране.
velocityX и **velocityY**	Скорости перемещения спрайта по оси X и Y соответственно.
Padding	Внутренний отступ от границ спрайта, необходимый для более точного определения пересечений спрайтов.

3. Создайте конструктор класса Sprite

```
public Sprite(double x, double y, double velocityX, double velocityY,
            Rect initialFrame, Bitmap bitmap) {

      this.x = x;
      this.y = y;
      this.velocityX = velocityX;
      this.velocityY = velocityY;
      this.bitmap = bitmap;
      this.frames = new ArrayList<Rect>();
      this.frames.add(initialFrame);
      this.bitmap = bitmap;
      this.timeForCurrentFrame = 0.0;
      this.frameTime = 0.1;
      this.currentFrame = 0;
      this.frameWidth = initialFrame.width();
      this.frameHeight = initialFrame.height();
      this.padding = 20;
}
```

4. Создайте геттеры и сеттеры для полей класса.

Поскольку поля класса должны быть скрыты от доступа из вне, для доступа к ним создадим методы – геттеры (для получения значений) и сеттеры (для установки значений).

Большинство методов можно сгенерировать автоматически **Code |Generate… |Getter and Setter.**

```
public double getX() {
	return x;
}

public void setX(double x) {
	this.x = x;
}

public double getY() {
	return y;
}

public void setY(double y) {
	this.y = y;
}

public int getFrameWidth() {
	return frameWidth;
}

public void setFrameWidth(int frameWidth) {
	this.frameWidth = frameWidth;
}

public int getFrameHeight() {
	return frameHeight;
}

public void setFrameHeight(int frameHeight) {
	this.frameHeight = frameHeight;
}


public double getVx() {
	return velocityX;
}

public void setVx(double velocityX) {
	this.velocityX = velocityX;
}

public double getVy() {
	return velocityY;
}

public void setVy(double velocityY) {
	this.velocityY = velocityY;
}

public int getCurrentFrame() {
	return currentFrame;
}
```

```
public void setCurrentFrame(int currentFrame) {
       this.currentFrame = currentFrame%frames.size();
}

public double getFrameTime() {
       return frameTime;
}

public void setFrameTime(double frameTime) {
       this.frameTime = Math.abs(frameTime);
}

public double getTimeForCurrentFrame() {
       return timeForCurrentFrame;
}

public void setTimeForCurrentFrame(double timeForCurrentFrame) {
       this.timeForCurrentFrame = Math.abs(timeForCurrentFrame);
}

public int getFramesCount () {
       return frames.size();
}
```

5. Создайте метод добавления кадров в анимационную последовательность.

```
       public void addFrame (Rect frame) {
              frames.add(frame);
       }
```

Добавление кадров в последовательность анимации заключается в добавлении соответствующей кадру прямоугольной области на изображении, заданной с помощью объекта Rect (прямоугольник).

6. Добавьте метод update () для обновления внутреннего состояния спрайта.

```
public void update (int ms) {

         timeForCurrentFrame += ms;

         if (timeForCurrentFrame >= frameTime) {
              currentFrame = (currentFrame + 1) % frames.size();
              timeForCurrentFrame = timeForCurrentFrame - frameTime;
         }

         x = x + velocityX * ms/1000.0;
         y = y + velocityY * ms/1000.0;
}
```

Метод **update()** вызываться таймером с определенной периодичностью. Внутрь метода передается время в миллисекундах, прошедшее с момента последнего вызова этого метода.

Время используется для изменения состояния спрайта – его перемещения в пространстве за это время, а также рассчитывается необходимость перехода к следующему кадру.

В **update()** реализуется механика перебора кадров изображения на основании текущего времени воспроизведения кадра и его сравнения с необходимым временем воспроизведения одного кадра. В зависимости от времени и скорости по осям X и Y осуществляется изменение координат спрайта на экране.

Переход к следующему кадру и зацикливание (переход от последнего к нулевому) осуществляется здесь:

```
currentFrame = (currentFrame + 1) % frames.size();
```

7. Создайте метод draw () для отрисовки спрайта на холсте.

```
public void draw (Canvas canvas) {

        Paint p = new Paint();

        Rect destination = new Rect((int)x, (int)y, (int)(x + frameWidth),
(int)(y + frameHeight));

        canvas.drawBitmap(bitmap, frames.get(currentFrame), destination,  p);

}
```

Метод **draw**() отрисовывает на переданном холсте текущий кадр **frames.get(currentFrame)** в области экрана, заданной в прямоугольнике **destination**.

8. Добавьте метод определения пересечения спрайтов.

Когда два объекта нашей игры взаимодействуют друг с другом нам нужно знать сталиваются ли они, и в зависимости от этого предпринимать определенные действия.

Поскольку спрайты могут содержать небольшую пустую область вокруг себя, для определения столкновений лучше использовать прямоугольные области, отсыпающие внутрь от границ спрайта на величину **padding**.

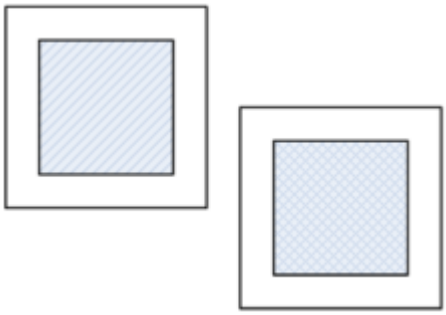
Метод, возвращающий прямоугольную область, участвующую в определении столкновений:

```
public Rect getBoundingBoxRect () {

        return new Rect((int)x+padding,

                (int)y+padding,

                (int)(x + frameWidth - 2 *padding),

                (int)(y + frameHeight - 2* padding));

}
```

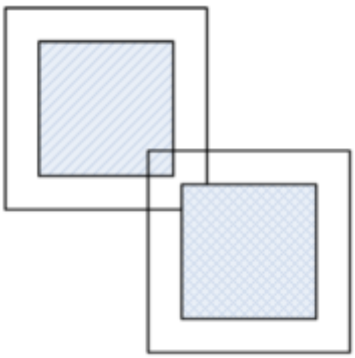
Метод определения пересечения спрайтов:

```
public boolean intersect (Sprite s) {
        return getBoundingBoxRect().intersect(s.getBoundingBoxRect());
}
```

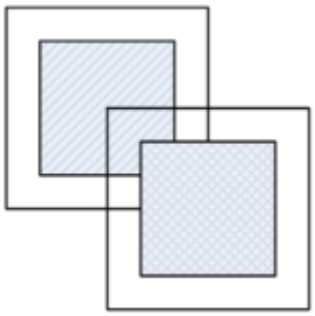
Нет наложения спрайтов, нет пересечения:



Есть наложения спрайтов, но нет пересечения:



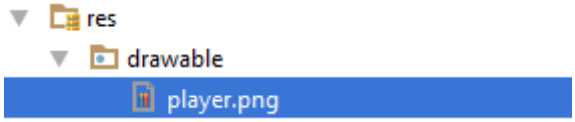
Есть наложение спрайтов, есть пересечение:



Создание Птицы, управляемой пользователем

1. Добавьте в папку **res/drawable** изображение птицы, состоящей из нескольких кадров **player.png**

Изображение можно добавить, используя буфер обмена – скопировать файл в проводнике и вставить в конечную папку**.**



2. Объявите переменную playerBird типа Sprite как поле класса GameView.

```
    private Sprite playerBird;
```

3. Создайте объект класса Sprite в конструкторе GameView.

Для создания объекта, конструктору класса Sprite нужно передать исходные координаты объекта на экране (10, 0), проекции скорости на ось X и Y – 0 и 200 соответственно, а также изображение с кадрами и прямоугольник, описывающий границы первого кадра.

Содержимое конструктора GameView:

```
public GameView(Context context) {

    super(context);
    Bitmap b = BitmapFactory.decodeResource(getResources(), R.drawable.player);

    int w = b.getWidth()/5;
    int h = b.getHeight()/3;

    Rect firstFrame = new Rect(0, 0, w, h);

    playerBird = new Sprite(10, 0, 0, 100, firstFrame, b);
}
```

4. Отрисуйте птицу на экране.

Для отрисовки спрайта птицы на компоненте GameView, вызовите у спрайта метод draw:

```
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    canvas.drawARGB(250, 127, 199, 255); // цвет фона
    playerBird.draw(canvas);
}
```

Добавление анимации у Птицы

1. Обновление игровой модели

В классе **GameView** определим метод **update()** в котором будет происходить изменение игрового состояния (модели игры). Например, в этом методе будут происходить обновления всех спрайтов в игре – вызов их метода **update().**

Вызов данного метода удобно осуществлять автоматически через определенные промежутки времени с помощью таймера.

Определим промежуток времени, за который должно происходить изменение игровой модели. Добавьте константу **timerInterval** как поле класса **GameView.**

```
        private final int timerInterval = 30;
```

В методе update() обновим состояние спрайта с птицей и перерисуем GameView:

```
  protected void update () {

        playerBird.update(timerInterval);
         invalidate();

}
```

2. Добавление класса таймера в класс GameView

Таймер под Android в Java можно реализовать при помощи класса-наследника **CountDownTimer**.

В реализуемом классе необходимо переопределить абстрактные методы **onTick()** и **onFinish().** В методе **onTick()** указываются действия, которые нужно делать периодически, например, вызов **update()** у спрайта и перерисовки **GameView** – **invalidate().** В методе **onFinish()** – действия, когда таймер заканчивает свою работу.

Определите класс **Timer** непосредственно внутри **GameView** для удобного вызова методов класса **GameView**. В конструкторе класса укажите общее время работы таймера – **Integer.MAX_VALUE** и время периодического срабатывания - **timerInterval**.

При срабатывании таймера **onTick()** вызовите метод **update()** класса **GameView.**

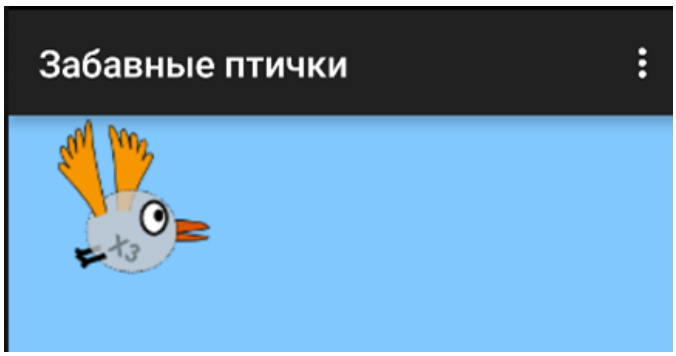
```
class Timer extends CountDownTimer {

        public Timer() {

        super(Integer.MAX_VALUE, timerInterval);

        }

        @Override
        public void onTick(long millisUntilFinished) {
                update ();

        }

        @Override
        public void onFinish() {

        }

}
```

3. Создайте и запустите таймер

В самом конце конструктора GameView после создания всех спрайтов создайте и запустите таймер.

```
Timer t = new Timer();
t.start();
```

Фрагмент экрана с игрой после запуска: (птица падает вниз)



4. Добавьте больше кадров с изображением птицы

После создания спрайта птицы в конструкторе GameView добавьте следующий фрагмент:

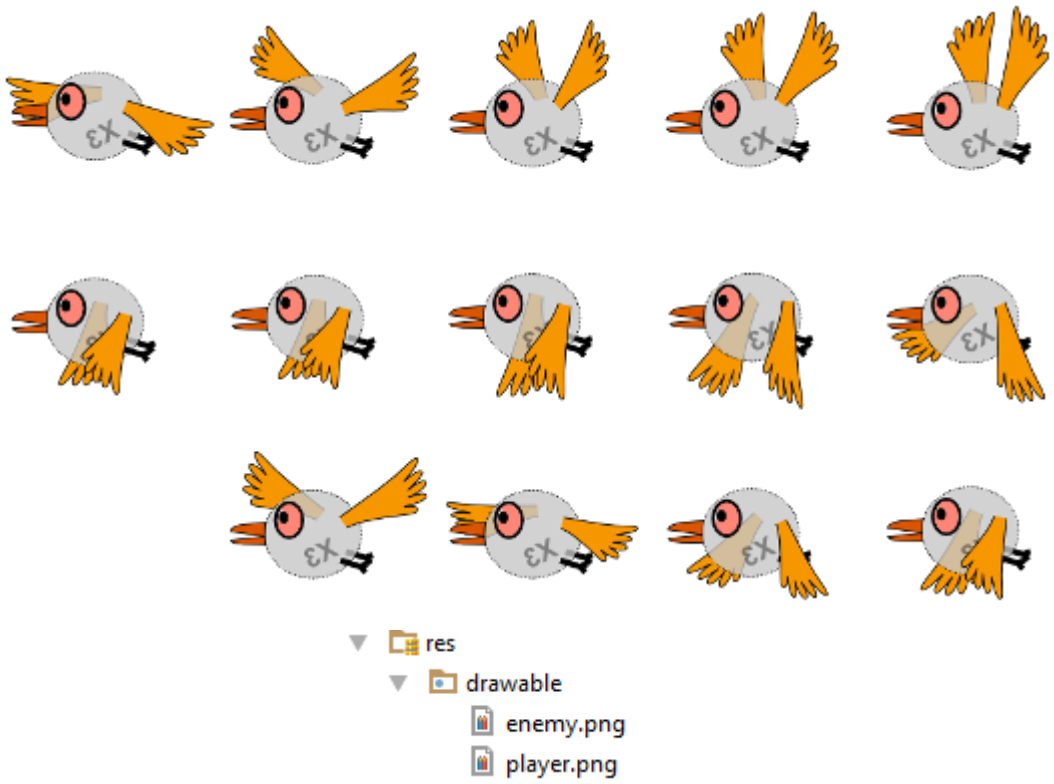
```
for (int i = 0; i < 3; i++) {

	for (int j = 0; j < 4; j++) {
		if (i == 0 && j == 0) {
			continue;
		}
		if (i == 2 && j == 3) {
			continue;
		}
		playerBird.addFrame(new Rect(j * w, i * h, j * w + w, i * w + w));
	}
}
```

Теперь птица сможет махать крыльями.

Добавление противника

Во многом процесс добавления противника – птицы, летящей на встречу игроку, аналогичен уже рассмотренному.

1. Добавьте в папку **res/drawable** изображение летящей на встречу птицы **enemy.png**:



2. Создайте поле **enemyBird** в классе **GameView**.

```
    private Sprite enemyBird;
```

3. Пропишите в конструкторе **GameView** создание спрайта и добавление в него кадров:

```
. . .
b = BitmapFactory.decodeResource(getResources(), R.drawable.enemy);

w = b.getWidth()/5;

h = b.getHeight()/3;

firstFrame = new Rect(4*w, 0, 5*w, h);

enemyBird = new Sprite(2000, 250, -300, 0, firstFrame, b);


for (int i = 0; i < 3; i++) {

      for (int j = 4; j >= 0; j--) {

          if (i ==0 && j == 4) {
             continue;
          }
          if (i ==2 && j == 0) {
             continue;
```

```
            }
            enemyBird.addFrame(new Rect(j*w, i*h, j*w+w, i*w+w));
        }
}
. . .
```

4. Отрисуйте спрайт в методе **onDraw()** класса **GameView:**

```
enemyBird.draw(canvas);
```

5. Измените состояние спрайта в методе **onUpdate()** класса **GameView:**

```
enemyBird.update(timerInterval);
```

Управление птицей и контроль столкновений

Пользователь может направлять движение птицы. Прикосновением пальца выше спрайта птицы – подняться выше, прикосновением ниже спрайта – опуститься ниже.

1. Переопределите метод **onTouchEvent ()** класса **GameView:**

```
    @Override
    public boolean onTouchEvent(MotionEvent event) {

        int eventAction = event.getAction();

        if (eventAction == MotionEvent.ACTION_DOWN)  {

            // Движение вверх
            if (event.getY() < playerBird.getBoundingBoxRect().top) {
                playerBird.setVy(-100);
                points--;
            }
            else if (event.getY() > (playerBird.getBoundingBoxRect().bottom)) {
                playerBird.setVy(100);
                points--;
            }

        }

        return true;
    }
```

При смене птицей направления полета расходуются очки:

points--;

2. Не позволяйте птице игрока вылететь за пределы экрана

Птица должна менять направление полета при соударении с верхней и нижней горизонтальной линией экрана.

Внутри метода update() после обновления состояния спрайтов:

```
. . .
if (playerBird.getY() + playerBird.getFrameHeight() > viewHeight) {

       playerBird.setY(viewHeight - playerBird.getFrameHeight());

       playerBird.setVy(-playerBird.getVy());
       points--;


}

else if (playerBird.getY() < 0) {

       playerBird.setY(0);

       playerBird.setVy(-playerBird.getVy());
       points--;
}
. . .
```

3. Добавьте метод возвращения птицы противника после пролета:

```
   private void teleportEnemy () {

       enemyBird.setX(viewWidth + Math.random() * 500);

       enemyBird.setY(Math.random() * (viewHeight - enemyBird.getFrameHeight()));

   }
```

4. Возвращение птицы противника в начальное положение осуществляется после пролета игрока

Внутри метода update():

```
   . . .

   if (enemyBird.getX() < - enemyBird.getFrameWidth()) {

        teleportEnemy ();

       points +=10;

   }
```

За облет птицы игроку начисляются очки.

5. Возвращение птицы противника в начальное положение осуществляется также после столкновения с птицей игрока

Внутри метода update():

```
// Проверка столкновений
if (enemyBird.intersect(playerBird)) {
```

```
        teleportEnemy ();
        points -= 40;

}
```

За столкновения с птицей у игрока снимаются очки.

Задание 3.7.1

Задания для самостоятельной работы:

1. Реализуйте “Бонусы”. Добавьте еще один вид спрайтов, столкновение с которыми добавляет пользователю очки.
2. Переход на другой уровень и окончание игры. При наборе определенного количества очков пользователь попадает на следующий уровень. С каждым следующим уровнем скорость полета птицы противника увеличивается. Номер уровня в игре должен отображаться на экране рядом с очками. После перехода на следующий уровень очки сбрасываются и отсчет начинается сначала. Если количество очков падает до определнного отрицательного уровня игра заканчивается.
3. Постановка игры на паузу. Придумайте способ постановки игры на паузу и снятие этого режима. В режиме паузы пользователь видит “замороженное” состояние игры с надписью “Пауза” в центре экрана.
4. Добавьте больше противников. Создайте больше птиц-противников, которых можно ликвидировать нажатием пальца.

Благодарности

Компания Samsung Electronics выражает благодарность за участие в подготовке данного материала преподавателю IT ШКОЛЫ SAMSUNG Максиму Анатольевичу Стрельцову.