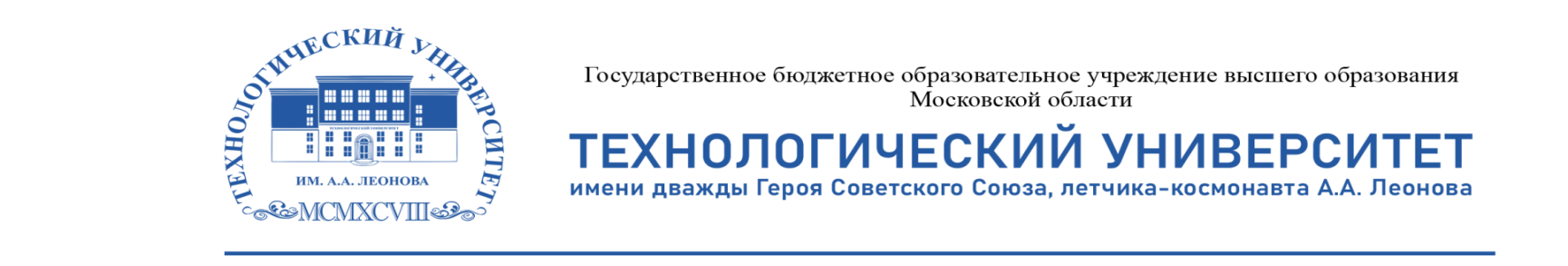
****

Колледж космического машиностроения и технологии

**Курсовой проект**

**РАЗРАБОТКА АНДРОИД-ПРИЛОЖЕНИЯ «ИГРА «ПРИКЛЮЧЕНИЕ ОДНОГО КОСМОНАВТА»».**

Пояснительная записка

КП.09.02.03.22.05ПЗ

Обучающийся группы П2-19 Дорохин М.А.

Руководитель курсового проекта Гусятинер Л.Б.

Результат защиты\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Королев, 2022 г.

Оглавление

[**Введение** 3](#_Toc107175241)

[**1 Теоретическая часть** 4](#_Toc107175242)

[**2 Проектная часть** 8](#_Toc107175247)

[**3.Эксплуатационная часть** 27](#_Toc107175248)

[**Заключение** 30](#_Toc107175249)

[**Приложение 1. Код главного модуля Main.kt** 31](#_Toc107175250)

[**Приложение 2. Код модуля music.** 43](#_Toc107175251)

[**Используемая литература** 44](#_Toc107175252)

# **Введение**

Данный курсовой проект заключается в разработке приключенческой для телефона. Разработка игры будет на Kotlin.

В первой части будут рассмотрены предметная область и существующие продукты по данной теме.

Во второй части будут описаны инструменты разработки программы и модули программы.

В третьей части описаны взаимодействие пользователя с программой и сообщения оператору.

В заключительной части будут приведены общие выводы по проекту.

# **1 Теоретическая часть**

**1.1 Описание предметной области**

**Приключенческая игра -** один из жанров игр, требующих от игрока решения задач для продвижения по сюжету. Сюжет может быть линейным или же разветвлённым, в котором концовка зависит полностью от дейстивий игрока.

**Функции игры**

Приключенческая игра выполняет несколько функций, развлечь пользователя (игрока) и позволить игроку интересно провести время.

Развлекательная функция, цель любой игры, веселье. Так что очевидно, что это же цель и нашей игры. Играя в неё, игрок должен получать удовольствие от интересного приключения и новых ощущений.

Время препровождение, часто человек хочет отдохнуть и расслабиться, провести интересно время. Моя игра создана для этого.

Виды приключенческих игр:

### Текстовые приключения

В них нет никакой графики, только текст и сюжет

### Графические квесты

Имеется графика и небольшие головоломки.

### Квесты-головоломки

В отличии от графических квестов тут присутствует большое кол-во головоломок

### Визуальные романы

Это текстовое приключение с небольшим количеством графики (например картинки)

**1.2 Описание существующих разработок**

В этом разделе рассмотрены некоторые уже имеющиеся игры, а также цели таких программ и требования.

Интерфейс таких игр чаще всего простой и понятный для обычных пользователей.

Цели использования таких игр:

1. Развитие интеллектуальных способностей.
2. Развлечение.

Основные требования:

1. Развлечение человека.
2. Наличие головоломок

**1.2.1 King's Quest**

King’s Quest (рис.1) - серия приключенческих фэнтезийных компьютерных игр от компании Sierra Entertainment.

King’s Quest описывает историю королевской семьи в вымышленном государстве Давентри. В игре вы играете за членов королевской семьи и вам предстоит прослушать историю короля Грэма о том, как он стал королем и его приключениях. В игре есть интересные головоломки, но она устарела и е нельзя запустить на новых версиях Windows.



Рис.1 Игровой процесс King’s Quest

**1.2.2 Mystery House**

**Mystery House** (рис.2) - приключенческая компьютерная игра, квест с использованием компьютерной графики.

Персонаж попадает в дом вместе с семью другими персонажами: сантехник Том, механик Сэм, швея Салли, хирург доктор Грин, могильщик Джо, мясник Билл и повар Дейзи. Изначально мы должны будем найти тайник в этом доме, но позже персонажи начинают умирать. Мы должны будет выяснить кто убийца и если возможно найти тайник.

[](https://ru-wiki.ru/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Mystery_House_-_Apple_II_-_3.png)

Рис.2 Игровой процесс Mystery House

### 2 Проектная часть

**2.1 Диаграмма прецедентов**

В этом разделе представлена диаграмма прецедентов (рис.3). На диаграмме показаны все возможные функциональные и поведенческие отношения.

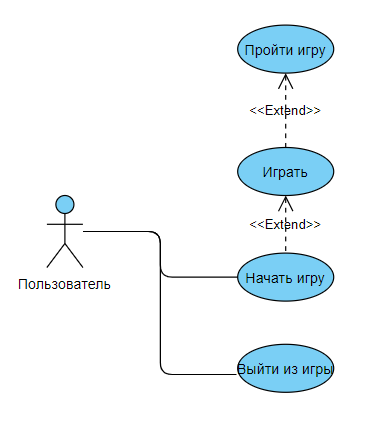
****

Рис. 3 Диаграмма прецедентов использования приложения

**2.2 Выбор инструментов**

При выборе инструментов было проведено сравнение по критериям, представленных в таблице 1.

Степень важности критерия выбиралась из: низкая, ниже средней, средняя, ниже высокой, высокая.

Таблица 1. Критерии выбора инструмента.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Критерий | Участие в корпоративном проекте | Простота сопровождения | Наличие библиотек | Наличие документации на русском языке | Скорость разработки |
| Важность критерия | Средняя | Высокая | Высокая | Ниже средней | Ниже высокой |

Исходя из этих критериев, выполнено сравнение 3 языков программирования от 0 до 10 баллов за критерий.

Таблица 2. Оценка языков программирования.

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий/язык программирования | Unity | Kotlin | Python |
| Участие в корпоративном проекте | 7 | 8 | 4 |
| Простота сопровождения | 7 | 10 | 3 |
| Наличие библиотек | 6 | 10 | 4 |
| Наличие документации на русском языке | 10 | 4 | 6 |
| Скорость разработки | 6 | 10 | 3 |
| Итого баллов | 36 | 42 | 20 |

По результатам сравнения был выбран язык программирования Kotlin.

**2.3 Проектирование сценария**

В данном разделе приведен сценарий использования программы пользователем. (рис.4)

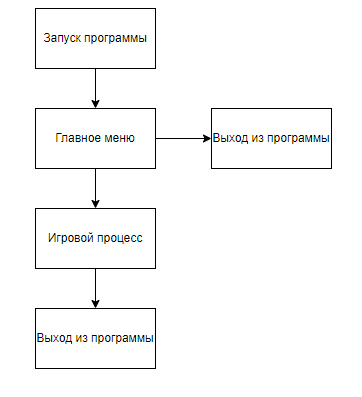


Рис. 4 Сценарий использования

Пользователь после запуска программы может выполнить 2 действия: начать игру или выйти из программы.

При выборе выхода программа заканчивает свою работу, при выборе начала игры пользователь может начать играть.

**2.4 Диаграмма классов**

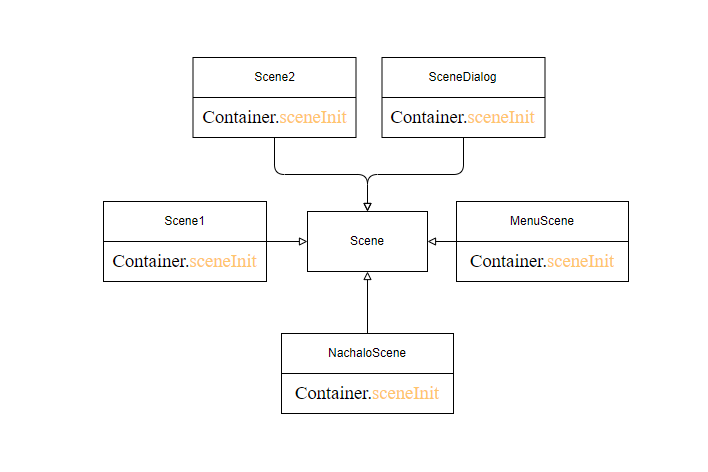
В данном разделе представлены все классы, использующиеся в проекте, а также их отношения между собой.(рис.5) [](https://i.imgur.com/DDMkhWe.png)

Рис. 5 Диаграмма классов для проекта

Класс «Scene» Встроенный класс движка KorGE, от него наследуются все остальные классы

Класс «MenuScene» содержит функции для кнопок в главном меню

Класс «NachaloScene» содержит функции для показа предисловия

Класс «Scene1» содержит функции для персонажей и локацию в начале игры

Класс «Scene2» содержит функции для персонажей и локаций в следующий сценах

Класс «SceneDialog» содержат функции для диалогов между персонажами

**2.5** **Описание главного модуля**

В главный модуль импортируется еще 1 модуля, этот модуль отвечает за все звуки в игре.

Также в главный модуль входит код, который запускает саму программу при ее запуске. Полный код главного модуля находится в приложении 1.

**Листинг 1. Обработка всех событий в программе.**

suspend fun main() = Korge(Korge.Config(module = ConfigModule))  
  
object ConfigModule : Module(){  
 override val bgcolor = Colors["#2b2b2b"]  
 override val size = sizeAll  
 override val mainScene : KClass<out Scene> = MenuScene::class  
  
 override suspend fun AsyncInjector.configure(){  
 mapPrototype **{** MenuScene() **}** mapPrototype **{** NachaloScene() **}** mapPrototype **{** Scene1() **}** mapPrototype **{** Scene2() **}** mapPrototype **{** SceneDialog() **}** }  
}

Данный метод отслеживает все действия пользователя и в зависимости от конкретного действия вызывает нужный метод.

**Листинг 2. Классы.**

class MenuScene() : Scene(){  
 override suspend fun Container.sceneInit(){  
  
 addUpdater **{  
  
  
 }** uiButton(text = "Начать игру") **{** size(300, 100)  
 textSize = 30.0  
 position(sizeAll.width/2 - 300, sizeAll.height/2 - 100)  
 onPress **{** launch **{** sceneContainer.changeTo<NachaloScene>()  
 **}  
 }** scale(2)  
 **}** }  
}  
  
class NachaloScene(): Scene(){  
 override suspend fun Container.sceneInit() {  
 uiButton(text = "Спасти космонавта!")**{** position(sizeAll.width/2, sizeAll.height-70)  
 onPress**{**launch **{** sceneContainer.changeTo<Scene1>() **} }** size(300,50)  
 **}** uiText("Предисловие",  
 (sizeAll.width).toDouble(), //ширина  
 (sizeAll.height).toDouble()) //высота  
 **{** textSize = 75.0  
 **}** }  
}  
  
class Scene1() : Scene(){  
 override suspend fun Container.sceneInit(){  
 val input = views.input  
 val fon = resourcesVfs["Angar\_New.png"].readBitmap()  
 val fon\_image = image(fon).scale(2)  
  
 //region Анимации  
 val spritemap = resourcesVfs["player.png"].readBitmap()  
 val playerAnimationIdle = SpriteAnimation(  
 spriteMap = spritemap,  
 spriteWidth = 59, //Ширина одного слайда в пикселях  
 spriteHeight = 116, //Высота одного слайда в пикселях  
 marginTop = 0,  
 marginLeft = 0,  
 columns = 2, //Количество столбцов  
 rows = 1, //количество строчек  
 offsetBetweenColumns = 0,  
 offsetBetweenRows = 0  
 )  
  
 val spritemapleft = resourcesVfs["left.png"].readBitmap()  
 val playerAnimationLeft = SpriteAnimation(  
 spriteMap = spritemapleft,  
 spriteWidth = 128, //Ширина одного слайда в пикселях  
 spriteHeight = 130, //Высота одного слайда в пикселях  
 marginTop = 0,  
 marginLeft = 0,  
 columns = 4, //Количество столбцов  
 rows = 1, //количество строчек  
 offsetBetweenColumns = 0,  
 offsetBetweenRows = 0  
 )  
  
 val spritemapRight = resourcesVfs["right.png"].readBitmap()  
 val playerAnimationRight = SpriteAnimation(  
 spriteMap = spritemapRight,  
 spriteWidth = 128, //Ширина одного слайда в пикселях  
 spriteHeight = 128, //Высота одного слайда в пикселях  
 marginTop = 0,  
 marginLeft = 0,  
 columns = 4, //Количество столбцов  
 rows = 1, //количество строчек  
 offsetBetweenColumns = 0,  
 offsetBetweenRows = 0  
 )  
  
 val spritemapRightFly = resourcesVfs["polet\_vpravo.png"].readBitmap()  
 val playerAnimationRightFly = SpriteAnimation(  
 spriteMap = spritemapRightFly,  
 spriteWidth = 128, //Ширина одного слайда в пикселях  
 spriteHeight = 128, //Высота одного слайда в пикселях  
 marginTop = 0,  
 marginLeft = 0,  
 columns = 2, //Количество столбцов  
 rows = 1, //количество строчек  
 offsetBetweenColumns = 0,  
 offsetBetweenRows = 0  
 )  
  
 val spritemapGagarin = resourcesVfs["Gagarin.png"].readBitmap()  
 val gagarinIdle= SpriteAnimation(  
 spriteMap = spritemapGagarin,  
 spriteWidth = 82, //Ширина одного слайда в пикселях  
 spriteHeight = 93, //Высота одного слайда в пикселях  
 marginTop = 0,  
 marginLeft = 0,  
 columns = 2, //Количество столбцов  
 rows = 1, //количество строчек  
 offsetBetweenColumns = 0,  
 offsetBetweenRows = 0  
 )  
//endregion  
  
 val gagarin = sprite(gagarinIdle).scale(3).position(sizeAll.width-600.0, sizeAll.height - 300.0)  
 gagarin.playAnimationLooped(spriteDisplayTime = 500.milliseconds)  
  
 //region Игрок  
 lateinit var player: Sprite  
 player = sprite(playerAnimationIdle).size(59, 116).scale(1.5).position(0, 0)  
 player.playAnimationLooped(spriteDisplayTime = 500.milliseconds)  
 player.onAnimationCompleted **{** player.playAnimationLooped(playerAnimationIdle, spriteDisplayTime = 500.milliseconds) **}** //endregion  
  
 var isPlaying = false  
  
 val jump = getJumpMusic()  
  
 val debugText = text("", textSize = 16.0, alignment = TextAlignment.TOP\_LEFT).position(32.0, 32.0)  
  
 var box = container **{** val qube = solidRect(200,200, Colors.RED).position(0,0)  
 val bottomHit = solidRect(190,1, Colors.TRANSPARENT\_BLACK).position(5,200).registerBodyWithFixture(type = BodyType.STATIC)  
 val leftHit = solidRect(1, 190, Colors.TRANSPARENT\_BLACK).position(-1, 5).registerBodyWithFixture(type = BodyType.STATIC)  
 val rightHit = solidRect(1, 190, Colors.TRANSPARENT\_BLACK).position(201, 5).registerBodyWithFixture(type = BodyType.STATIC)  
 val upHit = solidRect(190,1, Colors.TRANSPARENT\_BLACK).position(5,-1)  
 **}**.position(700, sizeAll.height-100)  
 var box1 = container **{** val qube = solidRect(200,200, Colors.RED).position(0,0)  
 val bottomHit = solidRect(190,1, Colors.TRANSPARENT\_BLACK).position(5,200).registerBodyWithFixture(type = BodyType.STATIC)  
 val leftHit = solidRect(1, 190, Colors.TRANSPARENT\_BLACK).position(-1, 5).registerBodyWithFixture(type = BodyType.STATIC)  
 val rightHit = solidRect(1, 190, Colors.TRANSPARENT\_BLACK).position(201, 5).registerBodyWithFixture(type = BodyType.STATIC)  
 val upHit = solidRect(190,1, Colors.TRANSPARENT\_BLACK).position(5,-1)  
 **}**.position(900, sizeAll.height-100)  
 var box2 = container **{** val qube = solidRect(200,200, Colors.RED).position(0,0)  
 val bottomHit = solidRect(190,1, Colors.TRANSPARENT\_BLACK).position(5,200).registerBodyWithFixture(type = BodyType.STATIC)  
 val leftHit = solidRect(1, 190, Colors.TRANSPARENT\_BLACK).position(-1, 5).registerBodyWithFixture(type = BodyType.STATIC)  
 val rightHit = solidRect(1, 190, Colors.TRANSPARENT\_BLACK).position(201, 5).registerBodyWithFixture(type = BodyType.STATIC)  
 val upHit = solidRect(190,1, Colors.TRANSPARENT\_BLACK).position(5,-1)  
 **}**.position(1100, sizeAll.height-300)  
  
 //Сделать списки для верхних и нижних хитбоксов  
 val upHitboxList: List<View> = mutableListOf(box.getChildAt(4),box1.getChildAt(4),box2.getChildAt(4))  
 val bottomHitboxList: List<View> = mutableListOf(box.getChildAt(1),box1.getChildAt(1),box2.getChildAt(1))  
 val leftHitboxList: List<View> = mutableListOf(box.getChildAt(2),box1.getChildAt(2),box2.getChildAt(2))  
 val rightHitboxList: List<View> = mutableListOf(box.getChildAt(3),box1.getChildAt(3),box2.getChildAt(3))  
  
 var btn = uiButton(text = "Говорить!")**{** position(sizeAll.width/2, sizeAll.height/2)  
 onPress**{** launch **{** sceneContainer.changeTo<SceneDialog>() **} }** size(300,100)  
 **}** btn.enabled = false  
 btn.visible = false  
  
 player.addUpdater **{** try {  
 debugText.text = player.y.toString()  
 val ntouches: Int = input.activeTouches.size  
 val touches: List<Touch> = input.activeTouches  
  
 var num = sizeAll.width / 4.0  
  
 //region Гравитация  
 if (player.y < sizeAll.height - 200 && !collidesWith(upHitboxList))  
 player.y += 5  
 if (collidesWith(bottomHitboxList))  
 player.y += 10  
 if (touches.size == 0){  
 player.playAnimation(playerAnimationIdle)  
 }  
 //endregion  
 if(collidesWith(gagarin)){  
 btn.enabled = true  
 btn.visible = true  
 }  
 else{  
 btn.enabled = false  
 btn.visible = false  
 }  
 var isFlying = false  
 touches.forEach **{** // Передвижение налево  
 if (it.x in 0.0..num && player.pos.x >= 10.0 && !collidesWith(rightHitboxList)) {  
 player.x -= 6  
 }  
 else if (it.x in num..num \* 2 && player.pos.x <= sizeAll.width - 160.0 && !collidesWith(leftHitboxList)) {  
 player.x += 6  
  
  
 if (player.pos.x == sizeAll.width - 180.0) {  
 launch **{** sceneContainer.changeTo<Scene2>()  
 **}** }  
 }  
 if (it.x in num \* 2..num \* 4 && player.pos.y >= 0 && !collidesWith(bottomHitboxList)) {  
 isFlying = true  
 player.y -= 10  
  
 if (!isPlaying) {  
 launch **{** jump.play()  
 isPlaying = true  
 delay(10L)  
 isPlaying = false  
  
 **}** }  
 }  
 else{  
 isFlying = false  
 }  
  
 **}** }  
 catch (e: Exception){  
 debugText.text = e.toString()  
 }  
  
 **}** }  
}  
  
class Scene2() : Scene(){  
 override suspend fun Container.sceneInit(){  
 val input = views.input  
 val fon = resourcesVfs["mishaeblan.jpg"].readBitmap()  
 val fon\_image = image(fon).scale(2)  
  
 //region Анимации  
 val spritemap = resourcesVfs["player\_model.png"].readBitmap()  
 val playerAnimation = SpriteAnimation(  
 spriteMap = spritemap,  
 spriteWidth = 128, //Ширина одного слайда в пикселях  
 spriteHeight = 128, //Высота одного слайда в пикселях  
 marginTop = 0,  
 marginLeft = 0,  
 columns = 2, //Количество столбцов  
 rows = 1, //количество строчек  
 offsetBetweenColumns = 0,  
 offsetBetweenRows = 0  
 )  
  
 val spritemapleft = resourcesVfs["left.png"].readBitmap()  
 val playerAnimationLeft = SpriteAnimation(  
 spriteMap = spritemapleft,  
 spriteWidth = 128, //Ширина одного слайда в пикселях  
 spriteHeight = 128, //Высота одного слайда в пикселях  
 marginTop = 0,  
 marginLeft = 0,  
 columns = 4, //Количество столбцов  
 rows = 1, //количество строчек  
 offsetBetweenColumns = 0,  
 offsetBetweenRows = 0  
 )  
  
 val spritemapRight = resourcesVfs["right.png"].readBitmap()  
 val playerAnimationRight = SpriteAnimation(  
 spriteMap = spritemapRight,  
 spriteWidth = 128, //Ширина одного слайда в пикселях  
 spriteHeight = 128, //Высота одного слайда в пикселях  
 marginTop = 0,  
 marginLeft = 0,  
 columns = 4, //Количество столбцов  
 rows = 1, //количество строчек  
 offsetBetweenColumns = 0,  
 offsetBetweenRows = 0  
 )  
  
 val spritemapRightFly = resourcesVfs["polet\_vpravo.png"].readBitmap()  
 val playerAnimationRightFly = SpriteAnimation(  
 spriteMap = spritemapRightFly,  
 spriteWidth = 128, //Ширина одного слайда в пикселях  
 spriteHeight = 128, //Высота одного слайда в пикселях  
 marginTop = 0,  
 marginLeft = 0,  
 columns = 2, //Количество столбцов  
 rows = 1, //количество строчек  
 offsetBetweenColumns = 0,  
 offsetBetweenRows = 0  
 )  
//endregion  
  
 //region Игрок  
 lateinit var player: Sprite  
 player = sprite(playerAnimation).size(180, 180).position(0, 0)  
 player.playAnimationLooped(spriteDisplayTime = 500.milliseconds)  
 player.onAnimationCompleted **{** player.playAnimationLooped(playerAnimation, spriteDisplayTime = 500.milliseconds) **}** //endregion  
  
 val testPlatformHitBox = solidRect(700,20, Colors.RED).position(700,600).registerBodyWithFixture(type = BodyType.STATIC)  
 val bottomHit = solidRect(700,20, Colors.TRANSPARENT\_BLACK).position(700,601).registerBodyWithFixture(type = BodyType.STATIC)  
 val leftHit = solidRect(1, 1, Colors.TRANSPARENT\_BLACK).position(700, 610).registerBodyWithFixture(type = BodyType.STATIC)  
  
 var isPlaying = false  
  
 val debugText = text("", textSize = 16.0, alignment = TextAlignment.TOP\_LEFT).position(32.0, 32.0)  
 val jump = resourcesVfs["jump.mp3"].readSound()  
  
 player.addUpdater **{** try {  
 val ntouches: Int = input.activeTouches.size  
 val touches: List<Touch> = input.activeTouches  
  
 var num = sizeAll.width / 4.0  
  
 //region Гравитация  
 if (player.y < sizeAll.height - 200 && !collidesWith(testPlatformHitBox))  
 player.y += 5  
 else if (collidesWith(bottomHit))  
 player.y += 5  
 if (touches.size == 0){  
 player.playAnimation(playerAnimation)  
 }  
 //endregion  
 touches.forEach **{** // Передвижение налево  
 if (it.x in 0.0..num && player.pos.x >= 10.0) {  
 player.x -= 6  
 player.playAnimation(playerAnimationLeft, spriteDisplayTime = 200.milliseconds)  
 }  
 else if (it.x in num..num \* 2 && player.pos.x <= sizeAll.width - 160.0 && !collidesWith(leftHit)) {  
 player.x += 6  
 player.playAnimation(playerAnimationRight, spriteDisplayTime = 200.milliseconds)  
  
 if (player.pos.x == sizeAll.width - 180.0) {  
 launch **{** sceneContainer.changeTo<Scene1>()  
 **}** }  
 }  
 if (it.x in num \* 2..num \* 4 && player.pos.y >= 0 && !collidesWith(bottomHit)) {  
 player.y -= 10  
 if (!isPlaying) {  
 launch **{** jump.play()  
 isPlaying = true  
 delay(10L)  
 isPlaying = false  
 **}** }  
 }  
 debugText.text = player.x.toString()  
 **}** }  
 catch (e: Exception){  
 debugText.text = e.toString()  
 }  
  
 **}**}}  
  
class SceneDialog(): Scene(){  
 override suspend fun Container.sceneInit(){  
 val fon = resourcesVfs["Angar\_New.png"].readBitmap()  
  
 val fon\_image = image(fon).scale(2)  
  
 uiText(" Тестовый текст",  
 (sizeAll.width).toDouble(), //ширина  
 (sizeAll.height).toDouble()) //высота  
 **{** textSize = 75.0  
 position(sizeAll.width-600.0, sizeAll.height - 300.0)  
 **}** }

С помощью этих классов и работает программа (подробнее в диаграммах классов)

**2.6 Описание спецификаций к модулям**

В данном разделе описаны публичные члены модулей курсового проекта.

В главном модуле (main) содержатся публичные методы классов, которые наследуется от класса Scene движка KorGE. Так как все эти методы были описаны в разделе 2.5, в данном разделе описание данных методов опущено.

Объекты классов:

1. textsize – Размер текста.
2. size – размер какого-либо объекта
3. spriteWidth - Ширина одного слайда в пикселях
4. spriteHeight - Высота одного слайда в пикселях
5. columns - Количество столбцов
6. rows - количество строчек

**2.7 Описание модулей**

Кроме главного модуля программа содержит еще один модуль, который реализует звуки (music).

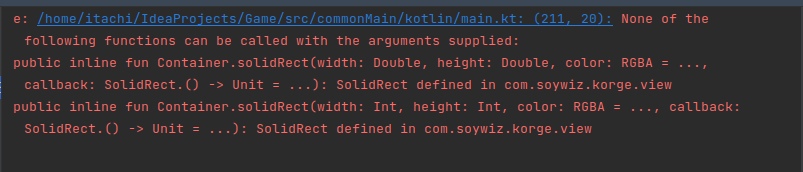
Полный код модуля приведен в приложении 2.

**Листинг. Методы класса Music.**

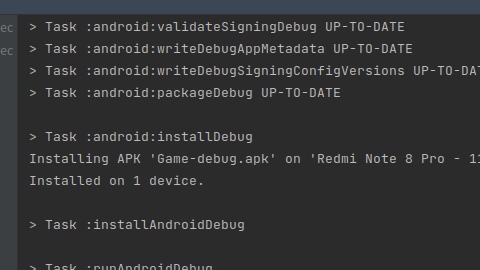
public suspend fun getJumpMusic(): Sound {  
 return resourcesVfs["jump.mp3"].readSound()  
}

**2.8 Описание применения средств отладки**

С помощью встроенный в IDE средств отладки находим ошибку и читаем причину ошибку.



Исправляем её и запускаем средство отладки для проверки.



Ошибка исправлена, все работает.

**3.Эксплуатационная часть**

**Руководство оператора**

**АННОТАЦИЯ**

В данном программном документе приведено руководство оператора по применению и эксплуатации игры «Приключение одного космонавта», предназначенной для расслабления пользователя.

В данном программном документе, в разделе «Назначение программы» указаны сведения о назначении программы и информация, достаточная для понимания функций программы.

В разделе «Условия выполнения программы» указаны условия, необходимые для выполнения программы (минимальный состав аппаратных и программных средств и т.п.).

В данном программном документе, в разделе «Выполнение программы» указана последовательность действий оператора, обеспечивающих загрузку, запуск, выполнение и завершение программы, приведено описание функций, формата и возможных вариантов команд, с помощью которых оператор осуществляет загрузку и управляет выполнением программы, а также ответы программы на эти команды.

Оформление программного документа «Руководство оператора» произведено по требованиям ЕСПД (ГОСТ 19.101-77 [[1]](#footnote-1)1), ГОСТ 19.103-77 [[2]](#footnote-2)2), ГОСТ 19.104-78\* [[3]](#footnote-3)3), ГОСТ 19.105-78\* [[4]](#footnote-4)4), ГОСТ 19.106-78\* [[5]](#footnote-5)5), ГОСТ 19.505-79\* [[6]](#footnote-6)6), ГОСТ 19.604-78\* [[7]](#footnote-7)7)).

**Назначение программы**

**Функциональное назначение программы**

Игра «Приключение одного космонавта» используется для расслабления пользователя.

**Эксплуатационное назначение программы**

Игра «Приключение одного космонавта» может использоваться только 1 пользователем

**Условия выполнения программы**

OC: Android 8.0

Процессор: 32-битные и 64-битные: ARM, MIPS, x86. Тактовая частота не ниже 1,8 ГГц.

ОЗУ: 2 ГБ

Место на диске: 16 МБ.

Дисплей: 2340x1080

**Выполнение программы**

**Загрузка и запуск программы**

Вам нужно запустить apk приложения



Рис. 6 Главное меню

Кнопки меню имеют следующие функции:

Начать игру – начинает игру.

**Выполнение программы**

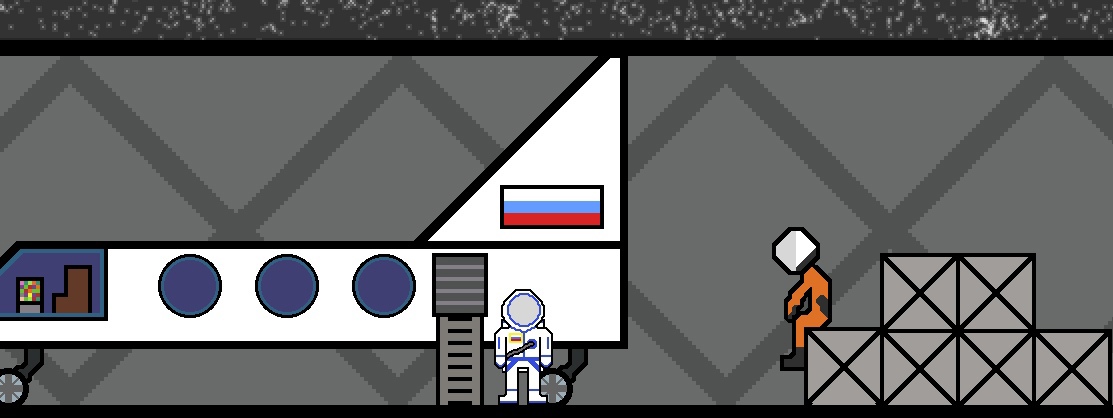


Рис. Игровой процесс

После нажатие на кнопку начинается сама игра. Что-бы управлять космонавтом можно нажать на левую часть экрана (что-бы идти налево), немного правее (что-бы идти направо) или на правую часть экрана (что-бы прыгнуть и использовать ранец).

**Завершение работы программы**

Чтобы завершить работу программы сверните ее, как и любое другое android приложение

# **Заключение**

В результате выполнения курсового проекта было написано андроид-приложение «Игра «Приключение одного космонавта»» для расслабления пользователя.

В ходе работы были проанализированы предметная область, существующие разработки, посвященные данному направлению, получены практические навыки по созданию android приложений с помощью языка Kotlin.

Также планируется продолжать работу над данным проектом с целью расширения возможностей и удобства приложения для пользователей. Планы по доработкам представлены ниже.

To-do лист:

1. Доработка всех пунктов главного меню, а именно настройки звука и разрешения экрана.
2. Добавление продолжения игры
3. Доработка интерфейса с целью упрощения работы с программой.
4. Улучшение дизайна интерфейса.
5. Добавление кнопки выхода из игры
6. Добавление хорошего сюжета

# **Приложение 1. Код главного модуля Main.kt**

import com.soywiz.klock.milliseconds  
import com.soywiz.korau.sound.readSound  
import com.soywiz.korev.Touch  
import com.soywiz.korge.\*  
import com.soywiz.korge.box2d.registerBodyWithFixture  
import com.soywiz.korge.scene.Module  
import com.soywiz.korge.scene.Scene  
import com.soywiz.korge.ui.\*  
import com.soywiz.korge.view.\*  
import com.soywiz.korge3d.format.ColladaParser  
import com.soywiz.korim.color.Colors  
import com.soywiz.korim.format.readBitmap  
import com.soywiz.korim.text.TextAlignment  
import com.soywiz.korinject.AsyncInjector  
import com.soywiz.korio.async.launch  
import com.soywiz.korio.async.launchUnscoped  
import com.soywiz.korio.file.std.resourcesVfs  
import com.soywiz.korma.geom.SizeInt  
import kotlinx.coroutines.GlobalScope  
import kotlinx.coroutines.delay  
import music.getJumpMusic  
import org.jbox2d.dynamics.BodyType  
import kotlin.reflect.KClass  
  
val sizeAll = SizeInt(2340, 1024)  
  
suspend fun main() = Korge(Korge.Config(module = ConfigModule))  
  
object ConfigModule : Module(){  
 override val bgcolor = Colors["#2b2b2b"]  
 override val size = sizeAll  
 override val mainScene : KClass<out Scene> = MenuScene::class  
  
 override suspend fun AsyncInjector.configure(){  
 mapPrototype **{** MenuScene() **}** mapPrototype **{** NachaloScene() **}** mapPrototype **{** Scene1() **}** mapPrototype **{** Scene2() **}** mapPrototype **{** SceneDialog() **}** }  
}  
  
class MenuScene() : Scene(){  
 override suspend fun Container.sceneInit(){  
uiButton(text = "Начать игру") **{** size(300, 100)  
 textSize = 30.0  
 position(sizeAll.width/2 - 300, sizeAll.height/2 - 100)  
 onPress **{** launch **{** sceneContainer.changeTo<NachaloScene>()  
 **}  
 }** scale(2)  
 **}** }  
}  
  
class NachaloScene(): Scene(){  
 override suspend fun Container.sceneInit() {  
 uiButton(text = "Спасти космонавта!")**{** position(sizeAll.width/2, sizeAll.height-70)  
 onPress**{**launch **{** sceneContainer.changeTo<Scene1>() **} }** size(300,50)  
 **}**  uiText("Предисловие",  
 (sizeAll.width).toDouble(), //ширина  
 (sizeAll.height).toDouble()) //высота  
 **{** textSize = 75.0  
 **}** }  
}  
  
class Scene1() : Scene(){  
 override suspend fun Container.sceneInit(){  
 val input = views.input  
 val fon = resourcesVfs["Angar\_New.png"].readBitmap()  
 val fon\_image = image(fon).scale(2)  
  
 //region Анимации  
 val spritemap = resourcesVfs["player.png"].readBitmap()  
 val playerAnimationIdle = SpriteAnimation(  
 spriteMap = spritemap,  
 spriteWidth = 59, //Ширина одного слайда в пикселях  
 spriteHeight = 116, //Высота одного слайда в пикселях  
 marginTop = 0,  
 marginLeft = 0,  
 columns = 2, //Количество столбцов  
 rows = 1, //количество строчек  
 offsetBetweenColumns = 0,  
 offsetBetweenRows = 0  
 )  
  
 val spritemapleft = resourcesVfs["left.png"].readBitmap()  
 val playerAnimationLeft = SpriteAnimation(  
 spriteMap = spritemapleft,  
 spriteWidth = 128, //Ширина одного слайда в пикселях  
 spriteHeight = 130, //Высота одного слайда в пикселях  
 marginTop = 0,  
 marginLeft = 0,  
 columns = 4, //Количество столбцов  
 rows = 1, //количество строчек  
 offsetBetweenColumns = 0,  
 offsetBetweenRows = 0  
 )  
  
 val spritemapRight = resourcesVfs["right.png"].readBitmap()  
 val playerAnimationRight = SpriteAnimation(  
 spriteMap = spritemapRight,  
 spriteWidth = 128, //Ширина одного слайда в пикселях  
 spriteHeight = 128, //Высота одного слайда в пикселях  
 marginTop = 0,  
 marginLeft = 0,  
 columns = 4, //Количество столбцов  
 rows = 1, //количество строчек  
 offsetBetweenColumns = 0,  
 offsetBetweenRows = 0  
 )  
  
 val spritemapRightFly = resourcesVfs["polet\_vpravo.png"].readBitmap()  
 val playerAnimationRightFly = SpriteAnimation(  
 spriteMap = spritemapRightFly,  
 spriteWidth = 128, //Ширина одного слайда в пикселях  
 spriteHeight = 128, //Высота одного слайда в пикселях  
 marginTop = 0,  
 marginLeft = 0,  
 columns = 2, //Количество столбцов  
 rows = 1, //количество строчек  
 offsetBetweenColumns = 0,  
 offsetBetweenRows = 0  
 )  
  
 val spritemapGagarin = resourcesVfs["Gagarin.png"].readBitmap()  
 val gagarinIdle= SpriteAnimation(  
 spriteMap = spritemapGagarin,  
 spriteWidth = 82, //Ширина одного слайда в пикселях  
 spriteHeight = 93, //Высота одного слайда в пикселях  
 marginTop = 0,  
 marginLeft = 0,  
 columns = 2, //Количество столбцов  
 rows = 1, //количество строчек  
 offsetBetweenColumns = 0,  
 offsetBetweenRows = 0  
 )  
//endregion  
  
 val gagarin = sprite(gagarinIdle).scale(3).position(sizeAll.width-600.0, sizeAll.height - 300.0)  
 gagarin.playAnimationLooped(spriteDisplayTime = 500.milliseconds)  
  
 //region Игрок  
 lateinit var player: Sprite  
 player = sprite(playerAnimationIdle).size(59, 116).scale(1.5).position(0, 0)  
 player.playAnimationLooped(spriteDisplayTime = 500.milliseconds)  
 player.onAnimationCompleted **{** player.playAnimationLooped(playerAnimationIdle, spriteDisplayTime = 500.milliseconds) **}** //endregion  
  
  
 var isPlaying = false  
  
 val jump = getJumpMusic()  
  
 val debugText = text("", textSize = 16.0, alignment = TextAlignment.TOP\_LEFT).position(32.0, 32.0)  
  
 var box = container **{** val qube = solidRect(200,200, Colors.RED).position(0,0)  
 val bottomHit = solidRect(190,1, Colors.TRANSPARENT\_BLACK).position(5,200).registerBodyWithFixture(type = BodyType.STATIC)  
 val leftHit = solidRect(1, 190, Colors.TRANSPARENT\_BLACK).position(-1, 5).registerBodyWithFixture(type = BodyType.STATIC)  
 val rightHit = solidRect(1, 190, Colors.TRANSPARENT\_BLACK).position(201, 5).registerBodyWithFixture(type = BodyType.STATIC)  
 val upHit = solidRect(190,1, Colors.TRANSPARENT\_BLACK).position(5,-1)  
 **}**.position(700, sizeAll.height-100)  
 var box1 = container **{** val qube = solidRect(200,200, Colors.RED).position(0,0)  
 val bottomHit = solidRect(190,1, Colors.TRANSPARENT\_BLACK).position(5,200).registerBodyWithFixture(type = BodyType.STATIC)  
 val leftHit = solidRect(1, 190, Colors.TRANSPARENT\_BLACK).position(-1, 5).registerBodyWithFixture(type = BodyType.STATIC)  
 val rightHit = solidRect(1, 190, Colors.TRANSPARENT\_BLACK).position(201, 5).registerBodyWithFixture(type = BodyType.STATIC)  
 val upHit = solidRect(190,1, Colors.TRANSPARENT\_BLACK).position(5,-1)  
 **}**.position(900, sizeAll.height-100)  
 var box2 = container **{** val qube = solidRect(200,200, Colors.RED).position(0,0)  
 val bottomHit = solidRect(190,1, Colors.TRANSPARENT\_BLACK).position(5,200).registerBodyWithFixture(type = BodyType.STATIC)  
 val leftHit = solidRect(1, 190, Colors.TRANSPARENT\_BLACK).position(-1, 5).registerBodyWithFixture(type = BodyType.STATIC)  
 val rightHit = solidRect(1, 190, Colors.TRANSPARENT\_BLACK).position(201, 5).registerBodyWithFixture(type = BodyType.STATIC)  
 val upHit = solidRect(190,1, Colors.TRANSPARENT\_BLACK).position(5,-1)  
 **}**.position(1100, sizeAll.height-300)  
  
 //Сделать списки для верхних и нижних хитбоксов  
 val upHitboxList: List<View> = mutableListOf(box.getChildAt(4),box1.getChildAt(4),box2.getChildAt(4))  
 val bottomHitboxList: List<View> = mutableListOf(box.getChildAt(1),box1.getChildAt(1),box2.getChildAt(1))  
 val leftHitboxList: List<View> = mutableListOf(box.getChildAt(2),box1.getChildAt(2),box2.getChildAt(2))  
 val rightHitboxList: List<View> = mutableListOf(box.getChildAt(3),box1.getChildAt(3),box2.getChildAt(3))  
  
  
 var btn = uiButton(text = "Говорить!")**{** position(sizeAll.width/2, sizeAll.height/2)  
 onPress**{** launch **{** sceneContainer.changeTo<SceneDialog>() **} }** size(300,100)  
 **}** btn.enabled = false  
 btn.visible = false  
  
 player.addUpdater **{** try {  
 debugText.text = player.y.toString()  
 val ntouches: Int = input.activeTouches.size  
 val touches: List<Touch> = input.activeTouches  
  
 var num = sizeAll.width / 4.0  
  
 //region Гравитация  
 if (player.y < sizeAll.height - 200 && !collidesWith(upHitboxList))  
 player.y += 5  
 if (collidesWith(bottomHitboxList))  
 player.y += 10  
 if (touches.size == 0){  
 player.playAnimation(playerAnimationIdle)  
 }  
 //endregion  
 if(collidesWith(gagarin)){  
 btn.enabled = true  
 btn.visible = true  
 }  
 else{  
 btn.enabled = false  
 btn.visible = false  
 }  
 var isFlying = false  
 touches.forEach **{** // Передвижение налево  
 if (it.x in 0.0..num && player.pos.x >= 10.0 && !collidesWith(rightHitboxList)) {  
 player.x -= 6  
 }  
 else if (it.x in num..num \* 2 && player.pos.x <= sizeAll.width - 160.0 && !collidesWith(leftHitboxList)) {  
 player.x += 6  
 if (player.pos.x == sizeAll.width - 180.0) {  
 launch **{** sceneContainer.changeTo<Scene2>()  
 **}** }  
 }  
 if (it.x in num \* 2..num \* 4 && player.pos.y >= 0 && !collidesWith(bottomHitboxList)) {  
 isFlying = true  
 player.y -= 10  
 if (!isPlaying) {  
 launch **{** jump.play()  
 isPlaying = true  
 delay(10L)  
 isPlaying = false  
  
 **}** }  
 }  
 else{  
 isFlying = false  
 }  
  
 **}** }  
 catch (e: Exception){  
 debugText.text = e.toString()  
 }  
  
 **}** }  
}  
  
class Scene2() : Scene(){  
 override suspend fun Container.sceneInit(){  
 val input = views.input  
 val fon = resourcesVfs["mishaeblan.jpg"].readBitmap()  
 val fon\_image = image(fon).scale(2)  
  
 //region Анимации  
 val spritemap = resourcesVfs["player\_model.png"].readBitmap()  
 val playerAnimation = SpriteAnimation(  
 spriteMap = spritemap,  
 spriteWidth = 128, //Ширина одного слайда в пикселях  
 spriteHeight = 128, //Высота одного слайда в пикселях  
 marginTop = 0,  
 marginLeft = 0,  
 columns = 2, //Количество столбцов  
 rows = 1, //количество строчек  
 offsetBetweenColumns = 0,  
 offsetBetweenRows = 0  
 )  
  
 val spritemapleft = resourcesVfs["left.png"].readBitmap()  
 val playerAnimationLeft = SpriteAnimation(  
 spriteMap = spritemapleft,  
 spriteWidth = 128, //Ширина одного слайда в пикселях  
 spriteHeight = 128, //Высота одного слайда в пикселях  
 marginTop = 0,  
 marginLeft = 0,  
 columns = 4, //Количество столбцов  
 rows = 1, //количество строчек  
 offsetBetweenColumns = 0,  
 offsetBetweenRows = 0  
 )  
  
 val spritemapRight = resourcesVfs["right.png"].readBitmap()  
 val playerAnimationRight = SpriteAnimation(  
 spriteMap = spritemapRight,  
 spriteWidth = 128, //Ширина одного слайда в пикселях  
 spriteHeight = 128, //Высота одного слайда в пикселях  
 marginTop = 0,  
 marginLeft = 0,  
 columns = 4, //Количество столбцов  
 rows = 1, //количество строчек  
 offsetBetweenColumns = 0,  
 offsetBetweenRows = 0  
 )  
  
 val spritemapRightFly = resourcesVfs["polet\_vpravo.png"].readBitmap()  
 val playerAnimationRightFly = SpriteAnimation(  
 spriteMap = spritemapRightFly,  
 spriteWidth = 128, //Ширина одного слайда в пикселях  
 spriteHeight = 128, //Высота одного слайда в пикселях  
 marginTop = 0,  
 marginLeft = 0,  
 columns = 2, //Количество столбцов  
 rows = 1, //количество строчек  
 offsetBetweenColumns = 0,  
 offsetBetweenRows = 0  
 )  
  
 //region Игрок  
 lateinit var player: Sprite  
 player = sprite(playerAnimation).size(180, 180).position(0, 0)  
 player.playAnimationLooped(spriteDisplayTime = 500.milliseconds)  
 player.onAnimationCompleted **{** player.playAnimationLooped(playerAnimation, spriteDisplayTime = 500.milliseconds) **}** //endregion  
  
 val testPlatformHitBox = solidRect(700,20, Colors.RED).position(700,600).registerBodyWithFixture(type = BodyType.STATIC)  
 val bottomHit = solidRect(700,20, Colors.TRANSPARENT\_BLACK).position(700,601).registerBodyWithFixture(type = BodyType.STATIC)  
 val leftHit = solidRect(1, 1, Colors.TRANSPARENT\_BLACK).position(700, 610).registerBodyWithFixture(type = BodyType.STATIC)  
  
 var isPlaying = false  
  
 val debugText = text("", textSize = 16.0, alignment = TextAlignment.TOP\_LEFT).position(32.0, 32.0)  
 val jump = resourcesVfs["jump.mp3"].readSound()  
  
 player.addUpdater **{** try {  
 val ntouches: Int = input.activeTouches.size  
 val touches: List<Touch> = input.activeTouches  
  
 var num = sizeAll.width / 4.0  
  
 //region Гравитация  
 if (player.y < sizeAll.height - 200 && !collidesWith(testPlatformHitBox))  
 player.y += 5  
 else if (collidesWith(bottomHit))  
 player.y += 5  
 if (touches.size == 0){  
 player.playAnimation(playerAnimation)  
 }  
 //endregion  
 touches.forEach **{** // Передвижение налево  
 if (it.x in 0.0..num && player.pos.x >= 10.0) {  
 player.x -= 6  
 player.playAnimation(playerAnimationLeft, spriteDisplayTime = 200.milliseconds)  
 }  
 else if (it.x in num..num \* 2 && player.pos.x <= sizeAll.width - 160.0 && !collidesWith(leftHit)) {  
 player.x += 6  
 player.playAnimation(playerAnimationRight, spriteDisplayTime = 200.milliseconds)  
  
 if (player.pos.x == sizeAll.width - 180.0) {  
 launch **{** sceneContainer.changeTo<Scene1>()  
 **}** }  
 }  
 if (it.x in num \* 2..num \* 4 && player.pos.y >= 0 && !collidesWith(bottomHit)) {  
 player.y -= 10  
 if (!isPlaying) {  
 launch **{** jump.play()  
 isPlaying = true  
 delay(10L)  
 isPlaying = false  
 **}** }  
 }  
 debugText.text = player.x.toString()  
 **}** }  
 catch (e: Exception){  
 debugText.text = e.toString()  
 }  
  
 **}**}}  
  
class SceneDialog(): Scene(){  
 override suspend fun Container.sceneInit(){  
 val fon = resourcesVfs["Angar\_New.png"].readBitmap()  
  
 val fon\_image = image(fon).scale(2)  
  
 uiText(" Тестовый текст",  
 (sizeAll.width).toDouble(), //ширина  
 (sizeAll.height).toDouble()) //высота  
 **{** textSize = 75.0  
 position(sizeAll.width-600.0, sizeAll.height - 300.0)  
 **}** }  
}

# **Приложение 2. Код модуля music.**

package music  
  
import com.soywiz.korau.sound.Sound  
import com.soywiz.korau.sound.readSound  
import com.soywiz.korio.file.std.resourcesVfs  
  
public suspend fun getJumpMusic(): Sound {  
 return resourcesVfs["jump.mp3"].readSound()  
}

# **Используемая литература**

<https://ru-wiki.ru/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0>

<https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D0%B9%D0%BC%D0%B5%D1%80>

https://github.com/Dasbat/kyrsovaya

1. 1) ГОСТ 19.101-77 ЕСПД. Виды программ и программных документов [↑](#footnote-ref-1)
2. 2) ГОСТ 19.103-77 ЕСПД. Обозначение программ и программных документов [↑](#footnote-ref-2)
3. 3) ГОСТ 19.104-78\* ЕСПД. Основные надписи [↑](#footnote-ref-3)
4. 4) ГОСТ 19.105-78\* ЕСПД. Общие требования к программным документам [↑](#footnote-ref-4)
5. 5) ГОСТ 19.106-78\* ЕСПД. Общие требования к программным документам, выполненным печатным способом [↑](#footnote-ref-5)
6. 6) ГОСТ 19.505-79\* ЕСПД. Руководство оператора. Требования к содержанию и оформлению [↑](#footnote-ref-6)
7. 7) ГОСТ 19.604-78\* ЕСПД. Правила внесения изменений в программные документы, выполненные печатным способом [↑](#footnote-ref-7)