- Часть 1 - Простое приложение для Android
- Часть 2 - Переходы между формами
- Часть 3 - Использование диалогов
- Часть 4 - Использование GridView

Знакомство с Android. Часть 1: Простое приложение для Android

Недавно заинтересовала меня платформа Android. Как-то много говорят о нем в последнее время, да и вообще хотелось узнать, такая же ли там ужасная Java, как в мидлетах. Так что потратила я некоторое время на копание в нем, написала простое приложение, и сейчас вот буду делиться опытом.

Постановка задачи

Первым нашим приложением для Android будет реализация всем известной игры Life. Местом дейтвия будет прямоугольное клеточное поле, размеры которого запрашиваются у пользователя. Также у пользователя запрашивается начальное количество клеток. Первое поколение расставляется по карте случайным образом. Последующие поколения получаются по следующим правилам:

- Если у живой клетки меньше двух или больше трёх соседей, то она погибает.
- Если у пустой клетки ровно три соседки, она оживает.

Все входные параметры должны проверяться на правильность: столбцов должно быть не меньше 5 и не больше 25, строк должно быть не меньше 5 и не больше 35, начальное количество клеток должно быть не больше, чем ячеек на поле. Для реализации поля будет использован класс GridView.

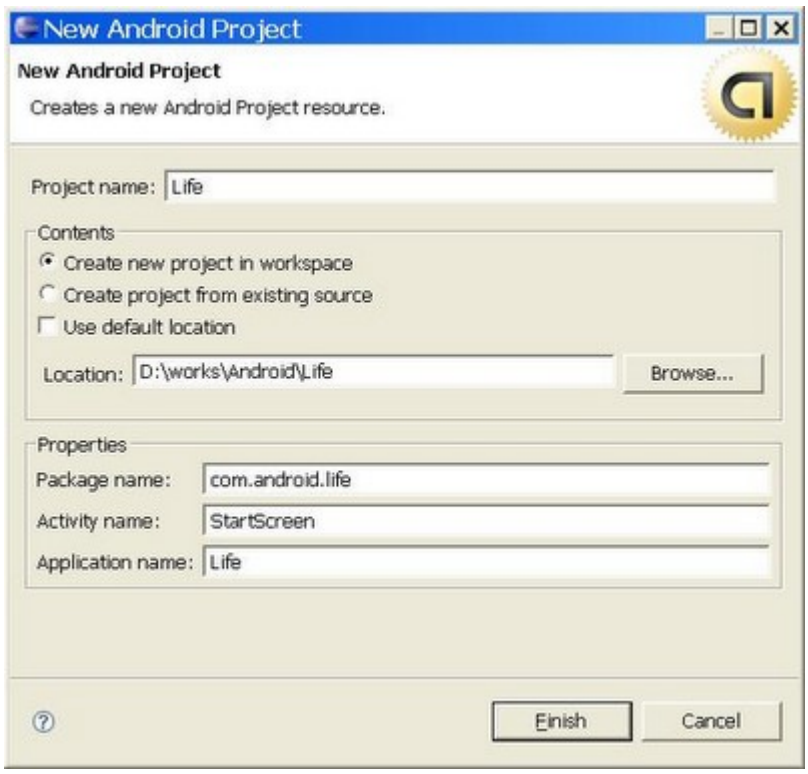
Для разработки была использована среда **Eclipse** и **Android plugin** для неё.

В этой части

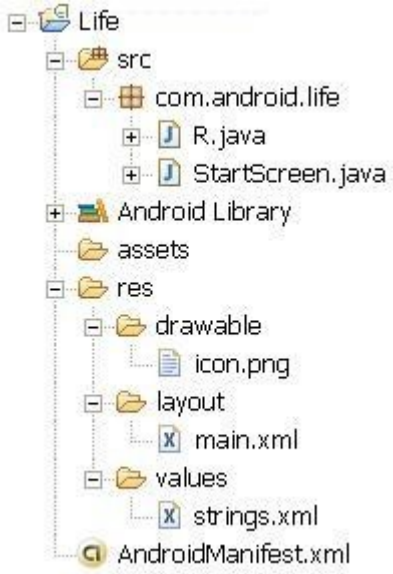
Мы создадим проект, рассмотрим его структуру и напишем простое приложение, состоящее из одной формы. На форме будет интерфейс для ввода данных и кнопка Run.

Создание и обзор проекта

На установке Android SDK и плагина для Eclipse останавливаться не будем, т.к. это достаточно подробно описано в официальном мануале. Создаем в Eclipse новый **Android Project**:



После нажатия на кнопку **Finish** создастся новый проект с такой структурой файлов:



Рассмотрим эту структуру внимательнее.

/res/drawable

Сюда помещаются все графические файлы, используемые в приложении. На данный момент там есть только файл icon.png - главная иконка приложения.

/res/layout

В эту папку помещаются файлы, в которых в формате XML описывается внешний вид форм, расположение контролов и т.д. (как dfm-ки в Дельфи). Плагин даже создал разметку для нашей единственной формы и назвал её main.xml. Позже мы рассмотрим ее подробнее.

/res/values

В этой папке хранятся общие константы для всего приложения, как то: текст, используемый элементами управления, цвета, стили и т.д.. Например, если мы хотим вывести "Hello World" в TextView, можно это сделать явно в разметке, как мы всю жизнь делали в тех же dfm-ках или aspx; либо создать в `strings.xml` константу `hello` со значением "Hello World", после чего пойти обратно в разметку и в атрибутах этого TextView прописать `android:text="@string/hello"`.

AndroidManifest.xml

В этом файле перечисляются общие свойства проекта (версия, package и прочее), а также все формы (Activities), входящие в проект.

R.java

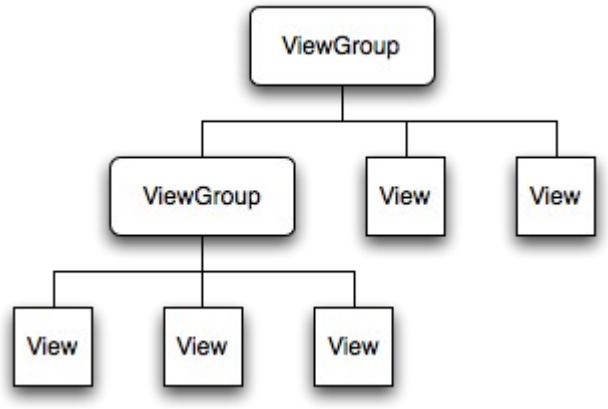
Это такой специальный сгенерированный класс, посредством которого осуществляется доступ к ресурсам приложения (т.е. ко всему тому, что есть в папке `res`). Например, `R.string.hello` возвращает константу с именем `hello` из `strings.xml`.

StartScreen.java

Это нам плагин сгенерировал класс для главной (и пока что единственной) формы приложения. Там пока содержится единственный обработчик `onCreate`, и написано там только `setContentView(R.layout.main);`. С помощью этой строчки к данной форме привязывается разметка, описанная в файле `/res/layout/main.xml`

Разметка формы (Layout)

Элементы управления в Android называются **Views** и наследуются от класса `View` или `ViewGroup`. Класс `ViewGroup` также унаследован от `View`, но его отличие в том, что в него могут быть вложены другие `View` или `ViewGroup`.



Плагин создал простейшую разметку для нашей единственной формы (`main.xml`):

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
```

```
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
    />
</LinearLayout>
```

Вначале задаётся **Layout**, т.е. правило, согласно которому элементы управления следуют друг за другом. `LinearLayout` значит, что они идут друг за другом сверху вниз (`android:orientation="vertical"`). Бывают и другие Layout-ы: `TableLayout`, с помощью которого можно выстроить контролы в таблицу; `FrameLayout`, который ставит контролы один на другой; и т.д.

Мы воспользуемся `TableLayout`

Сделаем вот такую разметку:

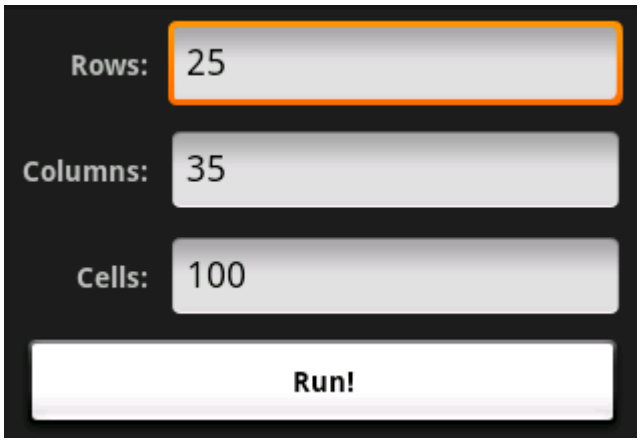
```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:stretchColumns="1"
    android:padding="10dip"
    >
    <TableRow android:paddingBottom="5dip">
        <TextView
            android:text="@string/rows_title"
            android:paddingRight="10dip"
            android:gravity="right"
            android:textStyle="bold"
        />
        <EditText
            android:id="@+id/RowsEditor"
            android:text="35"
            android:singleLine="true"
            android:numeric="integer"
        />
    </TableRow>
    <TableRow android:paddingBottom="5dip">
        <TextView
            android:text="@string/columns_title"
            android:paddingRight="10dip"
            android:gravity="right"
            android:textStyle="bold"
        />
        <EditText
            android:id="@+id/ColumnsEditor"
            android:text="25"
            android:singleLine="true"
            android:numeric="integer"
        />
    </TableRow>
    <TableRow android:paddingBottom="5dip">
        <TextView
            android:text="@string/cells_title"
            android:paddingRight="10dip"
            android:gravity="right"
            android:textStyle="bold"
        />
        <EditText
            android:id="@+id/CellsEditor"
            android:text="100"
            android:singleLine="true"
            android:numeric="integer"
```

```
        />
    </TableRow>
    <TableRow>
        <Button
            android:id="@+id/RunButton"
            android:text="@string/run_title"
            android:textStyle="bold"
            android:layout_span="2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
        />
    </TableRow>
</TableLayout>
```

В файл `strings.xml` при этом нужно добавить следующие строки:

```
<string name="run_title">Run!</string>
<string name="columns_title">Columns:</string>
<string name="rows_title">Rows:</string>
<string name="cells_title">Cells:</string>
```

Форма при этом будет выглядеть так:



Рассмотрим некоторые атрибуты, использованные в разметке

android:id

Идентификатор элемента. Если он указан, то в дальнейшем его можно найти на форме с помощью метода `findViewById(id)`. Для контролов, которых мы не планируем в дальнейшем трогать (например, для заголовков), можно это свойство и вовсе не указывать. Идентификаторы можно складывать в файл `ids.xml`, но вместо этого обычно применяется синтаксис `@+id/View1`. Это означает, что идентификатор `View1` добавляется в константы прямо на ходу. В `R.java` соответствующие поля также добавляются автоматически.

android:layout_width и android:layout_heigth

Свойства `layout_width` и `layout_heigth` обозначают, какую часть родительского контрола будет занимать данный элемент управления: всю (`fill_parent`) или ровно столько, сколько требуется (`wrap_content`).

android:numeric

Это атрибут `EditText`. Значение `integer` значит, что в это поле можно вводить только целые положительные числа.

android:gravity

Устанавливает выравнивание текста в данном элементе управления.

Заключение

Итак, мы создали проект для Android, рассмотрели его структуру, составили разметку для нашей единственной формы.

Ссылки

- Android documentation
- Пишем HelloWorld под Android

Исходники примера

Знакомство с Android. Часть 2: Переходы между формами

Итак, продолжим. В этой части мы добавим в проект ещё одну форму и будем открывать её по нажатию кнопки `Run`. Также сделаем так, чтобы параметры, которые были введены в первой форме, передавались во вторую (они там ещё пригодятся). Однако, ничего страшного мы пока с ними делать не будем, а просто напишем "Введены такие-то числа".

Обработка нажатия кнопки

Напишем в `StartScreen.java` следующее (строки, которые мы добавляем, выделены другим цветом):

```
package com.android.life;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class StartScreen extends Activity implements OnClickListener
{
    Button mButton;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mButton = (Button) findViewById(R.id.RunButton);
        mButton.setOnClickListener(this);
    }
    public void onClick(View v)
    {
        mButton.setText("It works!");
    }
}
```

Здесь выделены добавленные строки. Итак, мы сделали форму слушателем события нажатия на кнопку (имплементировав `OnClickListener`), и подписались на это событие для

кнопки `mButton` (с помощью `setOnClickListener`). Для того, чтобы проверить, что всё правильно, в методе, который обрабатывает событие, изменяем заголовок кнопки. Запускаем - работает.

Переход на другую форму

Итак, добавляем еще одну форму. Для этого надо создать два файла - разметки формы(`run.xml`) и класса формы (`RunScreen.java`).

Разметку сделаем простейшую:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
<TextView
    android:id="@+id/Message"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
/>
</FrameLayout>
```

В классе добавим `onCreate`, в котором свяжем форму с этой разметкой:

```
package com.android.life;

import android.app.Activity;
import android.os.Bundle;

public class RunScreen extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.run);
    }
}
```

Кроме того, нужно упомянуть о новой форме в `AndroidManifest.xml`. Для этого в раздел `application` добавляем следующее:

```
<activity android:name=".RunScreen">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.SAMPLE_CODE" />
    </intent-filter>
</activity>
```

Нам хочется, чтобы это окно появлялось после нажатия кнопки "Run". Пишем в обработчике этой кнопки следующее:

```
public void onClick(View v)
{
    Intent intent = new Intent();
    intent.setClass(this, RunScreen.class);

    startActivity(intent);
    finish();
}
```

Запускаем - работает. Теперь объясним, что сделали.

Intent

`Intent` - это описание некоторого действия, которое должно совершить приложение. Этим действием может быть переход к другому окну, совершение исходящего вызова, открытие списка контактов и т.д.. У Intent-а есть два основных атрибута:

- **action** - функция, которую надо выполнить. Задается в виде константы: `ACTION_VIEW`, `ACTION_DIAL` и т.д.
- **data** - аргумент этой функции, записанный в виде URI

Например, сочетание `VIEW_ACTION content://contacts/people/1` соответствует выводу информации о контакте с идентификатором 1, а `ACTION_DIAL tel:123` - выводу окна вызова с набранными цифрами `123`.

Однако, нам всего этого не нужно, а нужно, чтобы обработчиком Intent-а была сама форма `RunScreen`. Для этого мы используем функцию `setClass` (или есть ещё такая `setClassName`).

Вызов функции startActivity(intent) запускает операцию `intent` (т.е. открывает окно), `finish()` завершает текущую задачу (т.е. закрывает окно `StartScreen`).

Передача данных между окнами

Итак, окно `RunScreen` появляется, но на нем ничего нет. Рассмотрим, каким образом можно передать параметры, введенные в `StartScreen`, в окно `RunScreen`.

У класса `Intent`, помимо основных атрибутов, есть ещё и второстепенные. В частности, есть такой атрибут `extras`, где в виде хеша (`Bundle`) хранятся любые дополнительные параметры. Им-то мы и воспользуемся. Итак, добавим в класс `RunScreen.java` следующие константы (ключи хеша):

```
public static final String EXT_COLS = "cols";
public static final String EXT_ROWS = "rows";
public static final String EXT_CELLS = "cells";
```

Теперь модифицируем обработчик `onClick` в классе `StartScreen.java`, чтобы он выглядел так:

```
public void onClick(View v)
{
    EditText rowsEditor = (EditText)findViewById(R.id.RowsEditor);
    EditText colsEditor = (EditText)findViewById(R.id.ColumnsEditor);
    EditText cellsEditor = (EditText)findViewById(R.id.CellsEditor);

    int cols = Integer.parseInt(colsEditor.getText().toString());
    int rows = Integer.parseInt(rowsEditor.getText().toString());
    int cells = Integer.parseInt(cellsEditor.getText().toString());

    Intent intent = new Intent();
    intent.setClass(this, RunScreen.class);

    intent.putExtra(RunScreen.EXT_COLS, cols);
    intent.putExtra(RunScreen.EXT_ROWS, rows);
    intent.putExtra(RunScreen.EXT_CELLS, cells);

    startActivity(intent);
    finish();
```

```
}
```

Итак, мы передали в наш Intent дополнительные параметры. Теперь мы получим их в `RunScreen` в методе `onCreate`:

```
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.run);
    Bundle extras = getIntent().getExtras();
    int cols = extras.getInt(EXT_COLS);
    int rows = extras.getInt(EXT_ROWS);
    int cells = extras.getInt(EXT_CELLS);

    TextView message = (TextView)findViewById(R.id.Message);
    message.setText("Rows: " + rows + "\nColumns: " + cols + "\nCells: " +
cells);
}
```

Теперь, после нажатия на кнопку **Run** мы видим следующее:

Заключение

Итак, мы добавили в наше приложение ещё одну форму, создали переход на неё и передали параметры из первой формы.

Ссылки

- Android documentation
- Android: Playing with Intents
- Проектирование интерфейса, класс переходов - Intent
- Добрый монстр Android

Исходники примера

Знакомство с Android. Часть 3: Использование диалогов

В этой части мы сделаем проверку вводимых параметров по следующим правилам:

- Число столбцов должно быть не меньше 5 и не больше 25.
- Число строк должно быть не меньше 5 и не больше 35.
- Начальное количество клеток должно быть не больше, чем ячеек на поле.

Если какое-то из этих условий не выполняется, будем выводить соответствующее предупреждение

Кроме того, мы добавим кнопку **Close**, при нажатии на которую приложение будет закрываться, спрашивая вначале согласие пользователя.

Простой диалог

Сначала добавим в класс `StartScreen.java` следующие константы:

```
// код результата проверки
private static final int ALERT_NONE = 0; // параметры введены верно
private static final int ALERT_COLUMNS = 1; // некорректное число столбцов
private static final int ALERT_ROWS = 2; // некорректное число строк
private static final int ALERT_CELLS = 3; // некорректное начальное число клеток

// границы допустимых значений числа столбцов
private static final int COLUMNS_MIN = 5;
private static final int COLUMNS_MAX = 25;

// границы допустимых значений числа строк
private static final int ROWS_MIN = 5;
private static final int ROWS_MAX = 35;
```

Теперь добавляем метод, осуществляющий проверку введенных параметров:

```
private int checkInputParameters(int cols, int rows, int cells)
{
    if (cols < COLUMNS_MIN || cols > COLUMNS_MAX)
    {
        return ALERT_COLUMNS;
    }

    if (rows < ROWS_MIN || rows > ROWS_MAX)
    {
        return ALERT_ROWS;
    }

    if (cells > rows * cols)
    {
        return ALERT_CELLS;
    }

    return ALERT_NONE;
}
```

Метод `onClick` теперь должен работать немного по-другому: сначала проверять введенные параметры, и, если проверка прошла (`ALERT_NONE`), открывать окно `RunScreen`, иначе же показывать предупреждение. Итак, `onClick` будет выглядеть так:

```
public void onClick(View v)
{
    EditText rowsEditor = (EditText)findViewById(R.id.RowsEditor);
    EditText colsEditor = (EditText)findViewById(R.id.ColumnsEditor);
    EditText cellsEditor = (EditText)findViewById(R.id.CellsEditor);

    int cols = Integer.parseInt(colsEditor.getText().toString());
    int rows = Integer.parseInt(rowsEditor.getText().toString());
    int cells = Integer.parseInt(cellsEditor.getText().toString());
    int alertCode = checkInputParameters(cols, rows, cells);
    if (alertCode != ALERT_NONE)
    {
        showDialog(alertCode);
        return;
    }
    Intent intent = new Intent();
    intent.setClass(this, RunScreen.class);

    intent.putExtra(RunScreen.EXT_COLS, cols);
```

```
    intent.putExtra(RunScreen.EXT_ROWS, rows);
    intent.putExtra(RunScreen.EXT_CELLS, cells);

    startActivity(intent);
    finish();
}
```

Метод `showDialog(id)` класса `Activity` пытается открыть диалог. При этом вызывается метод `onCreateDialog`, возвращающий объект класса `Dialog`. Так что нужно перекрыть метод `onCreateDialog` и сконструировать тот диалог, который нам нужен. Ну у нас тут всё просто, диалоги будут отличаться только выводимым сообщением, так что можно сделать следующее:

```
@Override
protected Dialog onCreateDialog(int id)
{
    DialogInterface.OnClickListener doNothing = new
DialogInterface.OnClickListener()
    {
        public void onClick(DialogInterface dialog, int whichButton)
        {
        }
    };
    int alertMessage;

    switch (id)
    {
        case ALERT_COLUMNS:
            alertMessage = R.string.alert_columns;
            break;
        case ALERT_ROWS:
            alertMessage = R.string.alert_rows;
            break;
        case ALERT_CELLS:
            alertMessage = R.string.alert_cells;
            break;
        default:
            return null;
    }

    return new AlertDialog.Builder(this)
        .setMessage(alertMessage)
        .setNeutralButton(R.string.ok, doNothing)
        .create();
}
```

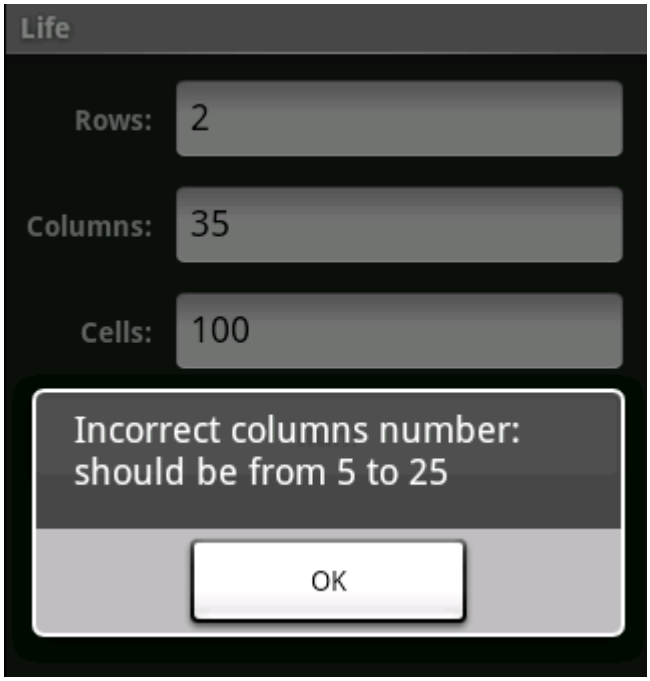
В `strings.xml` при этом нужно добавить следующие значения:

```
<string name="alert_columns">Incorrect columns number: should be from 5 to
25</string>
<string name="alert_rows">Incorrect rows number: should be from 5 to 35</string>
<string name="alert_cells">Number of cells is greater then the size of
grid</string>
<string name="ok">OK</string>
```

Итак, в методе `onCreateDialog` мы решили, какое сообщение (`alertMessage`) будем выводить, создали диалог с помощью `AlertDialog.Builder`, вывели туда сообщение, поставили единственную кнопку `OK` и привязали к ней обработчик `doNothing`, который ничего не делает (нам, в принципе, делать ничего и не надо). Аналогичным образом можно добавить иконку формы, добавить ещё кнопок и т.д.

Подробнее про то, как строятся диалоги, можно посмотреть в адроидовских сэмплах - там на самом деле все достаточно понятно и просто.

Теперь при вводе некорректных данных мы видим вот это:



Диалог для закрытия приложения

Добавим на форму `RunScreen` кнопку **Close**. Для этого напишем такую разметку в `run.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:id="@+id/Message"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:paddingBottom="10dip"
    />
    <Button
        android:id="@+id/CloseButton"
        android:text="@string/close"
        android:textStyle="bold"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
    />
</LinearLayout>
```

В `strings.xml` добавим следующие значения:

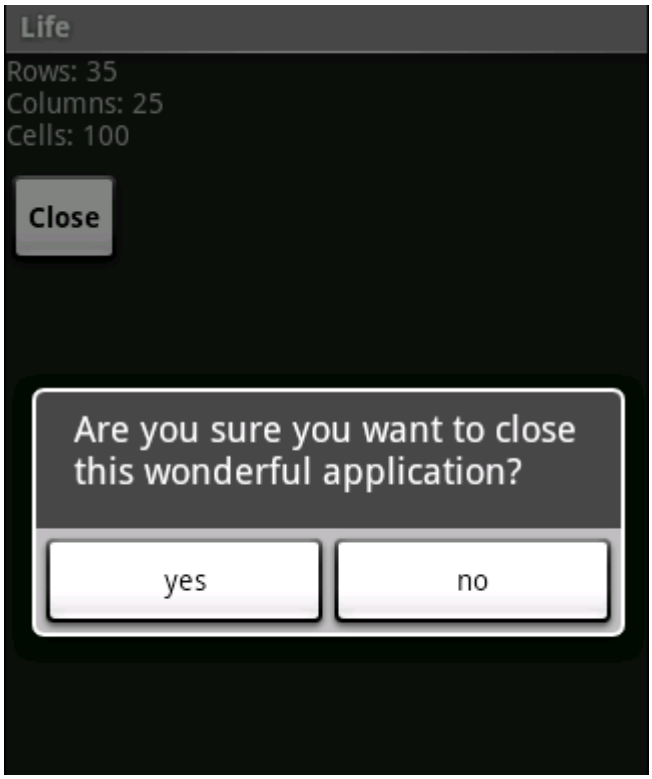
```
<string name="yes">yes</string>
<string name="no">no</string>
<string name="close">Close</string>
<string name="submit_close">Are you sure you want to close this wonderful
application?</string>
```

Теперь вносим изменения в класс `RunScreen`:

```
public class RunScreen extends Activity implements OnClickListener
{
   private static final int ALERT_CLOSE = 1;

   public static final String EXT_COLS = "cols";
   public static final String EXT_ROWS = "rows";
   public static final String EXT_CELLS = "cells";

   Button mCloseButton;
   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.run);
      mCloseButton = (Button) findViewById(R.id.CloseButton);
      mCloseButton.setOnClickListener(this);
      Bundle extras = getIntent().getExtras();
      int cols = extras.getInt(EXT_COLS);
      int rows = extras.getInt(EXT_ROWS);
      int cells = extras.getInt(EXT_CELLS);

      TextView message = (TextView)findViewById(R.id.Message);
      message.setText("Rows: " + rows + "\nColumns: " + cols + "\nCells: " +
cells);
   }
   /*
   * @see android.view.View.OnClickListener#onClick(android.view.View)
   */
   @Override
   public void onClick(View arg0)
   {
      showDialog(ALERT_CLOSE);
   }
   @Override
   protected Dialog onCreateDialog(int id)
   {
      switch (id)
      {
         case ALERT_CLOSE:
            return new AlertDialog.Builder(this)
            .setMessage(R.string.submit_close)
            // кнопка "Yes", при нажатии на которую приложение закроется
            .setPositiveButton(R.string.yes, new
DialogInterface.OnClickListener()
               {
                  public void onClick(DialogInterface dialog, int whichButton)
                  {
                     finish();
                  }
               })
            // кнопка "No", при нажатии на которую ничего не произойдет
            .setNegativeButton(R.string.no, new
DialogInterface.OnClickListener()
               {
                  public void onClick(DialogInterface dialog, int whichButton)
                  {

                  }
               })
            .create();
         default:
```

```
                break;
        }
        return null;
    }
}
```

Объяснять тут особо нечего, и так все понятно. Теперь, при нажатии на кнопку **Close**, мы видим следующее:



При нажатии на **yes** приложение закроется, на **no** - продолжит работу.

Ссылки

- Android documentation
- Как использовать alert-диалоги

Исходники примера

Знакомство с Android. Часть 4: Использование GridView

Итак, в нашем приложении осталось всего ничего: реализовать собственно алгоритм игры Life и отобразить его в GridView. Этим-то мы сейчас и займёмся.

Класс, реализующий логику Life

Добавим в проект новый класс, назовем его `LifeModel`. Тут у нас будет реализована вся логика Life

```
package com.android.life;

import java.util.Random;
public class LifeModel
```

```
{
   // состояния клетки
   private static final Byte CELL_ALIVE = 1; // клетка жива
   private static final Byte CELL_DEAD = 0; // клетки нет

   // константы для количества соседей
   private static final Byte NEIGHBOURS_MIN = 2; // минимальное число соседей
для живой клетки
   private static final Byte NEIGHBOURS_MAX = 3; // максимальное число соседей
для живой клетки
   private static final Byte NEIGHBOURS_BORN = 3; // необходимое число соседей
для рождения клетки

   private static int mCols; // количество столбцов на карте
   private static int mRows; // количество строк на карте
   private Byte[][] mCells; // расположение очередного поколения на карте.
                        //Каждая ячейка может содержать либо CELL_ACTIVE, либо
CELL_DEAD

   /**
   * Конструктор
   */
   public LifeModel(int rows, int cols, int cellsNumber)
   {
      mCols = cols;
      mRows = rows;
      mCells = new Byte[mRows][mCols];

      initValues(cellsNumber);
   }

   /**
   * Инициализация первого поколения случайным образом
   * @param cellsNumber количество клеток в первом поколении
   */
   private void initValues(int cellsNumber)
   {
      for (int i = 0; i < mRows; ++i)
         for (int j = 0; j < mCols; ++j)
            mCells[i][j] = CELL_DEAD;

      Random rnd = new Random(System.currentTimeMillis());
      for (int i = 0; i < cellsNumber; ++i)
      {
         int cc;
         int cr;
         do
         {
            cc = rnd.nextInt(mCols);
            cr = rnd.nextInt(mRows);
         }
         while (isCellAlive(cr, cc));
         mCells[cr][cc] = CELL_ALIVE;
      }
   }

   /**
   * Переход к следующему поколению
   */
   public void next()
   {
      Byte[][] tmp = new Byte[mRows][mCols];

      // цикл по всем клеткам
```

```
        for (int i = 0; i < mRows; ++i)
            for (int j = 0; j < mCols; ++j)
            {
                // вычисляем количество соседей для каждой клетки
                int n =
                    itemAt(i-1, j-1) + itemAt(i-1, j) + itemAt(i-1, j+1) +
                    itemAt(i, j-1) + itemAt(i, j+1) +
                    itemAt(i+1, j-1) + itemAt(i+1, j) + itemAt(i+1, j+1);

                tmp[i][j] = mCells[i][j];
                if (isCellAlive(i, j))
                {
                    // если клетка жива, а соседей у нее недостаточно или слишком
много, клетка умирает
                    if (n < NEIGHBOURS_MIN || n > NEIGHBOURS_MAX)
                        tmp[i][j] = CELL_DEAD;
                }
                else
                {
                    // если у пустой клетки ровно столько соседей, сколько нужно,
она оживает
                    if (n == NEIGHBOURS_BORN)
                        tmp[i][j] = CELL_ALIVE;
                }
            }
        mCells = tmp;
    }

    /**
    * @return Размер поля
    */
    public int getCount()
    {
        return mCols * mRows;
    }

    /**
    * @param row Номер строки
    * @param col Номер столбца
    * @return Значение ячейки, находящейся в указанной строке и указанном
столбце
    */
    private Byte itemAt(int row, int col)
    {
        if (row < 0 || row >= mRows || col < 0 || col >= mCols)
            return 0;

        return mCells[row][col];
    }

    /**
    * @param row Номер строки
    * @param col Номер столбца
    * @return Жива ли клетка, находящаяся в указанной строке и указанном столбце
    */
    public Boolean isCellAlive(int row, int col)
    {
        return itemAt(row, col) == CELL_ALIVE;
    }

    /**
    * @param position Позиция (для клетки [row, col], вычисляется как row *
mCols + col)
    * @return Жива ли клетка, находящаяся в указанной позиции
```

```
    */
    public Boolean isCellAlive(int position)
    {
        int r = position / mCols;
        int c = position % mCols;

        return isCellAlive(r,c);
    }
}
```

GridView. Отображение первого поколения клеток

Модифицируем разметку `run.xml` так, чтобы она выглядела следующим образом:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <GridView
        android:id="@+id/LifeGrid"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"

        android:padding="1dp"
        android:verticalSpacing="1dp"
        android:horizontalSpacing="1dp"
        android:columnWidth="10dp"

        android:gravity="center"
    />
    <Button
        android:id="@+id/CloseButton"
        android:text="@string/close"
        android:textStyle="bold"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
    />
</LinearLayout>
```

Теперь нам надо отобразить в этом GridView данные. Думаю, вполне логичным для данной задачи было бы отображение клеток в виде графических файлов. Создаем два графических файлика, на одном изображаем черный квадратик, на другом - зелёный. Первый назовём `empty.png` и он будет обозначать пустую клетку, второй - `cell.png`, и он будет изображать живую клетку. Оба файлика положим в папку `/res/drawable`

Нам нужно знать, что именно отображать в гриде. Для этого нужно создать для грида поставщик данных (`Adapter`). Есть стандартные классы для адаптеров (`ArrayAdapter` и др.), но нам будет удобнее написать свой, унаследованный от `BaseAdapter`. Дабы не плодить файлов (да и не нужен он больше никому), поместим его внутрь класса `RunScreen`. А напишем там следующее:

```
public class LifeAdapter extends BaseAdapter
{
    private Context mContext;
    private LifeModel mLifeModel;

    public LifeAdapter(Context context, int cols, int rows, int cells)
    {
        mContext = context;
```

```
        mLifeModel = new LifeModel(rows, cols, cells);
    }

    public void next()
    {
        mLifeModel.next();
    }

    /**
    * Возвращает количество элементов в GridView
    */
    @Override
    public int getCount()
    {
        return mLifeModel.getCount();
    }

    /**
    * Возвращает объект, хранящийся под номером position
    */
    @Override
    public Object getItem(int position)
    {
        return mLifeModel.isCellAlive(position);
    }

    /**
    * Возвращает идентификатор элемента, хранящегося в под номером position
    */
    @Override
    public long getItemId(int position)
    {
        return position;
    }

    /**
    * Возвращает элемент управления, который будет выведен под номером position
    */
    @Override
    public View getView(int position, View convertView, ViewGroup parent)
    {
        ImageView view; // выводиться у нас будет картинка

        if (convertView == null)
        {
            view = new ImageView(mContext);

            // задаем атрибуты
            view.setLayoutParams(new GridView.LayoutParams(10, 10));
            view.setAdjustViewBounds(false);
            view.setScaleType(ImageView.ScaleType.CENTER_CROP);
            view.setPadding(1, 1, 1, 1);
        }
        else
        {
            view = (ImageView)convertView;
        }

        // выводим черный квадратик, если клетка пустая, и зеленый, если она жива
        view.setImageResource(mLifeModel.isCellAlive(position) ?
R.drawable.cell : R.drawable.empty);

        return view;
    }
```

```
}
```

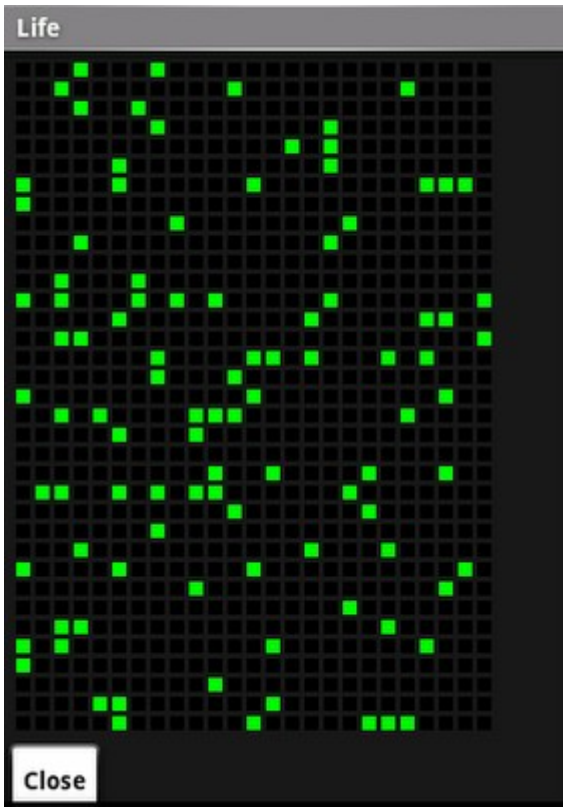
Теперь добавим в класс поля:

```
private GridView mLifeGrid;
private LifeAdapter mAdapter;
```

и модифицируем `onCreate`:

```
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.run);

    mCloseButton = (Button) findViewById(R.id.CloseButton);
    mCloseButton.setOnClickListener(this);

    Bundle extras = getIntent().getExtras();
    int cols = extras.getInt(EXT_COLS);
    int rows = extras.getInt(EXT_ROWS);
    int cells = extras.getInt(EXT_CELLS);
    mAdapter = new LifeAdapter(this, cols, rows, cells);

    mLifeGrid = (GridView)findViewById(R.id.LifeGrid);
    mLifeGrid.setAdapter(mAdapter);
    mLifeGrid.setNumColumns(cols);
    mLifeGrid.setEnabled(false);
    mLifeGrid.setStretchMode(0);
}
```

Запускаем и видим:



Отображение последующих поколений

Вот мы и добрались почти до самого конца. Осталось отобразить ход игры. Тут стоит

воспользоваться таймером. Таймер будет каждую секунду вызывать обработчик, в котором данные в адаптере будут пересчитываться. Сначала добавим в `RunScreen` поле:

```
private Timer mTimer;
```

а в `onCreate` - такой код:

```
mTimer = new Timer("LifeTimer");
mTimer.scheduleAtFixedRate(new SendMessageTask(), 0, 500);
```

`SendMessageTask` - это класс-обработчик таймера. Мы определим его прямо в классе `RunScreen` следующим образом:

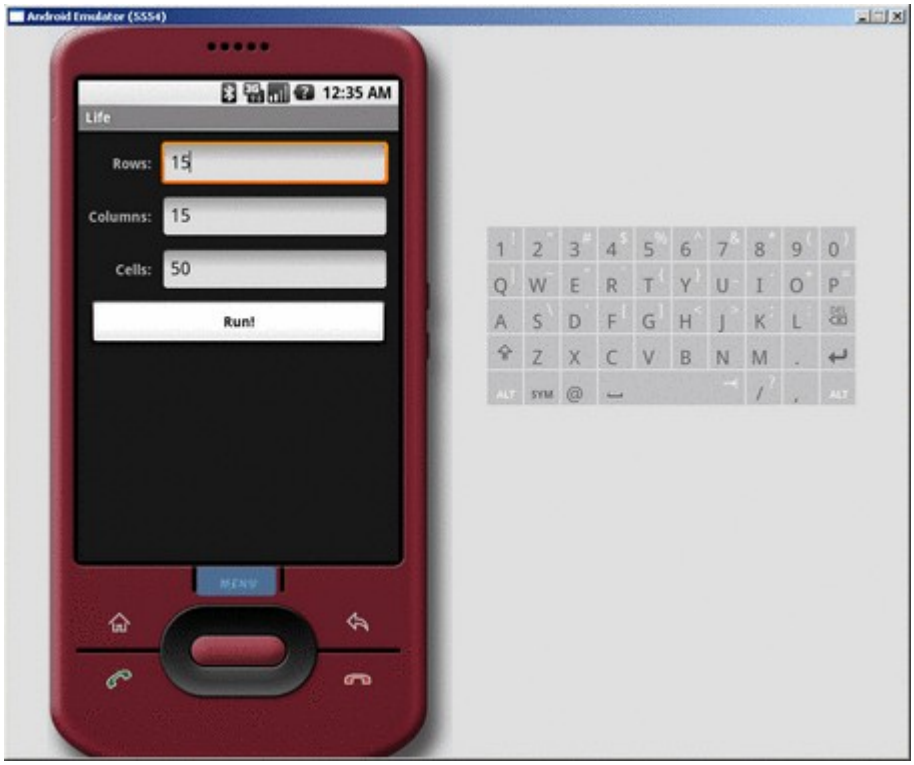
```
public class SendMessageTask extends TimerTask
{
    /**
    * @see java.util.TimerTask#run()
    */
    @Override
    public void run()
    {
        Message m = new Message();
        RunScreen.this.updateGridHandler.sendMessage(m);
    }
}
```

В `RunScreen` же добавим такую конструкцию:

```
Handler updateGridHandler = new Handler()
{
    public void handleMessage(Message msg)
    {
        mAdapter.next();
        mLifeGrid.setAdapter(mAdapter);

        super.handleMessage(msg);
    }
};
```

Таким образом, по таймауту мы посылаем нашей же форме сообщение, что пора обновиться, и в только обработчике этого сообщения обновляемся. Спрашивается, почему нельзя передать `mLifeGrid` и `mAdapter` в класс-обработчик таймера и обновить их там? Ответ - таймер работает в другом потоке, а андроид разрешает модифицировать элементы управления только в том потоке, в котором они были созданы.

Теперь, запустив Life, можно увидеть, например, следующее



Ссылки

- Android documentation
- The Timer - Threading/Drawing on Canvas

Заключение

Итак, мы написали первое приложение для Android, которое уже и не совсем "Hello, World". Лично мне писать для Android понравилось куда больше, чем классические мидлеты. Остался, правда, ряд претензий к Eclipse, но, возможно, это от недостатка опыта.

Спасибо, если кто осилил. Замечания приветствуются.

Исходники примера