

Игра тетрис

Создание нового проекта

Create New Project

Select a Project Template


Phone and Tablet

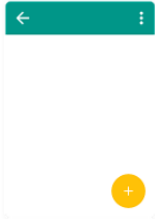
Wear OS


Android TV

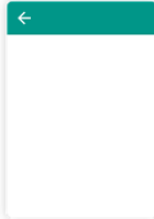
Automotive


Android Things


No Activity


Basic Activity


Bottom Navigation Activity


Empty Activity


Empty Activity
Creates a new empty activity

Previous

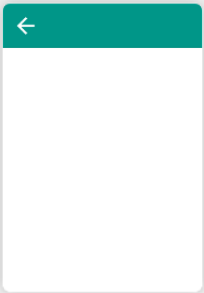
Next

Cancel

Finish

Create New Project

Configure Your Project


Empty Activity
Creates a new empty activity

Name

tetris

Package name

com.example.tetris

Save location

C:\Users\Admin\AndroidStudioProjects\tetris

Language

Kotlin

Minimum SDK

API 16: Android 4.1 (Jelly Bean)

Your app will run on approximately 99,8% of devices.

Help me choose

☐ Use legacy android.support libraries

The application name for most apps begins with an uppercase letter

Previous

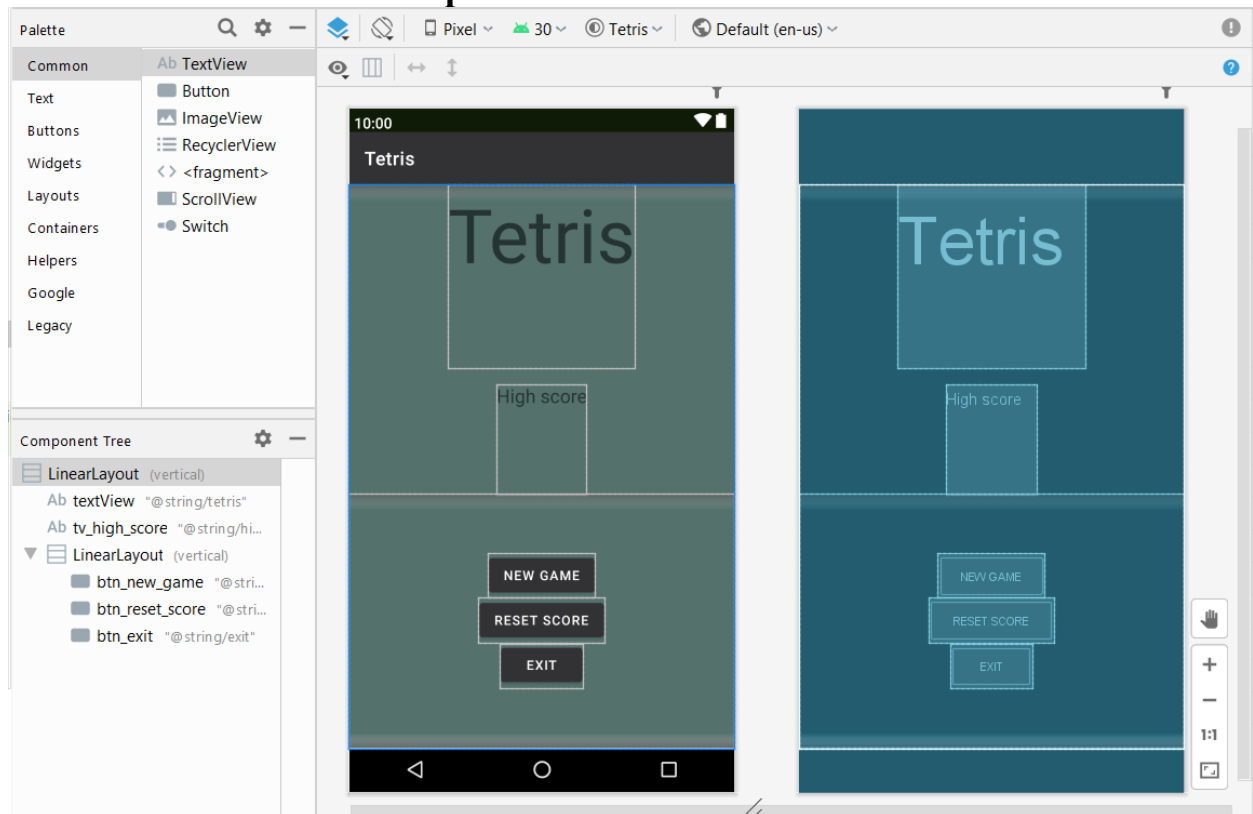
Next

Cancel

Finish

Создание интерфейса пользователя

Использование линейной разметки



Определение свойств элементов

```
<?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="#54716B"
    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="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="High score"
        android:textSize="20sp"
        android:layout_marginTop="16dp"/>
```

```

<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="New game"/>

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

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

</LinearLayout>

```

Все константы следует вынести в файлы ресурсов strings.xml

```

<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="reset_score">Reset score</string>
    <string name="exit">Exit</string>
</resources>

```

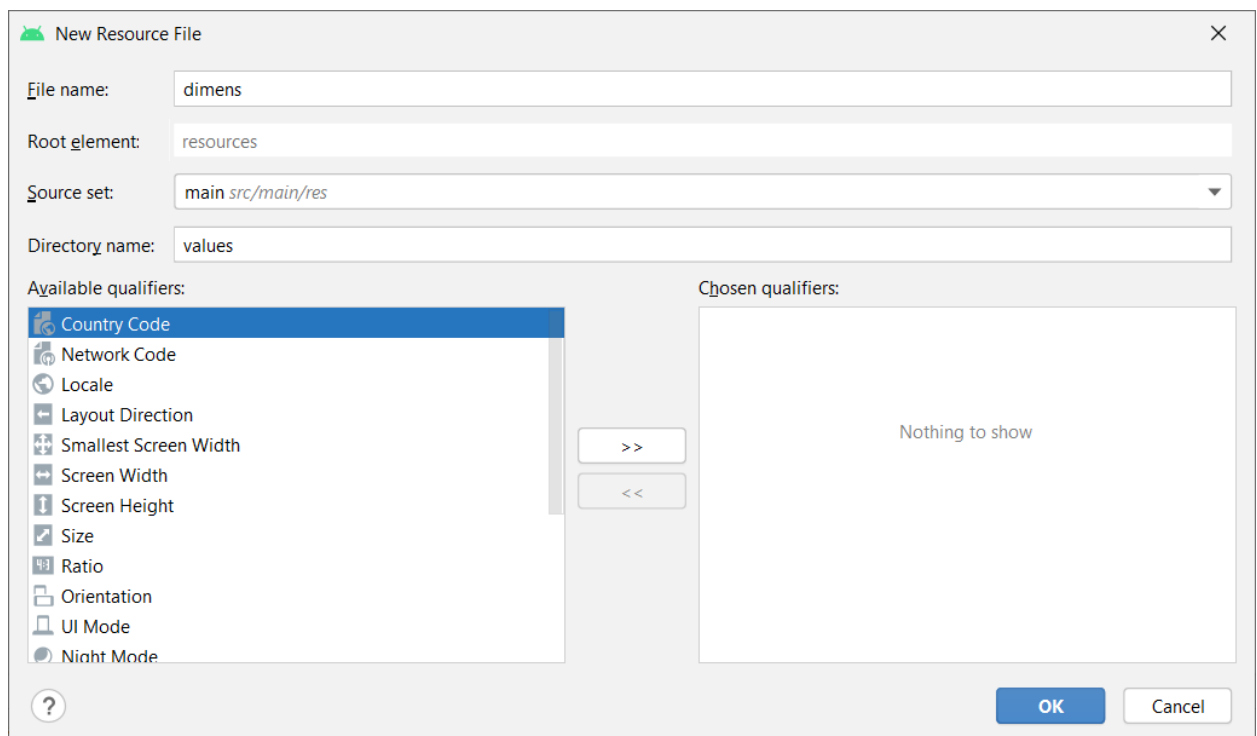
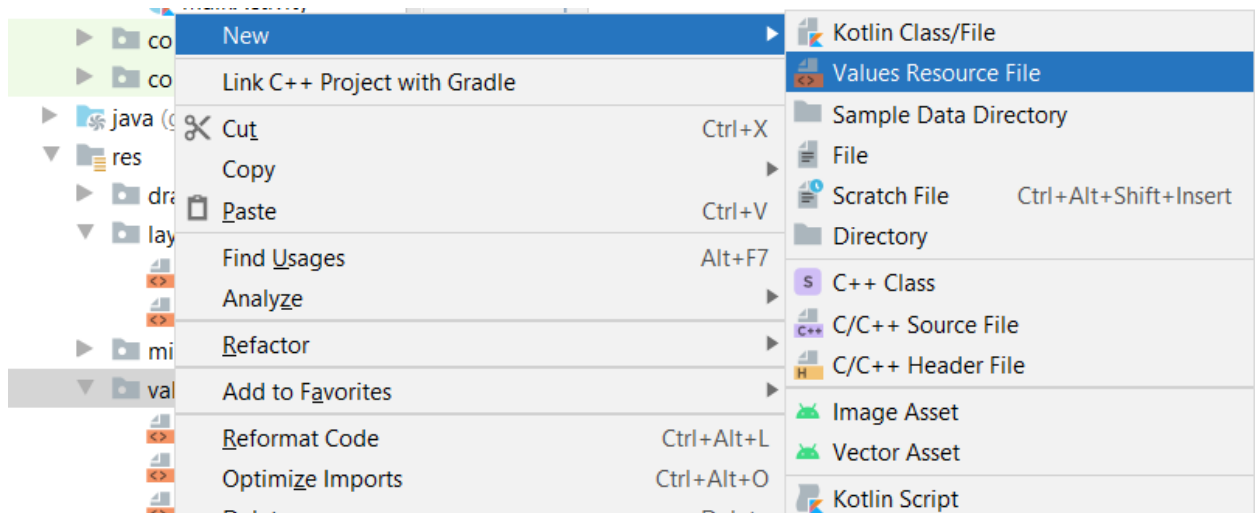
В файл colors.xml должен быть добавлен цвет фона, например:

```

<color name="background">#54716B</color>

```

В папку res\values добавить файл ресурсов dimens (ресурсы измерений):



Введите содержимое:

```
<?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>
```

Введите изменения в файл activity_main.xml

Обработка событий ввода данных

Ввод данных может перехватываться с помощью событий. Слушатель событий — это процедура в приложении, которая ожидает события пользовательского интерфейса. Примеры некоторых событий: щелчок мышью,

двойной щелчок, изменение текста. Для захвата события виджета и выполнения действия при его возникновении слушатель событий должен быть установлен в представлении. Для такого вызова применяется метод `Listener()` с передачей либо лямбда-выражения, либо ссылки на функцию.

Реализация логики в кодовом файле `MainActivity.kt`.

Скрыть панель приложения (`supportActionBar`). Сформировать ссылки на объекты для виджетов, которые есть в макете. Ссылки на объекты представления можно получить, передавая идентификатор ресурса представления методу `findViewById()`.

```
class MainActivity : AppCompatActivity() {  
  
    var tvHighScore: TextView? = null  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
        // Скрыть панель приложения  
        supportActionBar?.hide()  
        val btnNewGame = <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)  
    }  
}
```

Добавьте обработчики событий. Установите слушатели событий щелчков мыши для всех кнопок макета.

Кнопка NEW GAME (Новая игра) предназначена для перехода пользователя к игровому действию. Так как игровое действие будет реализовано на втором макете, поэтому пока нужно создать только заготовку для этого события.

Функция `onBtnResetScoreClick` для щелчка мышью на кнопке RESET SCORE (Сбросить счет) также будет реализована позже.

```
class MainActivity : AppCompatActivity() {  
  
    var tvHighScore: TextView? = null  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
        // Скрыть панель приложения  
        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) {
}

private fun onBtnResetScoreClick(view: View) {
}

private fun onBtnExitClick(view: View) {
    exitProcess(0)
}
}

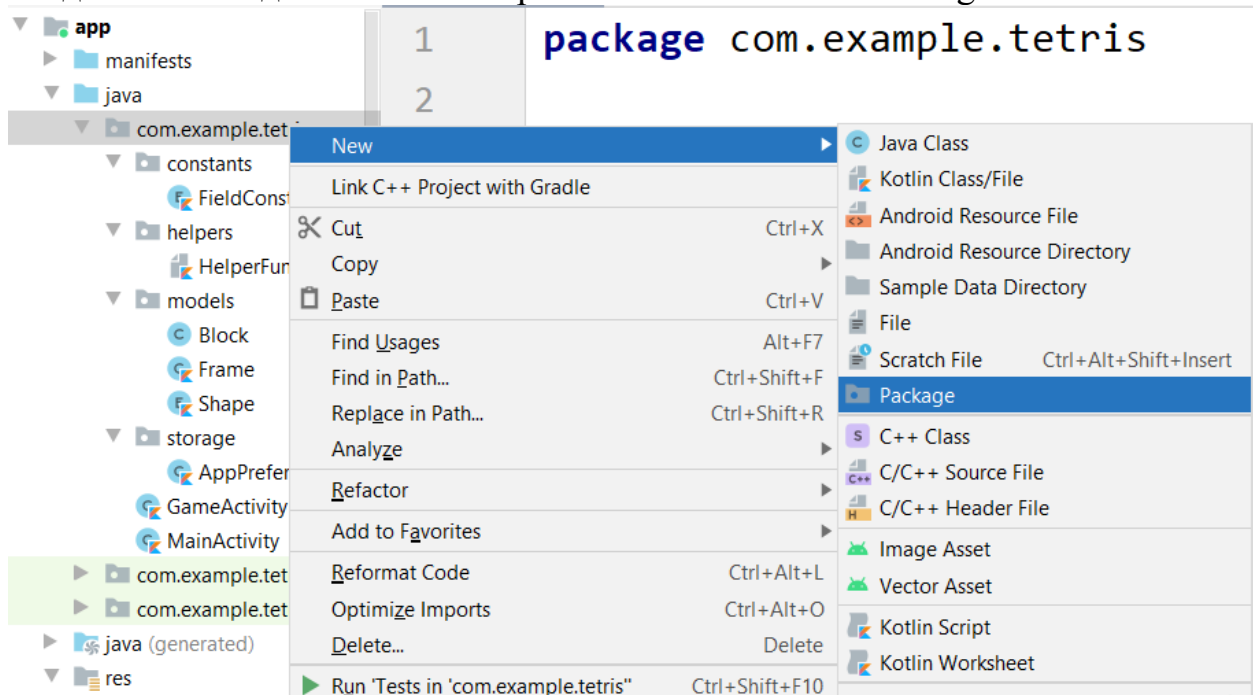
```

Переменная `intent` создает новый экземпляр класса `Intent` и передает конструктору текущий контекст и класс действия. Ключевое слово `this` используется для ссылки на текущий экземпляр, в котором он вызывается

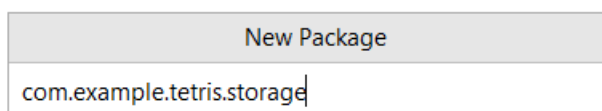
Реализация логики для сброса наибольшего количества очков

Сначала нужно реализовать логику для хранения данных. Для выполнения этой задачи воспользуемся интерфейсом `SharedPreferences`, который обеспечивает хранение, организацию доступа и модификацию данных в наборах пар ключ-значение.

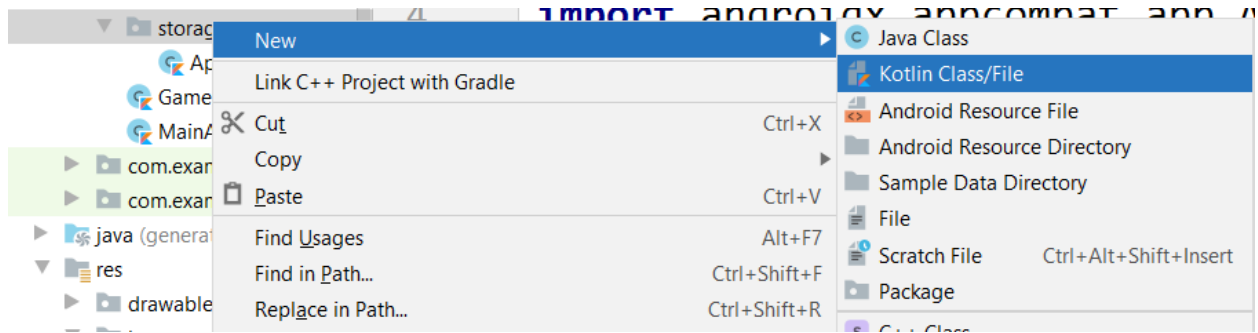
Создайте в исходном каталоге проекта пакет с именем `storage`:



Затем ввести имя пакета



Создайте внутри пакета новый класс `AppPreferences`



Введите в файл 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 (первый аргумент метода **"APP_PREFERENCES"**).

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

Функция `getHighScore()` возвращает наибольшее количество очков при вызове данных с помощью функции `data.getInt()`. Идентификатор **HIGH_SCORE** является уникальным идентификатором значения, которое

необходимо получить. Значение 0, передаваемое второму аргументу, является значением по умолчанию в случае отсутствия ключа в словаре.

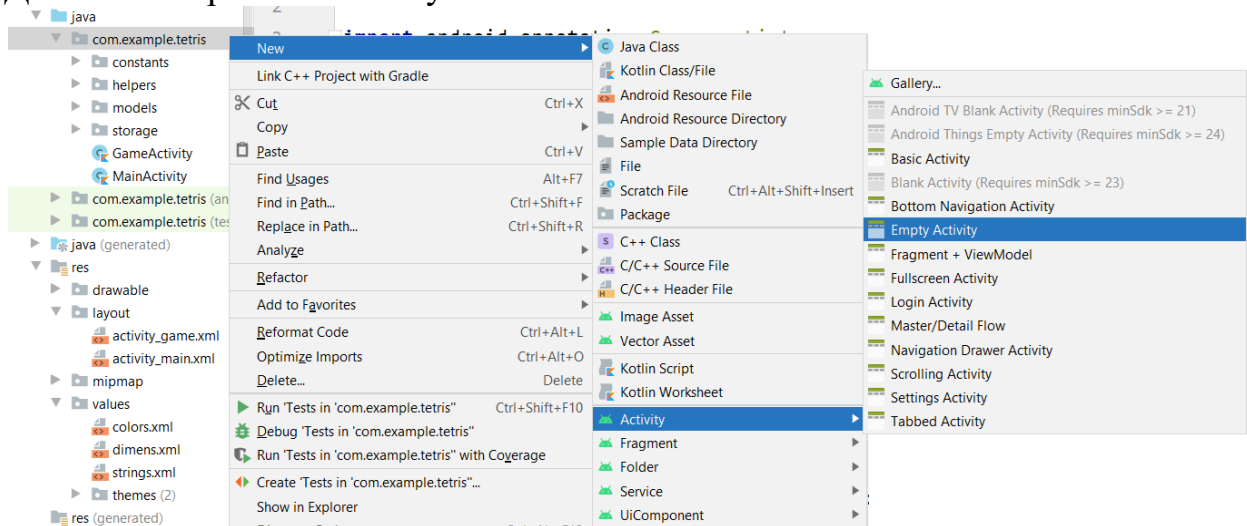
Функция `clearHighScore()` сбрасывает значение наибольшего количества очков до 0 для ключа `HIGH_SCORE`.

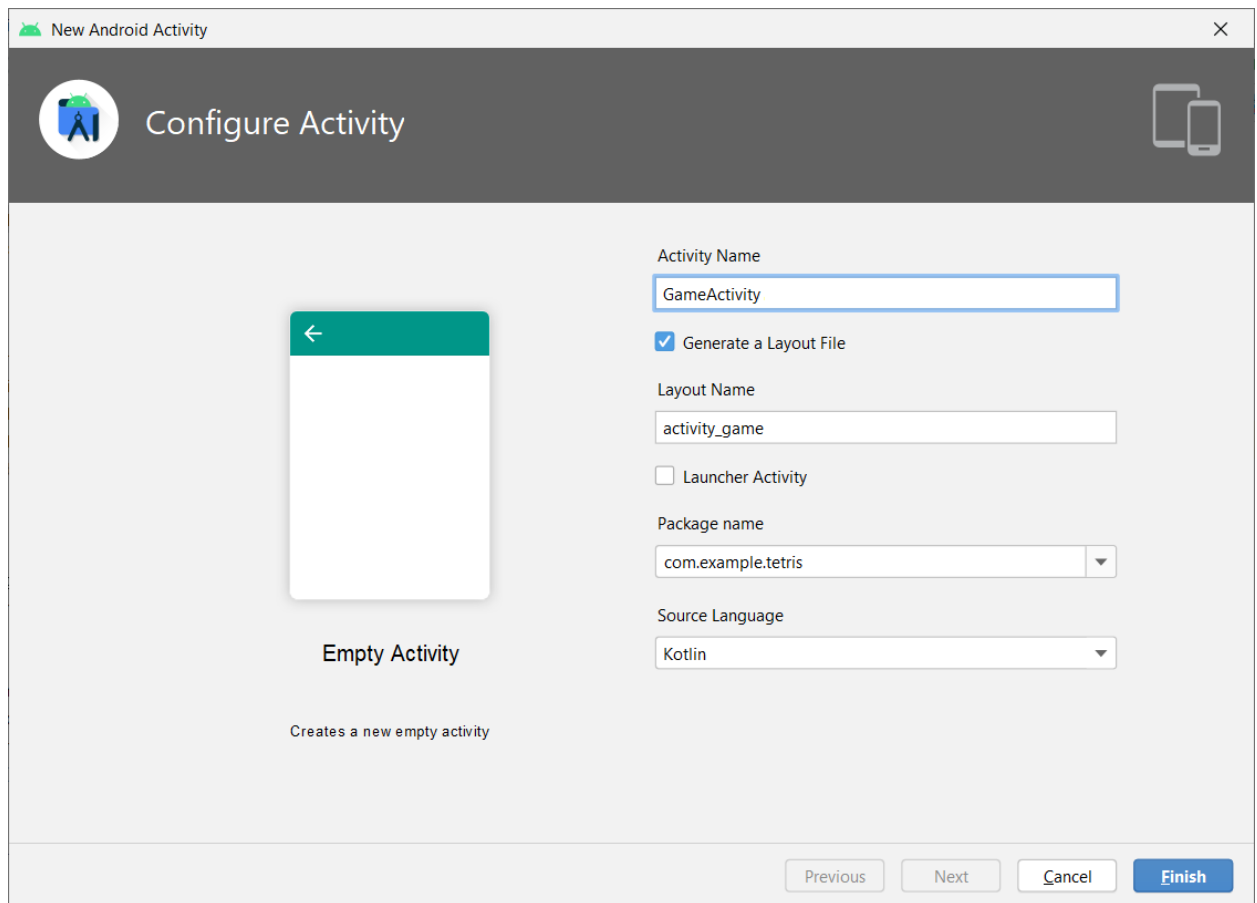
В `MainActivity.kt` нужно ввести код для функции:

```
private fun onBtnResetScoreClick(view: View) {  
    val preferences = AppPreferences(this)  
    preferences.clearHighScore()  
    // Вывести короткое сообщение для пользователя (обратная связь)  
    Snackbar.make(view, "Score successfully reset",  
        Snackbar.LENGTH_SHORT).show()  
    // Вывести текст на экран  
    tvHighScore?.text = "High score: ${preferences.getHighScore()}"  
}
```

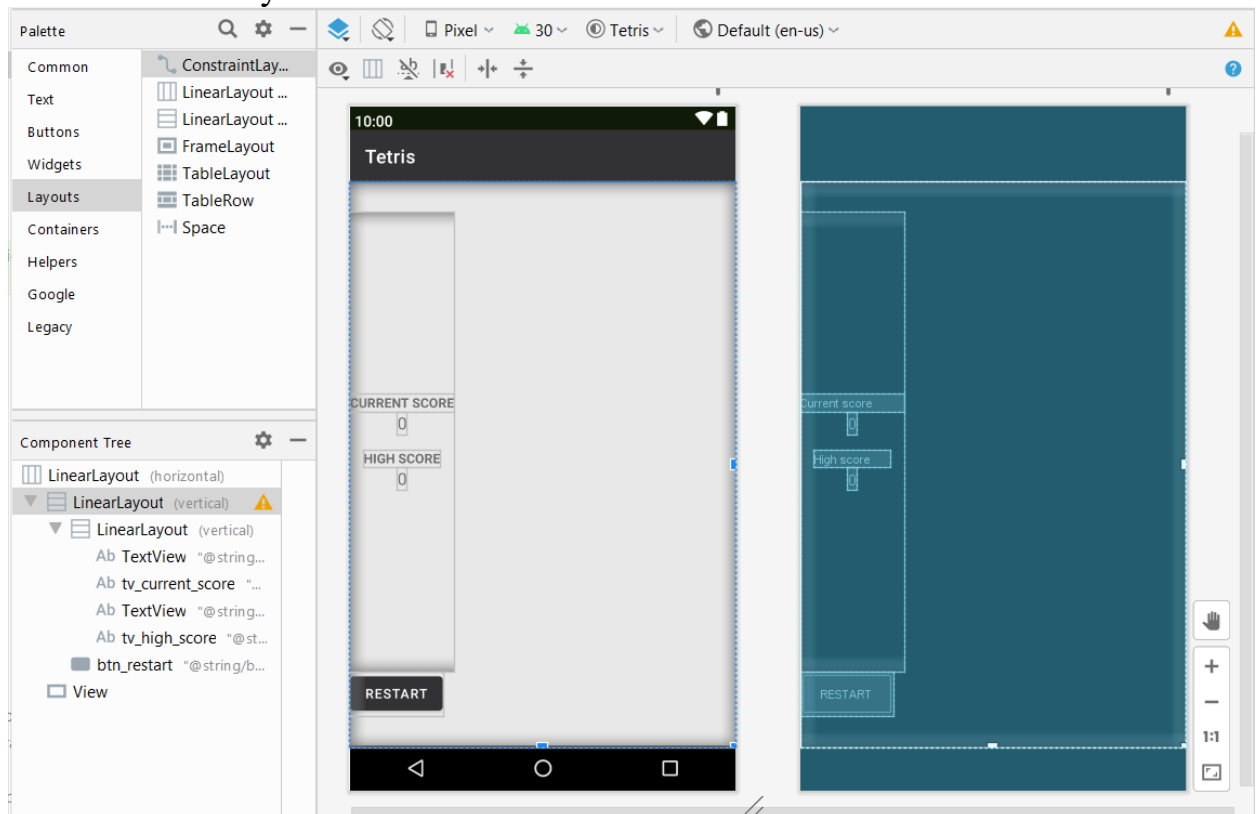
Реализация макета игрового действия

Добавьте в проект новое пустое активити.





Разметка Activity



Определение свойств элементов

```
<?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="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="0"
                android:textSize="18sp"/>

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginTop="@dimen/layout_margin_top"
                android:text="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="0"
            android:textSize="18sp"/>

    </LinearLayout>

    <Button
        android:id="@+id/btn_restart"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Restart"/>

</LinearLayout>

<View
    android:layout_width="1dp"
    android:layout_height="match_parent"
    android:background="#000000"/>

</LinearLayout>

```

Замените текст и цвет на константы в соответствующих файлах ресурсов.

Содержимое кодового файла `GameActivity.kt`

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

```

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)
    }
}

```

```

        updateHighScore()
        updateCurrentScore()
    }

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

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

```

Измените тело функции onBtnNewGameClick (MainActivity.kt):

```

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

```

Реализация логики и функциональности игры Tetris

Tetris – это игра-головоломка, которая основана на упорядочении падающих фигур – тетрамино, каждая из которых состоит из четырех соединенных различными сторонами цветных блоков-плиток.

Моделирование тетрамино

Характеристики блока:

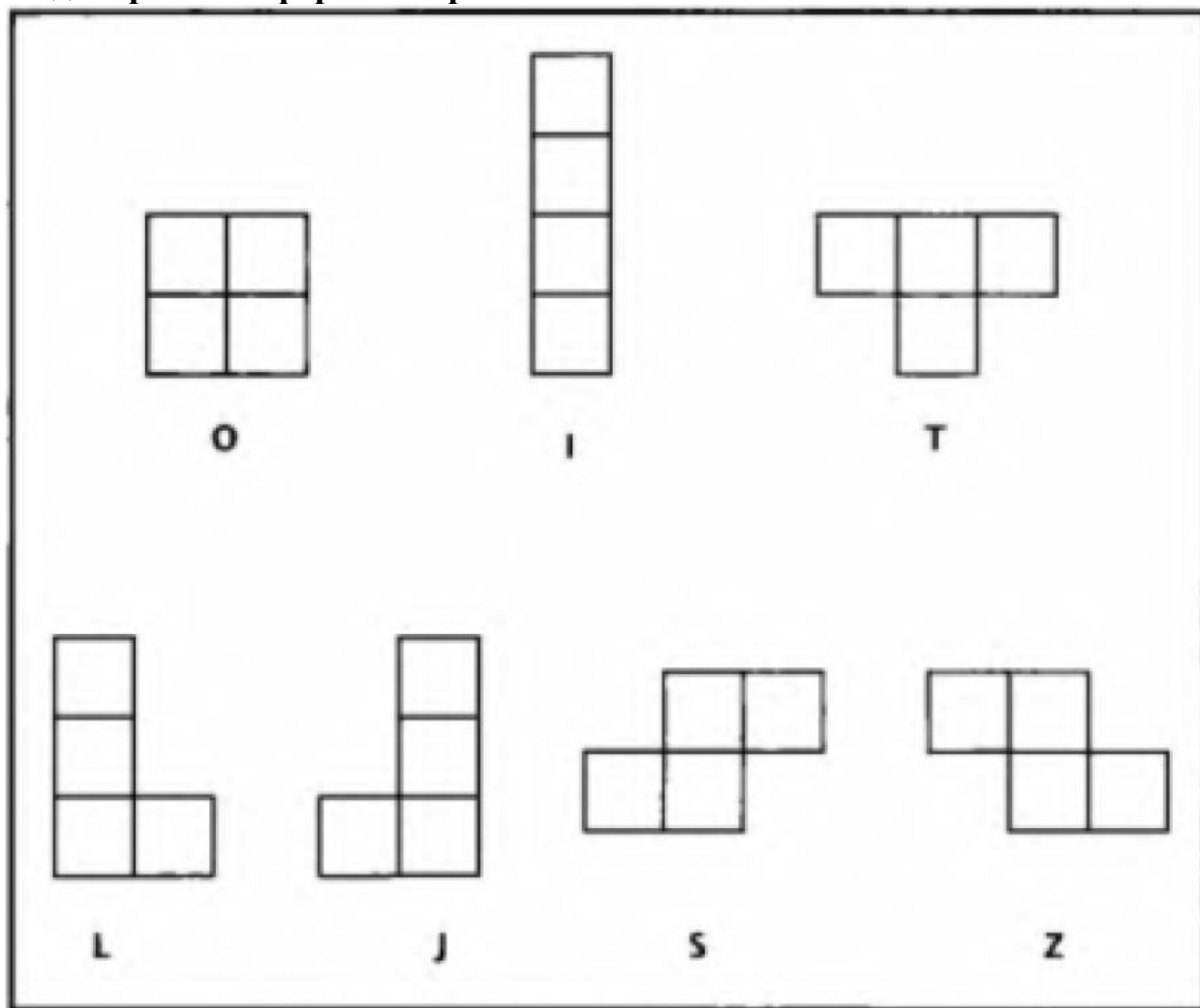
- форма – блок имеет фиксированную форму, которую нельзя менять;
- размерности – блок обладает высотой и шириной;
- цвет блока фиксирован и поддерживается все время его существования;
- пространственная характеристика – занимает фиксированное пространство;
- позиционная характеристика – в любой момент времени блок располагает позицией, которая существует вдоль осей X и Y.

Поведение блока

Основным поведением блока является его способность испытывать различные движения: поступательное и вращательное. Поступательное движение – это тип движения, когда тело перемещается из одной точки пространства в другую. Блок может двигаться влево, вправо и вниз.

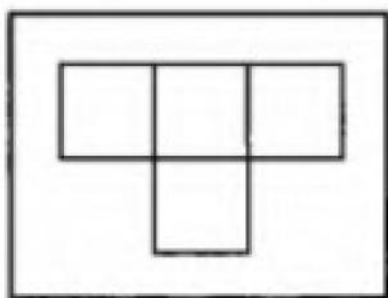
Вращательное движение – тип движения по криволинейной траектории, то есть вращение объекта в свободном пространстве.

Моделирование формы тетрамино

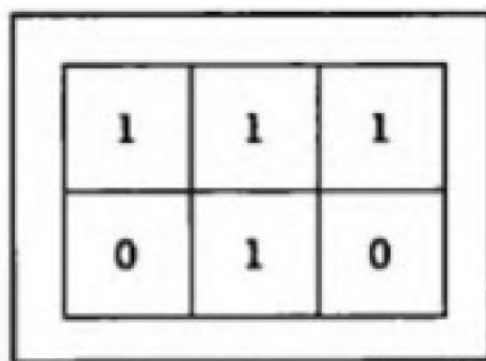


Семь основных блоков тетрамино

Поскольку отдельные плитки, составляющие терамино и входящие во фрейм, представляют собой двумерные объекты, для хранения информации, специфичной для фрейма применим двумерный массив. Байт – это цифровая единица информации, состоящая из 8 битов. Идея состоит в моделировании формы фрейма с помощью двумерного массива путем представления занятых плитками областей во фрейме значения байта, равном 1, а тех, что не заняты плитками, – значение 0.



а)



б)

Пример фрейма (а) и визуализация его в виде двумерного массива байтов (б)

Создайте в исходном проекте новый пакет с именем `helper`, а в нем файл `HelperFunctions.kt`. в нем будут содержаться все вспомогательные функции. Код файла:

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

```
fun array2dOfByte(sizeOuter: Int, sizeInner: Int): Array<ByteArray>
    = Array(sizeOuter){ ByteArray(sizeInner) }
```

Функция `array2dOfByte()` с двумя аргументами генерирует и возвращает новый массив с указанным числом строк и столбцов.

Создайте в исходном проекте новый пакет `models` и в нем файл `Frame.kt`. Введите код:

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

```
import com.example.tetris.helpers.array2dOfByte
```

```
class Frame(private val: Int) {
    val data: ArrayList<ByteArray> = ArrayList()

    fun addRow(byteStr: String): Frame {
        val row = ByteArray(byteStr.length)
        for (index in byteStr.indices) {
            row[index] = "${byteStr[index]}.toByte()"
        }
        data.add(row)
        return this
    }

    fun as2dByteArray(): Array<ByteArray> {
        val bytes = array2dOfByte(data.size, width)
        return data.toArray(bytes)
    }
}
```

Класс `Frame` содержит два свойства `width` и `data`. Свойство `width` задает необходимую ширину генерируемого фрейма (число столбцов). Свойство `data` содержит список элементов массива в пространстве значений `ByteArray`.

Функция обрабатывает строку, преобразуя каждый отдельный символ строки в байтовый массив, который добавляет в список данных.

Создайте в пакете `models` файл `Shape.kt`. В файле создайте класс `enum Shape`:

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

```
import java.lang.IllegalArgumentException
import java.text.FieldPosition
```

```

enum class Shape(val frameCount: Int, val startPosition: Int) {

    Tetromino1(1, 1) {
        override fun getFrame(frameNumber: Int): Frame {
            return Frame(2)
                .addRow("11")
                .addRow("11")
        }
    },

    Tetromino2(2, 1) {
        @Throws(IllegalArgumentException::class)
        override fun getFrame(frameNumber: Int): Frame {
            return when (frameNumber) {
                0 -> Frame(3)
                    .addRow("110")
                    .addRow("011")
                1 -> Frame(2)
                    .addRow("01")
                    .addRow("11")
                    .addRow("10")
                else -> throw IllegalArgumentException(
                    "$frameNumber is an invalid frame number."
                )
            }
        }
    },

    Tetromino3(2, 1) {
        override fun getFrame(frameNumber: Int): Frame {
            return when (frameNumber) {
                0 -> Frame(3)
                    .addRow("011")
                    .addRow("110")
                1 -> Frame(2)
                    .addRow("10")
                    .addRow("11")
                    .addRow("01")
                else -> throw IllegalArgumentException(
                    "$frameNumber is an invalid frame number."
                )
            }
        }
    },

    Tetromino4(2, 2) {
        override fun getFrame(frameNumber: Int): Frame {
            return when (frameNumber) {
                0 -> Frame(4).addRow("1111")
                1 -> Frame(1)
                    .addRow("1")
            }
        }
    }
}

```

```

        .addRow("1")
        .addRow("1")
        .addRow("1")
    else -> throw IllegalArgumentException(
        "$frameNumber is an invalid frame number."
    )
}
}
},

```

```

Tetromino5(4, 1) {
    override fun getFrame(frameNumber: Int): Frame {
        return when (frameNumber) {
            0 -> Frame(3)
                .addRow("010")
                .addRow("111")
            1 -> Frame(2)
                .addRow("10")
                .addRow("11")
                .addRow("10")
            2 -> Frame(3)
                .addRow("111")
                .addRow("010")
            3 -> Frame(2)
                .addRow("01")
                .addRow("11")
                .addRow("01")
            else -> throw IllegalArgumentException(
                "$frameNumber is an invalid frame number."
            )
        }
    }
}
},

```

```

Tetromino6(4, 1) {
    override fun getFrame(frameNumber: Int): Frame {
        return when (frameNumber) {
            0 -> Frame(3)
                .addRow("100")
                .addRow("111")
            1 -> Frame(2)
                .addRow("11")
                .addRow("10")
                .addRow("10")
            2 -> Frame(3)
                .addRow("111")
                .addRow("100")
            4 -> Frame(2)
                .addRow("01")
                .addRow("01")
                .addRow("11")
        }
    }
}

```



```

        else -> throw IllegalArgumentException(
            "$frameNumber is an invalid frame number."
        )
    }
}

Tetromino7(4, 1) {
    override fun getFrame(frameNumber: Int): Frame {
        return when (frameNumber) {
            0 -> Frame(3)
                .addRow("001")
                .addRow("111")
            1 -> Frame(2)
                .addRow("10")
                .addRow("10")
                .addRow("11")
            2 -> Frame(3)
                .addRow("111")
                .addRow("100")
            3 -> Frame(2)
                .addRow("10")
                .addRow("01")
                .addRow("01")
            else -> throw IllegalArgumentException(
                "$frameNumber is an invalid frame number."
            )
        }
    }
};

abstract fun getFrame(frameNumber: Int): Frame
}

```

Основной конструктор класса содержит два аргумента: **frameCount** — целочисленная переменная, указывающая число возможных фреймов, в которых может находиться форма. Второй аргумент **startPosition** указывает предполагаемую начальную позицию формы вдоль оси X в поле игрового процесса. В классе объявляется абстрактная функция **getFrame()**. Абстрактная функция не имеет реализации, ее поведение должно быть реализовано расширяющимся классом.

В классе создаются экземпляры класса (Tetramino), обеспечивающие реализацию объявленной абстрактной функции. Реализация функции **getFrame()** использует целочисленную переменную **frameNumber**, которая определяет, какой фрейм будет возвращаться.

Важно отметить, что Tetramino является одновременно и объектом и константой. Как правило, классы `enum` используются для создания констант.

Этот класс является идеальным выбором для моделирования форм тетрамино, поскольку в этом случае может реализовываться фиксированный набор форм.

Смоделировав фрейм блока и форму, необходимо программно смоделировать сам блок. Рассмотрим эту задачу как возможность демонстрации совместимости языков Kotlin и Java путем реализации модели с помощью Java. Создайте в пакете `models` новый класс Java под именем `Block.java`:

```
package com.example.tetris.models;

import android.graphics.Color;
import android.graphics.Point;

import androidx.annotation.NonNull;

import com.example.tetris.constants.FieldConstants;

import java.util.Random;

public class Block {
    private int shapeIndex;
    private int frameNumber;
    private BlockColor color;
    private Point position;

    public enum BlockColor {
        PINK(Color.rgb(255, 105, 180), (byte) 2),
        GREEN(Color.rgb(0, 128, 0), (byte) 3),
        ORANGE(Color.rgb(255, 140, 0), (byte) 4),
        YELLOW(Color.rgb(255, 255, 0), (byte) 5),
        CYAN(Color.rgb(0, 255, 255), (byte) 6);

        BlockColor(int rgbValue, byte value) {
            this.rgbValue = rgbValue;
            this.byteValue = value;
        }

        private final int rgbValue;
        private final byte byteValue;
    }
}
```

Переменная `shapeIndex` включает индекс формы блока; `frameNumber` отслеживает количество фреймов, относящихся к форме блока; `color` содержит цветовые характеристики блока; `position` используется для отслеживания текущей пространственной позиции блока в игровом поле.

Шаблон `enum BlockColor` добавляется внутри класса `Block`. Этот шаблон формирует постоянный набор экземпляров `BlockColor`, каждый из которых обладает свойствами `rgbValue` (цвет в палитре RGB) и `byteValue`.

Размеры игрового поля будут 20 строк × 10 столбцов. Эти значения надо задать константами, объявленными в отдельном файле.

Создайте в исходном пакете приложения пакет с именем constants и добавьте в него новый файл Kotlin с именем FieldConstants.kt. Затем введите код в файл:

```
enum class FieldConstants(val value: Int) {  
    COLUMN_COUNT(10), ROW_COUNT(20);  
}
```

Импортируйте этот пакет в файл Block.java:

```
import com.example.tetris.constants.FieldConstants;
```

Создайте конструктор класса Block:

```
private Block(int shapeIndex, BlockColor blockColor) {  
    this.frameNumber = 0;  
    this.shapeIndex = shapeIndex;  
    this.color = blockColor;  
    this.position = new Point(FieldConstants.COLUMN_COUNT.getValue() / 2,  
0);  
}
```

Конструктор должен быть доступен только внутри класса. Так как нужно разрешить другим классам создавать экземпляры блоков, следует определить статический метод, который это разрешает делать (createBlock):

```
public static Block createBlock() {  
    Random random = new Random();  
    int shapeIndex = random.nextInt(Shape.values().length);  
    BlockColor blockColor = BlockColor.values()  
        [random.nextInt(BlockColor.values().length)];  
    Block block = new Block(shapeIndex, blockColor);  
    block.position.x = block.position.x - Shape.values()  
        [shapeIndex].getStartPosition();  
    return block;  
}
```

Метод случайным createBlock() образом выбирает индекс для формы тетрамино в классе enum Shape и задает цвет блока, а затем присваивает два выбранных значения переменным shapeIndex и blockColor. Новый экземпляр класса Block создается с двумя переданными ему в качестве аргументов значениями, и позиция блока устанавливается вдоль оси X. После этого метод возвращает созданный блок.

Добавьте в класс Block несколько методов геттеров (получения) и сеттеров (установки). Эти методы позволят получать доступ к важнейшим свойствам экземпляров блока.

```
public static int getColor(byte value) {  
    for (BlockColor color : BlockColor.values()) {  
        if (value == color.byteValue) {  
            return color.rgbValue;  
        }  
    }  
}
```

```

        }
    }
    return -1;
}

public final void setState(int frame, Point position) {
    this.frameNumber = frame;
    this.position = position;
}

@NonNull
public final byte[][] getShape (int frameNumber) {
    return Shape.values()[shapeIndex].getFrame(
        frameNumber).as2dByteArray();
}

public Point getPosition() {
    return this.position;
}

public final int getFrameCount() {
    return Shape.values()[shapeIndex].getFrameCount();
}

public int getFrameNumber() {
    return frameNumber;
}

public int getColor() {
    return color.rgbValue;
}

public byte getStaticValue() {
    return color.byteValue;
}

```

Аннотация @NonNull предоставляется платформой приложения Android и обозначает, что возвращаемое поле, параметр или метод не могут иметь значение null. Для ее подключения нужно добавить импорт пакета в область перед описанием класса:

```
import androidx.annotation.NonNull;
```

Создание модели приложения

Создадим модель приложения, реализующую необходимую логику игрового поля Tetris, а также обслуживающую промежуточный интерфейс между представлениями и созданными компонентами блока.



Взаимодействие представления и модели приложения

Представление отправит модели приложения запрос на выполнение, модель выполнит действие, если оно допустимо, и отправит отзыв представлению. Для модели приложения необходим отдельный файл класса. Создайте в пакете `models` новый класс Kotlin с именем `AppModel`.

Сначала следует добавить константы, которые будут использоваться классом `AppModel`, – для возможных игровых статусов и движений, выполняемых в процессе игры. Эти константы создаются с помощью классов `enum`:

```

class AppModel {
    enum class Statuses {
        AWAITING_START, // состояние игры до запуска
        ACTIVE, // состояние игрового процесса (выполняется)
        INACTIVE, // состояние игрового процесса (не выполняется)
        OVER // статус, принимаемый игрой на момент ее завершения
    }

    enum class Motions {
        LEFT, RIGHT, DOWN, ROTATE
    }
}
  
```

Добавьте перед классом импорт пакетов:

```

import android.graphics.Point
import com.example.tetris.constants.FieldConstants
import com.example.tetris.helpers.array2dOfByte
import com.example.tetris.storage.AppPreferences
  
```

и в класс следующие свойства:

```

var score: Int = 0 // сохранение текущего счета игрока в игровом сеансе

// обеспечивает непосредственный доступ к файлу SharedPreferences
private var preferences: AppPreferences? = null
  
```

*// доступ к свойству, которое будет содержать текущий блок трасляции
через игровое поле*

```
var currentBlock: Block? = null
```

// содержит состояние игры

```
var currentState: String = Statuses.AWAITING_START.name
```

// двумерный массив, служащий в качестве игрового поля

```
private var field = array2dOfByte(  
    FieldConstants.ROW_COUNT.value,  
    FieldConstants.COLUMN_COUNT.value  
)
```

Добавьте несколько методов сеттеров и геттеров:

```
fun setPreferences(preferences: AppPreferences?) {  
    this.preferences = preferences  
}
```

```
fun getCellStatus(row: Int, column: Int): Byte? {  
    return field[row][column]  
}
```

```
private fun setCellStatus(row: Int, column: Int, status: Byte?) {  
    if (status != null) {  
        field[row][column] = status  
    }  
}
```

поскольку имеются три возможных статуса игры, которым соответствуют три игровых состояния, для каждого из них необходимо наличие функции. Добавьте в класс код методов:

```
fun isGameOver(): Boolean {  
    return currentState == Statuses.OVER.name  
}
```

```
fun isActive(): Boolean {  
    return currentState == Statuses.ACTIVE.name  
}
```

```
fun isGameAwaitingStart(): Boolean {  
    return currentState == Statuses.AWAITING_START.name  
}
```

Добавьте функцию, которая будет использоваться для увеличения значения очков, находящегося в некотором диапазоне:

```
private fun boostScore() {  
    score += 10  
    if (score > preferences?.getHighScore() as Int)  
        preferences?.saveHighScore(score)  
}
```

При вызове функция увеличивает текущий счет игрока на 10 очков, после чего проверяет, не превышает ли текущий счет игрока уже установленный рекорд, записанный в файле настроек. Если текущее значение больше сохраненного рекорда, то рекорд перезаписывается.

Функция `generateNextBlock()` создает новый экземпляр блока и устанавливает значение `currentBlock` равным вновь созданному экземпляру:

```
private fun generateNextBlock() {  
    currentBlock=Block.createBlock()  
}
```

Создайте в пакете `constants` файл `CellConstants.kt` и введите в него код:

```
enum class CellConstants(val value: Byte) {  
    EMPTY(0), EPHEMERAL(1)  
}
```

Для чего нужны эти константы? При создании класса `Frame` при моделировании фрейма блоков определялась функция `addRow()`, которая в качестве аргумента обрабатывает строку, состоящую из 0 и 1. Значение 1 соответствует ячейкам фрейма, а значение 0 записывается в ячейки, не принадлежащие фрейму.

Импортируйте созданный класс `enum` в `AppModel`:

```
import com.example.tetris.constants.CellConstants
```

А затем добавьте в этот класс функцию, проверяющую допустимость поступательного движения тетрамино в игровом поле на основе набора условий:

```
private fun validTranslation(position: Point, shape: Array<ByteArray>):  
Boolean {  
    return if (position.y < 0 || position.x < 0) {  
        false  
    } else if (position.y + shape.size > FieldConstants.ROW_COUNT.value)  
    {  
        false  
    } else if (position.x + shape[0].size >  
FieldConstants.COLUMN_COUNT.value) {  
        false  
    } else {  
        for (i in 0 until shape.size) {  
            for (j in 0 until shape[i].size) {  
                val y = position.y + i  
                val x = position.x + j  
                if (CellConstants.EMPTY.value != shape[i][j] &&  
                    CellConstants.EMPTY.value != field[y][x]  
                ) {  
                    return false  
                }  
            }  
        }  
    }  
    true  
}
```

```
}  
}
```

Метод возвращает логическое значение true, если трансляция корректна, и false – в противном случае. Первые три условия проверяют, находится ли в поле позиция, в которую переводится тетрамино. Блок else проверяет, свободны ли ячейки, в которые пытается перейти тетрамино. Если это не так, возвращается значение false.

Для использования этого метода нужна функция вызова. С этой целью добавьте функцию:

```
private fun moveValid(position: Point, frameNumber: Int?): Boolean{  
    val shape: Array<ByteArray>?=currentBlock?.getShape(frameNumber as  
Int)  
    return validTranslation(position,shape as Array<ByteArray>)  
}
```

с. 126 (132)