МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра программной инженерии

ОТЧЕТ

по лабораторной работе №4

по дисциплине: «Качество и тестирование программного обеспечения»

Выполнил: Евдокимов Н.А.

Институт приборостроения, автоматизации и информационных технологий

Направление: 09.03.04 «Программная инженерия»

Группа: 71-ПГ

Проверили: Конюхова О.В., Ужаринский А.Ю.

Отметка о зачете:

Дата: «____» _____ 2020 г.

Орёл, 2020

Тема: Тестирование пользовательского интерфейса.

Задание:

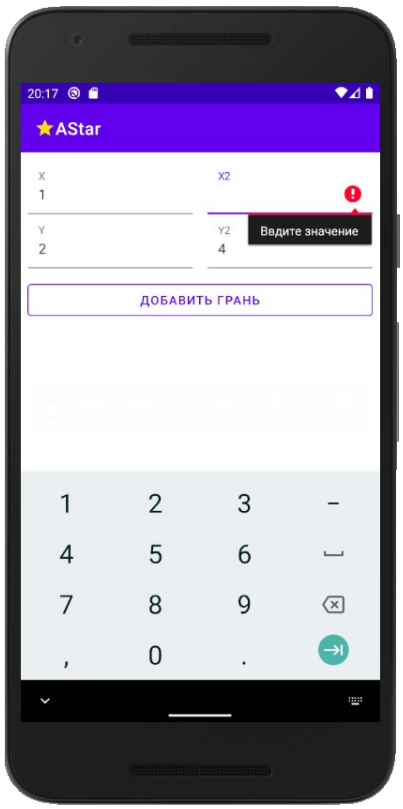
Добавляем графический интерфейс для задания из лабораторной работы 1. Тестируем полученный интерфейс на основе сценариев использования.

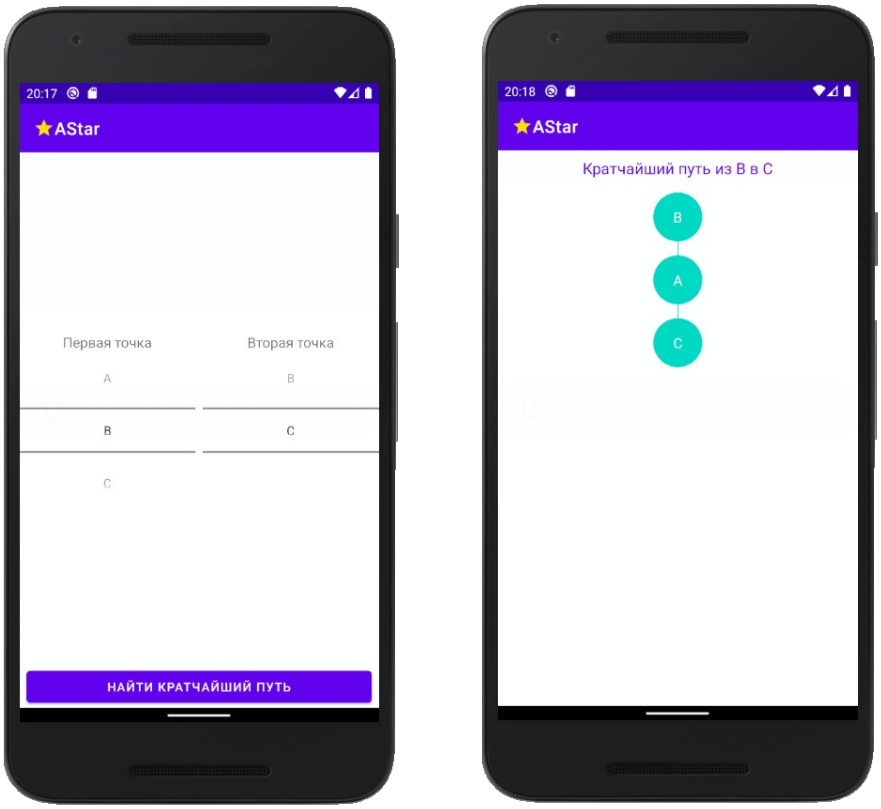
Описание интерфейса:

При старте приложения пользователь видит экран с четыремя текстовыми полями для ввода координат двух связанных вершин, ниже кнопка для добавления грани. После добавления грани она отображается в списке на этом же экране. При попытке добавления уже существующей грани, появляется сообщение об ошибке. Если при попытке добавить грань одно из полей пустует, фокус будет переключен на первое пустое поле, помимо того, будет выведено сообщение об ошибке.

После того, как будет введена по меньшей мере 1 грань появится кнопка «выбрать вершины». После ее нажатия откроется экран выбора вершин для поиска пути. На нем находится два прокручиваемых списка, ниже которых находится кнопка для поиска кратчайшего пути. При нажатии на последнюю открывается экран.

Изображение пользовательского интерфейса:





Исходный код пользовательского интерфейса:

```
class MainActivity : AppCompatActivity() {

    private lateinit var viewModel: GraphViewModel
    private lateinit var edgesRecyclerViewAdapter: GraphEdgesAdapter

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        setSupportActionBar(toolbar)
        supportActionBar?.setIcon(R.drawable.ic_star_yellow_24dp)

        initViewModel()
        setListeners()
        initRecyclerViews()
    }

    private fun initViewModel() {
        val factory = GraphViewModelFactory(application)
```

```
        val thisActivity = this
        viewModel = ViewModelProvider(this,
factory).get(GraphViewModel::class.java)
        viewModel.apply {
            // Init recycler view adapter
            edgesRecyclerViewAdapter = GraphEdgesAdapter(edges)
            rvGraphPoints.adapter = edgesRecyclerViewAdapter

            elementAddedEvent.observe(thisActivity, Observer { elementNumber ->
                elementNumber?.let {
                    edgesRecyclerViewAdapter.notifyItemInserted(elementNumber)
                    rvGraphPoints.scrollToPosition(elementNumber)
                    btnSelectPoints.apply {
                        if (edgesRecyclerViewAdapter.itemCount > 0) {
                            visibility = View.VISIBLE
                        }
                    }

                    elementAddedEvent.postValue(null)
                }
            })

            shortestWayLiveData.observe(thisActivity, Observer { path ->
                kotlin.run {
                    val point1 = viewModel.getPoint(npFirstPoint.value)
                    val point2 = viewModel.getPoint(npSecondPoint.value)

                    tvShortestWayHeader.text = if (path == null) {
                        getString(R.string.way_from_X_to_X_not_found, point1.name,
point2.name)
                    } else {
                        getString(R.string.shortest_way_from_X_to_X, point1.name,
point2.name)
                    }
                }

                path?.let {
                    rvPath.apply {
                        adapter = PathPointsAdapter(path)
                        layoutManager = LinearLayoutManager(thisActivity)
```

```
                }
                rvPath.visibility = View.VISIBLE
            }

            vfRootView.displayedChild = 2
        })

        enteredExistingEdgeLiveData.observe(thisActivity, Observer {
            if (it != null && it) {
                longToast(getString(R.string.already_have_this_edge))
                enteredExistingEdgeLiveData.postValue(null)
            }
        })
    }
}

private fun setListeners() {
    btnAddGraphEdge.setOnClickListener(AddEdgeBtnListener())
    btnSelectPoints.setOnClickListener {
        val initSpinner = { picker: NumberPicker ->
            picker.apply {
                minValue = 0
                maxValue = viewModel.getPointsCount() - 1 // Inclusive
                displayedValues = viewModel.getPointsNames().toTypedArray()
            }
        }

        initSpinner(npFirstPoint)
        initSpinner(npSecondPoint)
        hideKeyboard()
        vfRootView.displayedChild = 1
    }

    btnFindShortestWay.setOnClickListener {
        val point1 = viewModel.getPoint(npFirstPoint.value)
        val point2 = viewModel.getPoint(npSecondPoint.value)
        if (point1 == point2) {
            longToast(getString(R.string.choose_different_points))
            return@setOnClickListener
        }
```

```
        viewModel.findShortestWay(point1, point2)
    }
}

private fun initRecyclerViews() {
    rvGraphPoints.layoutManager = LinearLayoutManager(this)
}

inner class AddEdgeBtnListener : View.OnClickListener {
    private fun validateInputFields(): Boolean {
        var isAllValid = true

        val x1Str = etX1.text.toString()
        val y1Str = etY1.text.toString()
        val x2Str = etX2.text.toString()
        val y2Str = etY2.text.toString()

        // Check if all points are given
        val inputFields = listOf(etX1, etX2, etY1, etY2)

        val emptyFields = inputFields.filter { it?.text.isNullOrEmpty() }
        if (emptyFields.isNotEmpty()) {
            emptyFields.forEach { it.error = getString(R.string.enter_value) }
            isAllValid = false
        }

        // Only distance is empty
        if (emptyFields.isEmpty()) {
            val x1 = x1Str.toInt()
            val y1 = y1Str.toInt()
            val x2 = x2Str.toInt()
            val y2 = y2Str.toInt()

            if (x1 == x2 && y1 == y2) {
                etX1.requestFocus()
                isAllValid = false
                etX1.error = getString(R.string.enter_different_points)
            }
        }
```

```
            try {
                inputFields.first { !it.error.isNullOrEmpty() }.requestFocus()
            } catch (noElementEx: NoSuchElementException) {}

            return isAllValid
        }

        override fun onClick(v: View?) {
            val isAllFieldsValid = validateInputFields()
            if (!isAllFieldsValid) { return }

            val x1 = etX1.text.toString().toInt()
            val y1 = etY1.text.toString().toInt()
            val x2 = etX2.text.toString().toInt()
            val y2 = etY2.text.toString().toInt()

            viewModel.addEdge(x1, y1, x2, y2)
        }
    }
}

class GraphEdgesAdapter(private val graphEdges: List<Edge>)
    : RecyclerView.Adapter<GraphEdgesAdapter.GraphEdgeHolder>() {

    inner class GraphEdgeHolder(view: View) : RecyclerView.ViewHolder(view) {
        private val ctx = view.context

        private val tvCoords1 = view.findViewById<TextView>(R.id.tvCoords1)
        private val tvCoords2 = view.findViewById<TextView>(R.id.tvCoords2)
        private val tvDistance = view.findViewById<TextView>(R.id.tvDistance)

        fun bind(edge: Edge) {
            edge.run {
                point1.run {
                    tvCoords1.text = ctx.getString(R.string.edge_item_coords,
point1.name, x, y)
                }
                point2.run {
                    tvCoords2.text = ctx.getString(R.string.edge_item_coords,
point2.name, x, y)
```

```
                }

                // It's int
                if (distance - distance.toInt() == 0.0) {
                    tvDistance.text =
ctx.getString(R.string.edge_item_distance_int, distance.toInt())
                } else {
                    tvDistance.text =
ctx.getString(R.string.edge_item_distance_double, distance)
                }
            }
        }
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
GraphEdgeHolder {
        val holderView = LayoutInflater.from(parent.context)
            .inflate(R.layout.graph_edge_item, parent, false)

        return GraphEdgeHolder(holderView)
    }

    override fun getItemCount(): Int {
        return graphEdges.size
    }

    override fun onBindViewHolder(holder: GraphEdgeHolder, position: Int) {
        val edge = graphEdges[position]
        holder.bind(edge)
    }
}

class PathPointsAdapter(private val path: List<GraphPoint>)
    : RecyclerView.Adapter<PathPointsAdapter.PathPointViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
PathPointViewHolder {
        val view = LayoutInflater.from(parent.context)
            .inflate(R.layout.path_item, parent, false)
```

```
        return PathPointViewHolder(view)
    }

    override fun getItemCount() = path.size

    override fun onBindViewHolder(holder: PathPointViewHolder, position: Int) {
        val point = path[position]
        val isFirstItem = position == 0
        val isLastItem = position == path.size - 1

        holder.bind(point, isFirstItem, isLastItem)
    }

    inner class PathPointViewHolder(view: View) : RecyclerView.ViewHolder(view) {
        private val topLine = view.findViewById<View>(R.id.topLine)
        private val bottomLine = view.findViewById<View>(R.id.bottomLine)
        private val tvPointLabel = view.findViewById<TextView>(R.id.tvPointLabel)

        fun bind(point: GraphPoint, isFirstItem: Boolean = false, isLastItem:
Boolean = false) {
            if (isFirstItem) {
                topLine.visibility = View.INVISIBLE
            } else {
                topLine.visibility = View.VISIBLE
            }

            if (isLastItem) {
                bottomLine.visibility = View.INVISIBLE
            } else {
                bottomLine.visibility = View.VISIBLE
            }

            tvPointLabel.text = point.name
        }
    }
}
```

Сценарии:

Действие	Состояние	Результат
Клик на «Добавить грань»	Одно или более поле пустое	Переключение фокуса на пустое поле, отображение ошибки
	Введена уже существующая грань	Отображение ошибки
	Введены одинаковые вершины	Отображение ошибки
	Данные корректны	Отображение грани в списке, отображение кнопки «Выбрать вершины»
Клик на «Выбрать вершины»	-	Переключение на экран выбора вершин
Клик на «Найти кратчайший путь»	Выбраны одинаковые вершины	Отображение ошибки
	Выбраны разные вершины, путь есть	Переход на экран кратчайшего пути, отображение пути
	Выбраны разные вершины, пути нет	Переход на экран кратчайшего пути, отображение сообщения об отсутствии пути

Исходный код тестов:

```
class MainActivityTest {
    @Rule
    @JvmField
    public val activityRule = ActivityTestRule(MainActivity::class.java)
```

```
        private fun toastIsShown(text: String) {
            val validator = not(`is`(activityRule.activity.window.decorView))

onView(withText(text)).inRoot(withDecorView(validator)).check(matches(isDisplayed()
))
        }

        private fun isTextInputLayoutHasError(id: Int): Boolean {
            var hasError = false
            onView(withId(id)).perform(object  : ViewAction {
                override fun getDescription() = "Returns if TextInputLayout has
error or not"
                override fun getConstraints() =
isAssignableFrom(TextInputLayout::class.java)

                override fun perform(uiController: UiController?, view: View?) {
                    val til = view as TextInputLayout
                    hasError = !til.error.isNullOrEmpty()
                }

            })

            return hasError
        }

        private fun enterEdge(x1: Int, y1: Int, x2: Int, y2: Int) {
            onView(withId(R.id.etX1)).perform(typeText(x1.toString()))
            onView(withId(R.id.etY1)).perform(typeText(y1.toString()))
            onView(withId(R.id.etX2)).perform(typeText(x2.toString()))
            onView(withId(R.id.etY2)).perform(typeText(y2.toString()))

            onView(withId(R.id.btnAddGraphEdge)).perform(click())
        }

        private fun enterCorrectPoints(numberOfEdges: Int) {
            var i = 0
            val enteredEdges = mutableSetOf<Edge>()
            while (i < numberOfEdges) {
                val x1 = Random.nextInt(100)
                val y1 = Random.nextInt(100)
```

```
                val x2 = x1 - 1
                val y2 = y1 - 1

                val edge = Edge(GraphPoint(x1, y1, ""), GraphPoint(x2, y2, ""))
                if (edge in enteredEdges) {
                    continue
                }
                enteredEdges.add(edge)
                enterEdge(x1, y1, x2, y2)

                i++
            }
        }

        private fun viewFlipperDisplayedChildIndex(flipperId: Int): Int {
            var displayedChildIndex = -1
            onView(withId(flipperId)).perform(object  : ViewAction {
                override fun getDescription() = "Returns ViewFlipper current
displayed child"
                override fun getConstraints() =
isAssignableFrom(ViewFlipper::class.java)

                override fun perform(uiController: UiController?, view: View?) {
                    val flipper = view as ViewFlipper
                    displayedChildIndex = flipper.displayedChild
                }

            })

            return displayedChildIndex
        }

        @Test
        fun addEdgeClick_oneEmptyField() {
            onView(withId(R.id.etX1)).perform(typeText("1"))
            onView(withId(R.id.etY1)).perform(typeText("1"))
            onView(withId(R.id.etX2)).perform(typeText("1"))

            onView(withId(R.id.btnAddGraphEdge)).perform(click())
            val isY2HasError = isTextInputLayoutHasError(R.id.tilY2)
```

```
            assert(isY2HasError)
        }

        @Test
        fun addEdgeClick_twoSimilarPointsAreEntered() {
            onView(withId(R.id.etX1)).perform(typeText("1"))
            onView(withId(R.id.etY1)).perform(typeText("1"))
            onView(withId(R.id.etX2)).perform(typeText("1"))
            onView(withId(R.id.etY2)).perform(typeText("1"))

            onView(withId(R.id.btnAddGraphEdge)).perform(click())

            val isX1HasError = isTextInputLayoutHasError(R.id.tilX1)
            assert(isX1HasError)
        }

        @Test
        fun addEdgeClick_twoSimilarEdgesAreEntered() {
            enterCorrectPoints(1)

            onView(withId(R.id.btnAddGraphEdge)).perform(click())
            onView(withId(R.id.btnAddGraphEdge)).perform(click())


toastIsShown(activityRule.activity.getString(R.string.already_have_this_edge))
        }

        @Test
        fun addEdgeClick_dataCorrect() {
            enterCorrectPoints(1)

            onView(withId(R.id.btnAddGraphEdge)).perform(click())

            onView(withId(R.id.btnSelectPoints)).check(matches(isDisplayed()))
        }

        @Test
        fun selectPointsClicked() {
            enterCorrectPoints(1)
```

```
            onView(withId(R.id.btnAddGraphEdge)).perform(click())
            onView(withId(R.id.btnSelectPoints)).perform(click())

            val displayedChildIndex =
viewFlipperDisplayedChildIndex(R.id.vfRootView)
            Assert.assertEquals(1, displayedChildIndex)
        }

        @Test
        fun findShortestWayClicked_similarPointsSelected() {
            enterCorrectPoints(1)

            onView(withId(R.id.btnAddGraphEdge)).perform(click())
            onView(withId(R.id.btnSelectPoints)).perform(click())
            onView(withId(R.id.btnFindShortestWay)).perform(click())


toastIsShown(activityRule.activity.getString(R.string.choose_different_points))
        }

        @Test
        fun findShortestWayClicked_wayFound() {
            enterCorrectPoints(1)

            onView(withId(R.id.btnAddGraphEdge)).perform(click())
            onView(withId(R.id.btnSelectPoints)).perform(click())


onView(withId(R.id.npSecondPoint)).perform(setNumberPickerValueByIndex(1))
            onView(withId(R.id.btnFindShortestWay)).perform(click())

            val displayedChildIndex =
viewFlipperDisplayedChildIndex(R.id.vfRootView)
            Assert.assertEquals(2, displayedChildIndex)
            onView(withId(R.id.rvPath)).check(matches(isDisplayed()))
        }

        @Test
        fun findShortestWayClicked_noWay() {
```

```
            enterEdge(0, 0, 1, 1)
            enterEdge(2, 2, 3, 3)

            onView(withId(R.id.btnSelectPoints)).perform(click())


onView(withId(R.id.npSecondPoint)).perform(setNumberPickerValueByIndex(2))
            onView(withId(R.id.btnFindShortestWay)).perform(click())

            val displayedChildIndex =
viewFlipperDisplayedChildIndex(R.id.vfRootView)
            Assert.assertEquals(2, displayedChildIndex)
            onView(withId(R.id.rvPath)).check(matches(not(isDisplayed())))
        }

        inner class setNumberPickerValueByIndex(private val index: Int) :
ViewAction {
            override fun getDescription() = "Sets NumberPicker to certain value"
            override fun getConstraints(): Matcher<View> {
                return isAssignableFrom(NumberPicker::class.java)
            }

            override fun perform(uiController: UiController?, view: View?) {
                val np = view as NumberPicker
                np.value = index
            }
        }
    }
```