Модуль 3. Основы программирования Android приложений

Тема 3.8. Разработка игровых приложений. SurfaceView

2 часа

Оглавление

3.8. Разработка игровых приложений. SurfaceView

Компьютерные игры – один из самых популярных способов использования компьютеров, смартфонов и планшетов.

Первые игры отличались простотой графики и программной логики. Но со временем игры становились все сложнее. Над их созданием работал уже не один программист, а целый коллектив разработчиков – программисты, дизайнеры, художники, аниматоры, специалисты по звуковым эффектам и люди других профессий. Для обеспечения эффективности разработки, ее ускорения и повышения качества игрового продукта в середине 90-х годов появился новый вид программного обеспечения для разработчиков – игровой движок.

3.8.1. Общие подходы для реализации игровых приложений

Последовательные этапы проектирования и реализации игрового приложения

В проектировании игрового приложения можно выделить некоторые этапы.

- **Идея**. Прежде всего, разработка игры начинается с идеи. Она может возникнуть случайно или в процессе долгого перебора. Можно отталкиваться от любимого жанра или от жанра, который популярен на данный момент. Возможно, ваша идея будет заимствована у другой игры или, напротив, представляет нечто совершенно новое.
- **Исследование**. Иногда для разработки игры нужно выяснить некоторые делали. Что, если вы разрбатываете симулятор спортивной игры? Вам нужно подробно ознакомиться с её правилами. Решили делать игру про пиратов или рыцарей? Не поленитесь подробно изучить тему. Скорее всего, в процессе найдется что-то новое. Это поможет привнести в игру оригинальные находки в сюжет (если он присутствует) или игровой процесс. Если в вашей игре вымышленный мир, то самое время подумать, по каким правилам он существует.
- **Разработка геймплея**. На этом этапе начинается проработка игрового процесса. Если на этапе идеи хватило формулировки “по реке плывет бегемот и уварачивается от препятствий”, то сейчас нужно знать, что за препятствия встречаются бегемоту (камни, другие животные), как игрок будет им управлять (гироскоп, swype-движения или касания по экрану), что будет с ним, если он столкнется с препятствием (игра начнется сначала или у бегемота есть несколько жизней), ну и, конечно, неплохо было бы знать куда он плывет. Как можно увидеть, геймплей охватывает множество мелочей. Важно всё хорошо продумать. Проработанный геймплей лучше заинтересовывает игрока. Но нужно не перестараться, например, введя пол сотни видов врагов.

После тщательной проработки геймлея, когда вы четко представляете, как ваша игра будет выглядеть для пользователя, можно приступать к следующему этапу.

- **Проектирование архитектуры.** Подходы к проектированию игровых приложений в целом не отличаются от остальных. Данная тема рассматривалась в главе 3.1. Главное на этом этапе определиться с необходимыми классами, рассмотреть возможность применения наследования, абстракции. К примеру, если мир в игре случайно генерируемый, нужно прорабоать алгоритм генерации, если в игре много уровней, которые создаются вручную, возможно, стоит задуматься о написании редактора уровней. Нет уникального рецепта проектирования архитектуры, многое зависит от конкретного приложения.
- **Реализация, тестирование**. Этот этап включает себя написание кода. Также сюда входит рисование графики, подбор звукового сопровождения, проектирование уровней. Другими словами, подготовка или создание игровых ресурсов и контента.

Существуют различные модели разработки программного обеспечения, и, в частности, игровых приложений. Выделяют такие модели как каскадную, итеративную, спиральную, V-модель разработки. К примеру, каскадная модель подразумевает, что разработка состоит из нескольких этапов и переход к следующему этапу осуществляется только после успешного выполнения предыдущего. То есть, до тех пор, пока проектирование полностью не закончено, нельзя приступать к написанию кода. (Нужно упомянуть, что данный подход устаревший и совершенно не гибкий).
Конечно, данные модели применяются в серьезных проектах и нет необходимости применять их в учебных приложениях.

- **Распространение, эксплуатация.** Для игровых мобильных приложений единственный путь распространения - это магазины приложений. Впервые эту идею в современном виде предложила компания Apple в 2008 году и с тех пор ни одна мобильная платформа не обходится без них.
  Например, до распространения Интернета, для десктопных приложений путь до клиента был дорогим и длинным: нужна была реклама, заводы наносили программы на дискеты, CD и DVD, а затем они рассылались по почте или продавались в магазинах. Сейчас разработчик выкладывает свою программу в соответствующий операционной системе магазин, и она становится доступной пользователям с любого уголка планеты, где есть доступ в Интернет.

- **Поддержка.** Зачастую поддержку называют продолжающейся разработкой. Почему? Посмотрите, например, отзывы о программах на Google Play. Пользователи оставляют свои замечания, авторы объясняют работу программы, исправляют ошибки, выходят новые версии и т.д. Хотя, надо заметить, бывают случаи, когда приложение не предполагает поддержку, т.е. разработчики по ряду причин могут прекратить работу над проектом.

Профессии в мире индустрии игр

Индустрия игр породила множество профессий. Во времена простых игр, когда игровая сфера только развивалась, в команде разработчиков вполне могло хватить одного геймдизайнера и программиста. Но современные проекты очень масштабны и поэтому требуют разделения труда. Программист не сможет заниматься всем сразу, разрабатывать искуственный интеллект и интерфейс, а также заниматься серверной частью. Для этого выделены специальности AI programmer, GUI programmer, Server programmer. Среди множества профессий игровой индустрии выделяются несколько основных.

- **Геймдизайнер**. Круг обязанностей геймдизайнера очень широк. Можно сказать, что он должен определить внешний вид и возможности игрового процесса.
- **Программист.** Среди программистов выделяют такие специализации как Core programmer, gui programmer, physics programmer, sound programmer и др.
- **Дизайнер, художник**. Профессии, связанные с разработкой интерфейса, графики для игры.
- **Тестировщик** - выискивает ошибки и следит за качеством продукта. Тестировщик моделирует различные ситуации, которые могут возникнуть в ходе игрового процесса и проверяет, всё ли работает правильно. Это очень важный и трудоемкий этап. По статистике тестирование требует в 4 раза больше ресурсов, чем разработка.

3.8.2. Понятие игрового движка и его использование при разработке игры

Профессиональные разработчики игр с графикой, обрабатываемой в реальном времени, не пишут игры “с нуля”. Они используют готовые игровые движки. Игровой движок - это центральный программный компонент приложения, который обеспечивает основные технологии и, зачастую, кроссплатформенность.

Изначально под игровым движком понимали подсистему обеспечения визуализации двумерной и, затем, трехмерной графики, создания анимации и визуальных эффектов. Позже движки стали обеспечивать поддержку физики игрового мира (физический движок), звука, сетевого взаимодействия, искусственного интеллекта.

Какие преимущества дает игровой движок разработчикам?

- Прежде всего, игровой движок создает каркас игрового приложения, делая код более организованным и управляемым. Не секрет, что современные игры содержат миллионы строчек кода. Продуманная архитектура игры, которая во многом обеспечивается движком, упрощает командную разработку, управление и поддержку кода из миллионов строк.
- Движок сокращает рутинные технические моменты реализации игровых процессов. Например, вместо изучения технических аспектов, обеспечивающих создание быстрой и плавной анимации, вызова множества библиотечных функций, можно использовать высокоуровневые функции движка, компактно решающих эту задачу.
- Движок позволяет сконцентрироваться на разработке игровой логики, позволяет мыслить более высокоуровневыми категориями.
- Особая ценность движка – переносимость, кроссплатформенность. Это, конечно, справедливо не для всех игровых движков. Но хорошо спроектированный игровой движок упрощает перенос игры на другие платформы.

Игровые движки - это сложные программы, которые невозможно освоить “с ходу”. Поэтому и говорят, что это инструмент для профессионалов.

3.8.3. Класс SurfaceView

В предыдущей главе было подробно рассмотрен способ рисования на View с помощью класса Canvas. Получить доступ к объекту Canvas можно в переопределенном методе onDraw класса View. Отметим некоторые особенности рисования с помощью класса View.

- Способ легок в реализации. Нужно только переопределить метод onDraw.
- Данный способ подходит для отрисовки небольшого количества кадров, так как рисование происходит в основном потоке.

Таким образом, рисование на View больше подходит для создания собственного элемента интерфейса, который выполняет специальные задачи (например, изображает график, небольшую анимацию). Также можно применять для создания игр, которые не требуют частой перерисовки, а рисование кадра не требует сложных вычислений (такие как шахматы, шашки, викторины).

Для более сложных задач рисования используется класс **SurfaceView**, который является подклассом View. SurfaceView представляет собой выделенную поверхность для рисования внутри иерархии View. Главное его отличие от View в том что, рисование на этой поверхности возможно предоставить дополнительному потоку приложения. Соответственно, приложение не должно ждать, пока иерархия View будет готова перерисоваться.

Доступ к поверхности рисования в параллельном потоке происходит не напрямую, он может быть получен с помощью объекта **SurfaceHolder**. Получить этот объект можно вызовом метода getHolder(), после инициализации SurfaceView. Этот объект нужно передать в дополнительный поток, чтобы в нем получить доступ к Canvas вашего SurfaceView. Создание и удаление потока нужно синхронизировать с созданием и удалением SurfaceView. Для этого нужно использовать интерфейс SerfaceHolder.Callback, который содержит методы serfaceCreate(SurfaceHolder) и serfaceDestroy(SurfaceHolder), чтобы отслеживать события создания и уничтожения SurfaceView. Нужно сделать так, чтобы доступ к поверхности осуществлялся только между этими событиями. Также интерфейс SerfaceHolder.Callback содержит метод, который реагирует на изменения SurfaceView.

`surfaceCreated (SurfaceHolder holder)`	Метод вызывается сразу после создания поверхности
`surfaceChanged (SurfaceHolder holder, int format, int width, int height)`	Метод вызывается после структурных изменений (формата или размера). Как минимум вызывается один раз после вызова метода surfaceCreated
`surfaceDestroyed (SurfaceHolder holder)`	Метод вызывается перед уничтожением поверхности.

3.8.4. Создание приложений

Перейдем к рассмотрению простого примера рисования с использованием класса SurfaceView.

Создание класса - наследника SurfaceView

Для начала нужно создать свой класс, унаследованный от класса `SurfaceView` с интерфейсом `SurfaceHolder.Callback` и переопределить один из конструкторов.

Создадим класс DrawView. Объект SurfaceHolder осуществляет доступ к поверхности для рисования. Для него нужно вызвать метод addCallback(SurfaceHolder.Callback). Это говорит ему, что он должен получать обратные вызовы от методов объекта Callback. Другими словами, должен быть в курсе, когда создается, изменяется и уничтожается поверхность. Для того, чтобы получить к нему доступ, нужно вызвать метод getHolder.

```
import android.content.Context;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class DrawView extends SurfaceView implements SurfaceHolder.Callback
{
   public DrawView(Context context) {
       super(context);
       getHolder().addCallback(this);
   }

   @Override
   public void surfaceCreated(SurfaceHolder holder) {
       // создание SurfaceView
   }

   @Override
   public void surfaceChanged(SurfaceHolder holder, int format,
                              int width, int height) {
       // изменение размеров SurfaceView
   }

   @Override
   public void surfaceDestroyed(SurfaceHolder holder) {
       // уничтожение SurfaceView
   }
}
```

Создание отдельного потока для рисования

Для рисования нужно создать отдельный поток. Сделать это можно, создав класс, и унаследовав его от Thread. Рисование будет происходить в методе run(), который нужно переопределить. Поток должен постоянно работать до тех пор, пока SurfaceView будет показываться на экране. Если этого делать больше не нужно, поток должен как-то узнать о завершении своей работы. Как

уже было сказано, доступ к поверхности для рисования хранит объект SurfaceHolder. Данный объект можно передать в класс DrawThread через его конструктор.

Итак, чтобы получить поверхность для рисования в дополнительном потоке, нужно вызвать метод **lockCanvas()** для SurfaceHolder. После этого на поверхности можно рисовать. После того, как необходимые методы рисования для Canvas были вызваны, нужно вызвать метод **unlockCanvasAndPost()** для объекта SurfaceHolder. После этого SurfaceView отрисует Canvas. При необходимости перерисовать кадр, нужно каждый раз вызывать эти методы lockCanvas() и unlockCanvasAndPost().

Метод **requestStop()** необходим для выхода из цикла, который находится внутри метода run(). Его вызов позволит в нужное время завершить поток.

```
public class DrawThread extends Thread {

    private SurfaceHolder surfaceHolder;

    private volatile boolean running = true;//флаг для остановки потока

    public DrawThread(Context context, SurfaceHolder surfaceHolder) {
        this.surfaceHolder = surfaceHolder;
    }

    public void requestStop() {
        running = false;
    }

    @Override
    public void run() {
        while (running) {
            Canvas canvas = surfaceHolder.lockCanvas();
            if (canvas != null) {
                try {
                    // рисование на canvas
                } finally {
                    surfaceHolder.unlockCanvasAndPost(canvas);
                }
            }
        }
    }
}
```

В классе DrawView нужно создать новое поле для дополнительного потока:

```
private DrawThread drawThread;
```

В переопределенный метод **surfaceCreate()** нужно добавить создание и вызов дополнительного пока, а также добавить его объявление в поле класса DrawView.

```
@Override
public void surfaceCreated(SurfaceHolder holder) {
```

```
    drawThread = new DrawThread(getContext(),getHolder());
    drawThread.start();
}
```

Перед удалением SurfaceView нужно позаботится о завершении дополнительного потока, так как перерисовывать уже будет нечего. Сделать это нужно в методе **SurfaceDestroy()**.

```
@Override
public void surfaceDestroyed(SurfaceHolder holder) {
    drawThread.requestStop();
    boolean retry = true;
    while (retry) {
        try {
            drawThread.join();
            retry = false;
        } catch (InterruptedException e) {
            //
        }
    }
}
```

В классе MainActivity в качестве параметра методу setContentView() нужно передать новый экземляр класса DrawView.

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(new DrawView(this));
    }
}
```

Затем в класс `DrawThread` добавляем методы рисования на Canvas.

Упражнение 3.8.1

Из полученной заготовки создадим приложение, в котором смайлик будет передвигаться по экрану по направлению к месту касания.

Итак, у нас уже реализован класс MainActivity, он остается без изменений.

Поместим графический файлик со смайликом **smile.png** в папку с ресурсами (в нашем проекте Android Studio - папка drawable):

В уже реализованном ранее классе DrawThread добавляем поля для хранения картинки **bitmap**, координат касания (**towardPointX, towardPointY**), устанавливаем цвет фона, определяем конструктор:

```
public class DrawThread extends Thread {

    private SurfaceHolder surfaceHolder;

    private volatile boolean running = true;
    private Paint backgroundPaint = new Paint();
    private Bitmap bitmap;
    private int towardPointX;
    private int towardPointY;
    {
    backgroundPaint.setColor(Color.BLUE);
    backgroundPaint.setStyle(Paint.Style.FILL);
    }

    public DrawThread(Context context, SurfaceHolder surfaceHolder) {
        bitmap = BitmapFactory.decodeResource(context.getResources(),
                                              R.drawable.smile);
        this.surfaceHolder = surfaceHolder;
    }
...
```

сеттер для координат:

```
public void setTowardPoint(int x, int y) {
   towardPointX = x;
   towardPointY = y;
}
```

и переопределим метод run():

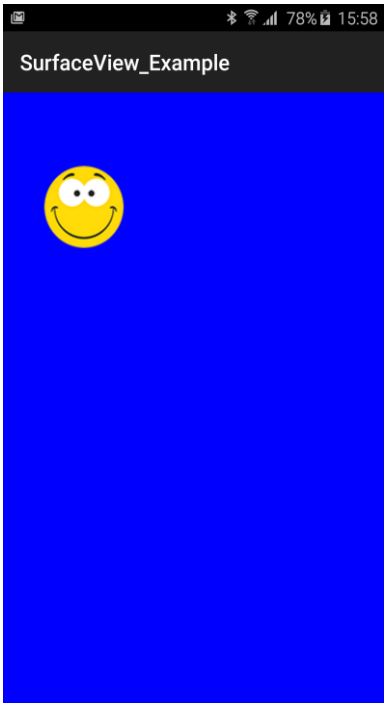
```
@Override
public void run() {
  int smileX = 0;
  int smileY = 0;
  while (running) {
      Canvas canvas = surfaceHolder.lockCanvas();
      if (canvas != null) {
        try {
            canvas.drawRect(0, 0, canvas.getWidth(), canvas.getHeight(),
                            backgroundPaint);
            canvas.drawBitmap(bitmap, smileX, smileY, backgroundPaint);

            if (smileX + bitmap.getWidth() / 2 < towardPointX) smileX+=10;
            if (smileX + bitmap.getWidth() / 2 > towardPointX) smileX-=10;
            if (smileY + bitmap.getHeight() / 2 < towardPointY) smileY+=10;
            if (smileY + bitmap.getHeight() / 2 > towardPointY) smileY-=10;
        } finally {
            surfaceHolder.unlockCanvasAndPost(canvas);
        }

      }
  }
}
```

Здесь переменные smileX и smileY задают изменение позиции смайлика при перерисовке поверхности в зависимости от координат касания.

И наконец, в классе DrawView переопределим унаследованный метод **onTouchEvent()**, который будет передавать координаты касания экрана классу drawThread:

```
@Override
public boolean onTouchEvent(MotionEvent event) {
      drawThread.setTowardPoint((int)event.getX(),(int)event.getY());
      return false;
}
```

Запускаем проект и получим требуемое приложение (полностью проект приведен в архиве к теме 3.8. SurfaceView_Example).



Задание 3.8.1

Посмотрите запись вебинара по SurfaceView. Разберитесь в разработанном проекте UfoDestroyer (архив в материалах курса).

Задание 3.8.2

Используя класс SurfaceView, модифицируйте игровое приложение с птичками, реализованное в предыдущей теме 3.7.

Благодарности

Компания Samsung Electronics выражает благодарность за участие в подготовке данного материала преподавателю IT ШКОЛЫ SAMSUNG Вихрову Виктору Андреевичу.