Компоненты приложения Android

Платформа приложений Android предоставляет разработчикам ряд компонентов, которые можно задействовать для создания пользовательского интерфейса приложений в целом и приложения Tetris в частности. *Компонент Android* — это многократно используемый шаблон программы или объекта, с помощью которого можно определить те или иные возможности приложения.

Некоторые важные компоненты, предоставляемые платформой приложений Android, приведены в следующем списке:

- действия (Activity);
- намерения (Intents);
- фильтры намерений (Intent filters);
- фрагменты (Fragments);
- службы (Services);
- загрузчики (Loaders);
- провайдеры контента (Content providers).

Действия

Действие — это компонент Android, который играет центральную роль в реализации потока приложения и взаимодействия между его компонентами. Действия реализуются в форме классов. Экземпляр действия используется системой Android для инициации кода.

Действие важно при создании пользовательских интерфейсов приложений. Оно предоставляет окно, в котором рисуются элементы пользовательского интерфейса. Проще говоря, с помощью действий создаются экраны приложений.

Намерения

Намерение облегчает взаимосвязь между действиями. По сути, намерения играют в приложении Android роль мессенджеров. Они служат объектами обмена сообщениями, посылаемыми для запроса действий от компонентов приложения. Намерения могут использоваться для выполнения таких действий, как запрос начала действия и передача в среде Android широковещательных сообщений.

Имеется два следующих типа намерений:

- *неявные намерения* — это объекты-мессенджеры, которые конкретно не определяют компонент приложения для выполнения действия, но указывают действие, которое должно выполняться, и позволяют компоненту, который может находиться в другом приложении, выполнить действие. Компоненты, которые могут неявно обрабатывать запрашиваемое действие, определяются системой Android;
- *явные намерения* — эти намерения явно указывают компонент приложения, который должен выполнить действие. Их можно применять для выполнения действий в пользовательском приложении — таких, например, как запуск действия

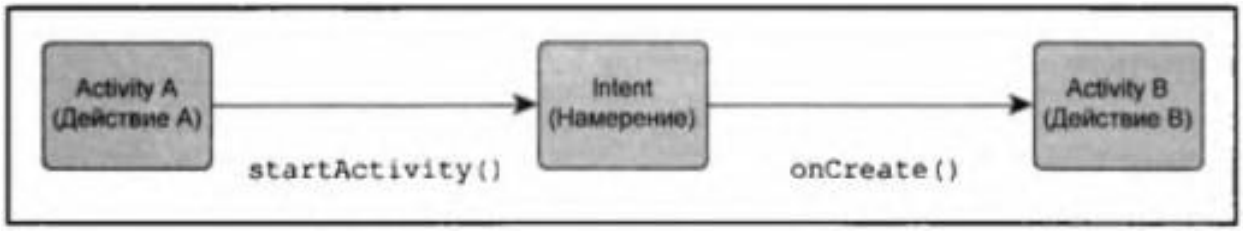

Рис. 1. Схема применения явного намерения

Фильтры намерений

Фильтр намерений — это объявление в файле манифеста приложения, где указывается тип намерения, которое хотел бы получить компонент. Это полезно для ряда случаев — таких, например, как выполняемый сценарий, когда желательно, чтобы действие в одном приложении выполняло определенное действие, запрошенное компонентами другого приложения. Тогда фильтр намерений может быть объявлен в манифесте приложения для действия, которое желательно обработать с помощью внешнего запроса. Если вы не желаете, чтобы действие обрабатывало неявные намерения, то не объявляете для него фильтр намерений.

Фрагменты

Фрагмент — это прикладной компонент, представляющий часть пользовательского интерфейса, существующего в действии. Подобно действию, фрагмент имеет макет, который можно изменять и который отображается в окне действия.

Службы

В отличие от большинства других компонентов, *служба* не предоставляет пользовательского интерфейса. Службы применяются для выполнения в приложении фоновых процессов. Службе не требуется приложение, которое ее создало, чтобы быть на переднем плане для запуска на выполнение.

Загрузчики

Загрузчик — это компонент, который позволяет загружать данные для последующего отображения их в действии или во фрагменте из источника данных, такого как провайдер контента.

Провайдеры контента

Эти компоненты помогают приложению контролировать доступ к ресурсам данных, сохраняющимся в том или ином приложении. Кроме того, *провайдер контента* облегчает обмен данными с другим приложением через открытый интерфейс программирования приложений (рис. 2.2).

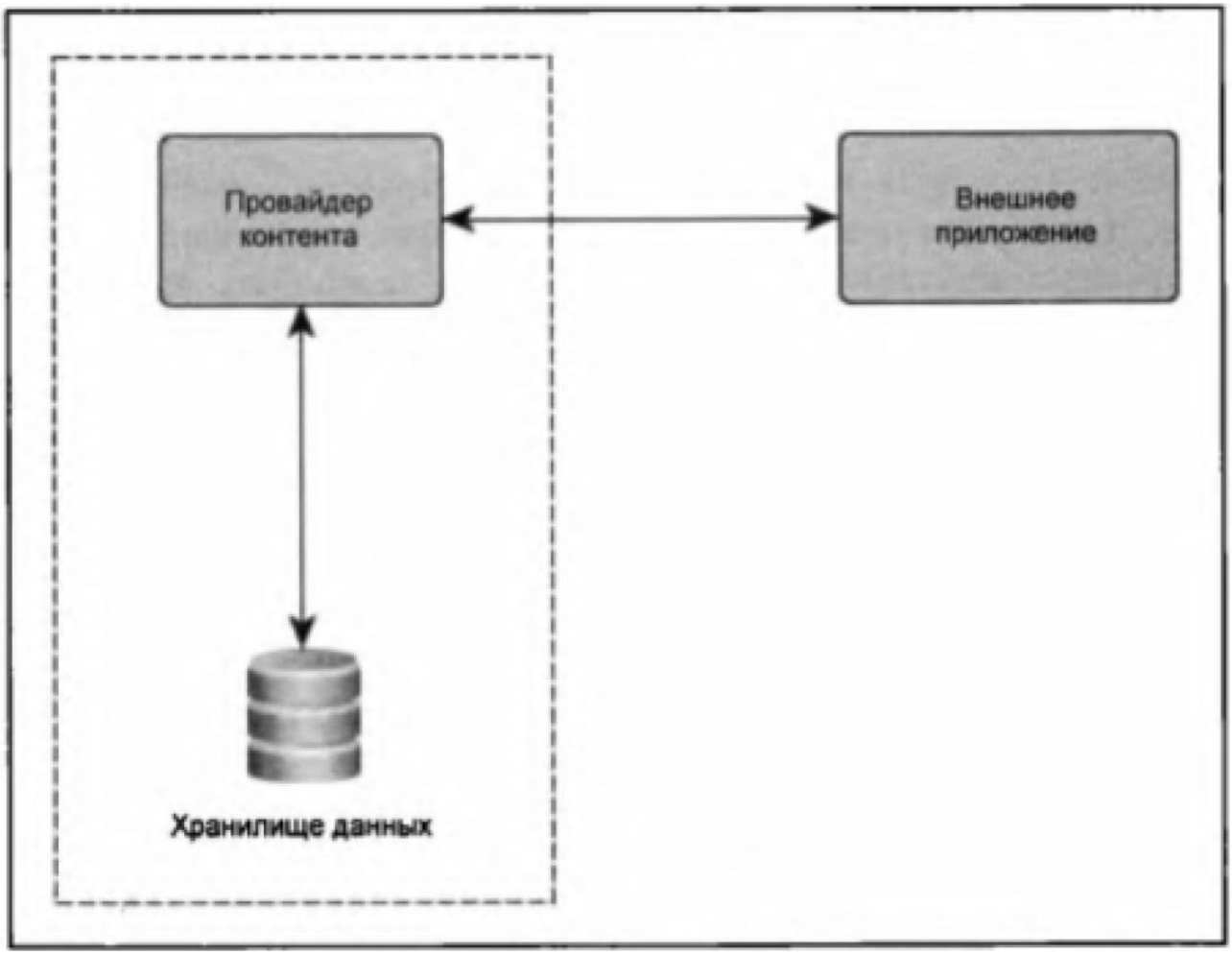

Рис. 2. Схема взаимодействия провайдера контента с внешним приложением

Представления (Views)

Представление — элемент макета, который занимает заданную область экрана и отвечает за отрисовку и обработку событий. Представление является базовым классом для элементов пользовательского интерфейса, или виджетов, таких как текстовые поля, поля ввода и кнопки. Все представления расширяют класс `View`.

Представления могут формироваться в макете XML в исходном файле:

```
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Roll the dice!"/>
```

Помимо создания представлений непосредственно в файле макета, также можно создавать их программно в файлах программы. Например, текстовое представление можно сформировать путем создания экземпляра класса `TextView` и передачи контекста его конструктору. Этот подход продемонстрирован в следующем фрагменте

Группы представлений (View groups)

Группа представлений — это особое представление, которое может включать в себя представления. Группу представлений, содержащую одно или несколько представлений, обычно называют *родительским представлением*, и представления

включаются в нее как ее дочерние представления. Группа представлений является родительским классом для некоторых других контейнеров представлений. Примерами групп представлений могут служить: `LinearLayout`, `CoordinatorLayout`, `ConstriantLayout`, `RelativeLayout`, `AbsoluteLayout`, `GridLayout` и `FrameLayout`.

Группы представлений могут формироваться в макете XML в исходном файле:

Подобно представлениям, группы представлений могут также формироваться программно в классах компонентов. В следующем фрагменте кода создается линейный

Tetris — правила игры

Прежде чем приступить к разработке приложения Tetris для Android, познакомимся с правилами этой игры.

Tetris («тетрис») — это игра-головоломка, основными компонентами которой являются цветные квадратные блоки-плитки. Название «тетрис» происходит от слов «тетра» — греческий числовой префикс для четырех предметов — и «теннис». Плитки в тетрисе формируют так называемые *тетрамино́*, представляющие собой геометрические фигуры, состоящие из четырех соединенных сторонами плиток (рис. 2.3).

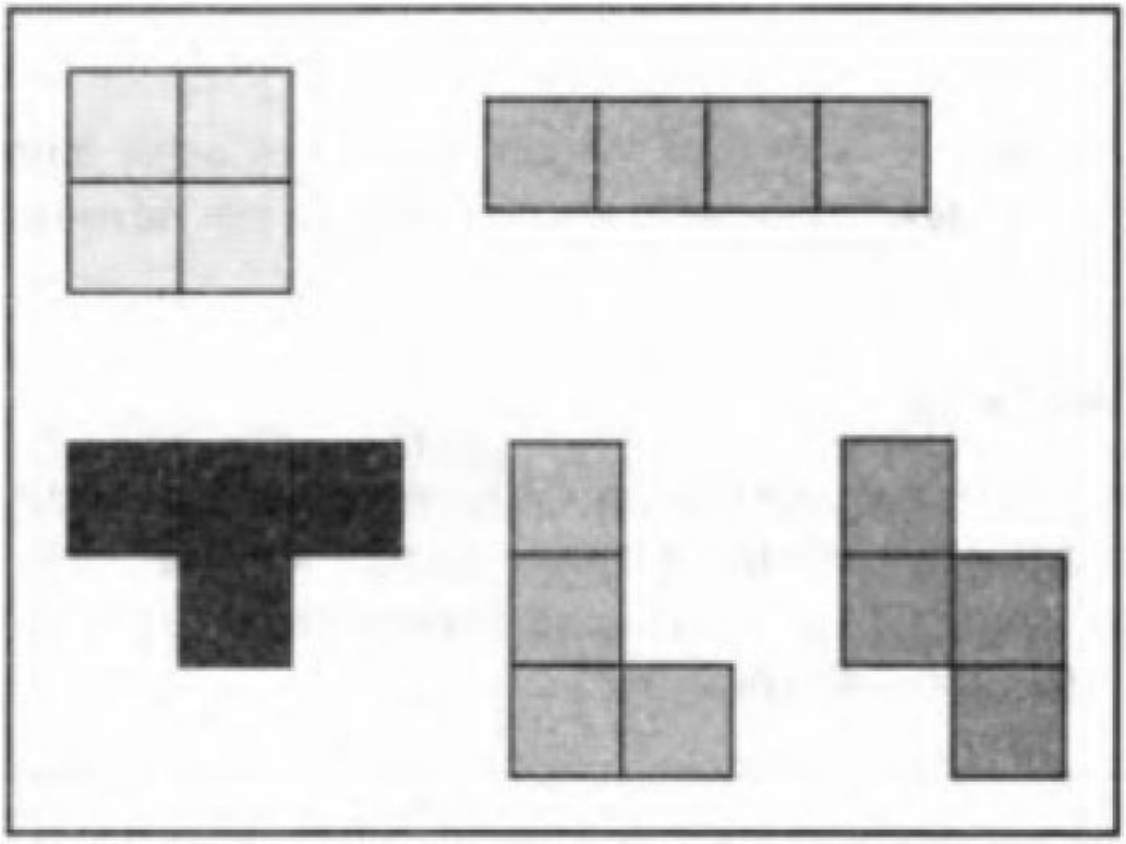

Рис. 3. Варианты тетрамино игры тетрис

Во время игры случайная последовательность тетрамино, имеющих различную ориентацию, падает на игровое поле. Игрок управляет перемещениями тетрамино. Каждое тетрамино способно по указаниям игрока выполнять ряд движений: их можно перемещать влево, вправо и вращать. Кроме того, скорость падения каждого тетрамино может быть увеличена. Цель игры — сформировать из опускающихся тетрамино непрерывную горизонтальную линию, состоящую из десяти квадратных плиток. Если подобная линия сформирована, она убирается с экрана.

Теперь, имея представление о функционировании игры Tetris, разберемся со спецификой формирования пользовательского интерфейса приложения.

Создание интерфейса пользователя

Как упоминалось ранее, пользовательский интерфейс служит средством, с помощью которого пользователь взаимодействует с приложением. Важность пользовательского интерфейса трудно переоценить. Прежде чем приступить к процессу программирования пользовательского интерфейса, полезно создать графическое представление о реализуемом пользовательском интерфейсе. Это может быть сде-

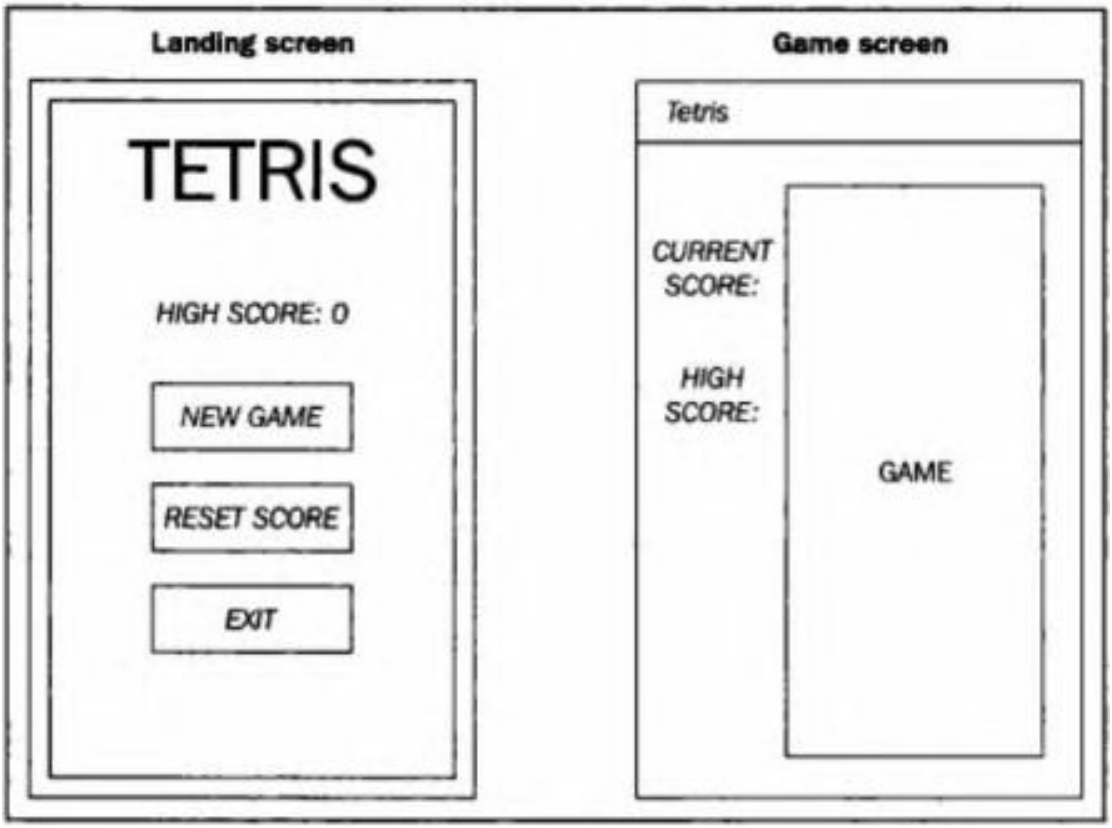

Рис. 4. Эскиз пользовательского интерфейса

Реализация макета

В каталоге res\values создайте новый файл ресурсов измерений dimens.

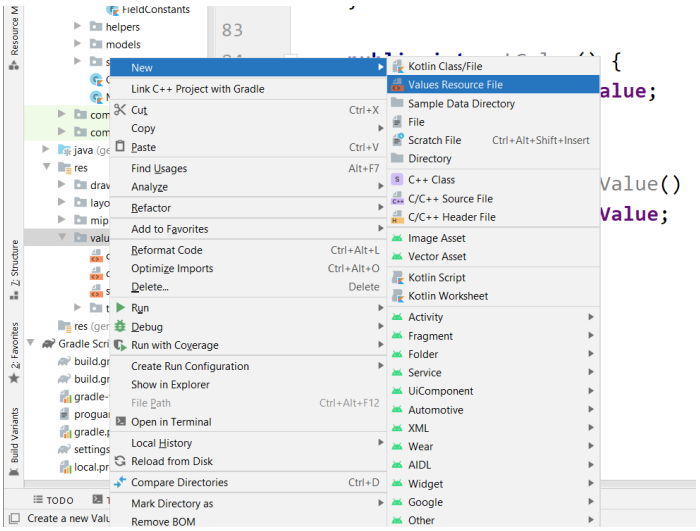

Рис. 5. Создание нового файла ресурсов измерений

Содержимое файла

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="layout_margin_top">16dp</dimen>
    <dimen name="layout_margin_bottom">16dp</dimen>
    <dimen name="layout_margin_start">16dp</dimen>
    <dimen name="layout_margin_end">16dp</dimen>
    <dimen name="layout_margin_vertical">16dp</dimen>
</resources>
```

Файл разметки главного окна приложения:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/background"
    android:orientation="vertical"
    tools:context=".MainActivity"
    android:gravity="center">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/tetris"
        android:textSize="80sp"/>

    <TextView
        android:id="@+id/tv_high_score"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/high_score"
```

```
        android:textSize="20sp"
        android:layout_marginTop="@dimen/layout_margin_top"/>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="3"
        android:orientation="vertical"
        android:gravity="center">

        <Button
            android:id="@+id/btn_new_game"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/new_game" />

        <Button
            android:id="@+id/btn_reset_score"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/reset_score"/>

        <Button
            android:id="@+id/btn_exit"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/exit"/>
    </LinearLayout>

</LinearLayout>
```

Ресурсы строк

```
<resources>
    <string name="app_name">Tetris</string>
    <string name="new_game">New game</string>
    <string name="high_score">High score</string>
    <string name="current_score">Current score</string>
    <string name="tetris">Tetris</string>
    <string name="reset_score">Reset score</string>
    <string name="exit">Exit</string>
    <string name="btn_restart">Restart</string>
    <string name="zero">0</string>
</resources>
```

Обработка событий ввода данных

В цикле взаимодействия пользователя с приложением имеется средство, поддерживающее для него некоторую форму ввода данных. Этим средством является взаимодействие с виджетом. Ввод данных может перехватываться с помощью событий. В приложениях Android события перехватываются из определенного объекта представления, с которым взаимодействует пользователь. Необходимые для обработки входных событий структуры и процедуры предоставляются классом `View`.

Слушатели событий

Слушатель событий — это процедура в приложении, которая ожидает события пользовательского интерфейса. Типов событий, которые могут генерироваться в приложении, достаточно много. Вот примеры некоторых общих событий: события щелчка, события касания, события длинного щелчка и события изменения текста.

Для захвата события виджета и выполнения действия при его возникновении слушатель событий должен быть в представлении установлен. Это может достигаться путем вызова набора представлений. Для такого вызова применяется метод `Listener()` с передачей либо лямбда-выражения, либо ссылки на функцию.

В следующем примере демонстрируется захват события щелчка на кнопке. Лямбда-выражение при этом передается методу `setOnClickListener` класса представления:

Содержимое файла кодового файла MainActivity

```
package com.example.tetris

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.Button
import android.widget.TextView
import com.example.tetris.storage.AppPreferences
import com.google.android.material.snackbar.Snackbar

class MainActivity : AppCompatActivity() {

    var tvHighScore: TextView? = null
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        // убрать строку с заголовком
        supportActionBar?.hide()
        val btnNewGame = findViewById<Button>(R.id.btn_new_game)
        val btnResetScore = findViewById<Button>(R.id.btn_reset_score)
        val btnExit = findViewById<Button>(R.id.btn_exit)
        tvHighScore = findViewById(R.id.tv_high_score)
        btnNewGame.setOnClickListener(this::onBtnNewGameClick)
        btnResetScore.setOnClickListener(this::onBtnResetScoreClick)
```

```
        btnExit.setOnClickListener(this::onBtnExitClick)
    }

    private fun onBtnNewGameClick(view: View) {
        val intent = Intent(this, GameActivity::class.java)
        startActivity(intent)
    }

    private fun onBtnResetScoreClick(view: View) {
        val preferences = AppPreferences(this)
        preferences.clearHighScore()
        Snackbar.make(view, "Score successfully reset",
Snackbar.LENGTH_SHORT).show()
    }

    private fun onBtnExitClick(view: View) {
        System.exit(0)
    }
}
```

Использование интерфейса *SharedPreferences*

Интерфейс `SharedPreferences` обеспечивает хранение, организацию доступа и модификацию данных, а также сохранение данных в наборах пар ключ-значение.

Настроим простую утилиту для обработки потребности в хранении данных для нашего приложения, применив интерфейс `SharedPreferences`. Создайте в исходном каталоге проекта пакет с именем storage — щелкните правой кнопкой мыши на исходном каталоге и выберите команды **New | Package** (Создать | Пакет) (рис. 2.15).

Затем внутри пакета storage создайте новый класс Kotlin под названием `AppPreferences`. Введите в файл класса следующий код:

```
package com.example.tetris.storage

import android.content.Context
import android.content.SharedPreferences

class AppPreferences(ctx: Context) {
    var data: SharedPreferences =
ctx.getSharedPreferences("APP_PREFERENCES", Context.MODE_PRIVATE)

    fun saveHighScore(highScore: Int){
        data.edit().putInt("HIGH_SCORE", highScore).apply()
    }

    fun getHighScore(): Int {
        return data.getInt("HIGH_SCORE", 0)
    }

    fun clearHighScore() {
        data.edit().putInt("HIGH_SCORE", 0).apply()
```

```
    }
}
```

В приведенном фрагменте кода класс `Context` необходим для передачи конструктора класса при создании экземпляра класса. Класс `Context` поддерживает доступ к методу `getSharedPreferences()`, который получает файл настроек. Файл настроек идентифицируется по имени в строке, переданной как первый аргумент метода `getSharedPreferences()`.

Функция `saveHighScore()` принимает целое число — наибольшее количество очков, которое будет сохраняться, когда единственный аргумент `data.edit()` возвратит объект `Editor`, разрешающий изменение файла настроек. Метод объекта `Editor`, `putInt()`, вызывается для сохранения целого числа в файле настроек. Первый аргумент, переданный `putInt()`, служит строкой, представляющей ключ, который будет использоваться для доступа к сохраненному значению. Вторым аргументом метода служит целое число, которое необходимо сохранить, — в нашем случае наибольшее количество очков.

Функция `getHighScore()` возвращает наибольшее количество очков при вызове данных с помощью функции `getInt()`. Функция `getInt()` реализуется с помощью интерфейса `SharedPreferences` и поддерживает доступ (для чтения) к сохраненному целочисленному значению. Идентификатор `HIGH_SCORE` является уникальным идентификатором значения, которое необходимо получить. Значение `0`, передаваемое второму аргументу функции, указывает заданное по умолчанию значение, которое должно возвращаться сценарием, если в сценарии отсутствует соответствующее указанному ключу значение.

Функция `clearHighScore()` сбрасывает значение наибольшего количества очков до нуля, просто перезаписывая нулем значение, соответствующее ключу `HIGH_SCORE`.

Реализация макета игрового действия (GameActivity)

Файл разметки:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".GameActivity"
    android:orientation="horizontal"
    android:weightSum="10"
    android:background="#e8e8e8">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:gravity="left"
        android:paddingTop="32dp"
        android:paddingBottom="32dp"
        android:layout_weight="1"
```

```
        tools:ignore="RtlHardcoded">
        <LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="0dp"
            android:layout_weight="1"
            android:orientation="vertical"
            android:gravity="center"
            tools:ignore="NestedWeights">
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/current_score"
                android:textAllCaps="true"
                android:textStyle="bold"
                android:textSize="14sp"/>
            <TextView
                android:id="@+id/tv_current_score"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/zero"
                android:textSize="18sp"/>
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginTop="@dimen/layout_margin_top"
                android:text="@string/high_score"
                android:textAllCaps="true"
                android:textStyle="bold"
                android:textSize="14sp"/>
            <TextView
                android:id="@+id/tv_high_score"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/zero"
                android:textSize="18sp"/>
        </LinearLayout>
        <Button
            android:id="@+id/btn_restart"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/btn_restart"/>
    </LinearLayout>
    <View
        android:layout_width="1dp"
        android:layout_height="match_parent"
        android:background="@color/black"/>
</LinearLayout>
```

Кодовый файл:

```
package com.example.tetris
```

```kotlin
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import android.widget.TextView
import com.example.tetris.storage.AppPreferences

class GameActivity : AppCompatActivity() {

    var tvHighScore: TextView? = null
    var tvCurrentScore: TextView? = null
    var appPreferences: AppPreferences? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_game)
        appPreferences = AppPreferences(this)

        val btnRestart = findViewById<Button>(R.id.btn_restart)
        tvHighScore = findViewById<TextView>(R.id.tv_high_score)
        tvCurrentScore = findViewById<TextView>(R.id.tv_current_score)
    }

    private fun updateHighScore() {
        tvHighScore?.text = "${appPreferences?.getHighScore()}"
    }

    private fun updateCurrentScore() {
        tvCurrentScore?.text = "0"
    }
}
```

Реализация логики и функциональности игры Тетрис

Характеристики блока

Рассмотрим некоторые характеристики, присущие блокам, составляющим тетрамино:

- *форма* — блок имеет фиксированную форму, которую изменять нельзя;
- *размерности* — блок обладает характеристиками размерности, за которые мы примем их высоту и ширину;
- *цвет* — блок всегда окрашен. Цвет блока фиксирован и поддерживается все время его существования;
- *пространственная характеристика* — блок занимает фиксированное пространство;
- *позиционная характеристика* — в любой момент времени блок располагает позицией, которая существует вдоль двух осей: X и Y.

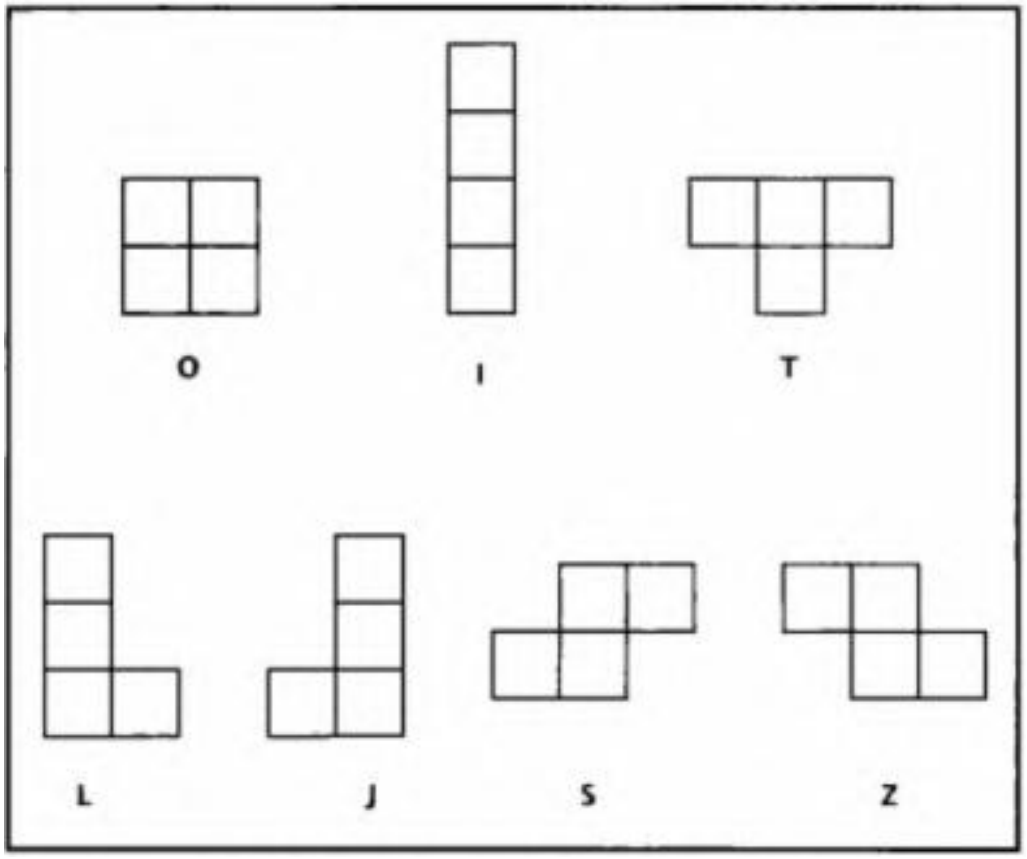

Рис. 6. Семь основных блоков тетрамино

Все показанные на рис. 3.2 формы блоков тетрамино занимают пространство в пределах своих границ. Область пространства, занятая формой, может отображаться как контур, или фрейм. Это схоже с размещением изображения в кадре. Следует смоделировать этот фрейм, который и будет представлять собой отдельную форму. Поскольку отдельные плитки, составляющие тетрамино и входящие во фрейм, представляют собой двумерные объекты, для хранения информации, специфичной для фрейма, применим двумерный байтовый массив. *Байт* — это цифровая единица информации, состоящая из восьми битов. *Бит* — это двоичная цифра, наименьшая единица данных в компьютере, которая принимает значение 1 или 0.

Идея состоит в моделировании формы фрейма с помощью двумерного массива путем представления занятых плитками областей во фрейме значением байта, равном 1, а тех, что не заняты плитками, — значением 0. Рассмотрим, например, следующий фрейм (рис. 3.3, *а*).

Рис. 7. Пример фрейма и визуализация его в виде двумерного массива байтов