МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»
КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ
(КАФЕДРА №43)

КУРСОВОЙ ПРОЕКТ
ЗАЩИЩЁН С ОЦЕНКОЙ

старший преподаватель		П. А. Степанов
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

по дисциплине: УПРАВЛЕНИЕ КАЧЕСТВОМ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

РАБОТУ ВЫПОЛНИЛ
СТУДЕНТ ГР.

4631		Д. С. Воробьев
№ группы	подпись, дата	инициалы, фамилия

Санкт-Петербург

2019 г.

Содержание

1. Задание

1. Взять свою лабораторную работу по этому курсу и создать для нее репозиторий. Выложить на Github.
2. Разработать план выпуска новой версии (только содержательную часть).
3. Оформить работы по выпуску новой версии в виде списка issues
4. Создать скелет функционала новой программы (классы и экранные формы).
5. Создать тесты на новый функционал. Отделить тесты на реализованный функционал от нереализованных.
6. Создать план по тестированию (артефакт согласно PMBOK)
7. Настроить систему непрерывной интеграции. Сделать так, чтобы тесты на реализованный функционал выполнялись в ответ на интеграцию.
8. Создать отчет по курсовому проектировании.

2. Выполнение

Язык выполнения лабораторной работы: Kotlin. Был создан репозиторий по адресу: `https://github.com/phdeh/a`. В него была загружена лабораторная работа, реализующая поиск пути в неориентированном графе в плоской декартовой системе координат методом A^*. На вход подаются две вершины, на выходе — путь между вершинами.

Все задачи были оформлены в виде списка Issues.

1. Добавить возможность найти полный список путей.
2. Добавить тесты на реализованный функционал.
3. Добавить тесты на нереализованный функционал.
4. Добавить оконный интерфейс для удобства пользовательского ввода.

3. Пример решения задачи

Задача 1. Добавить возможность пользователю вводить целые числа.

Реализация: Была добавлена одна новая функция `findAllPaths(from, to)`, вызывающая алгоритм поиска кратчайшего пути при наличии эвристического расстояния до цели A^*, получающая из него минимальный путь и находящая все прочие пути, не превышающие длины найденного с помощью A^*.

Листинг 1 – `main/org/phdeh/a/astar/AStar.kt`

```kotlin
package org.phdeh.a.astar

import java.lang.RuntimeException
import java.util.*

object AStar {
    class VerticeBelongToDifferentGraphsException(msg: String) : RuntimeException(msg)
    class VertexDoesntExistException(msg: String) : RuntimeException(msg)

    fun findPath(from: Graph.Vertex, to: Graph.Vertex): List<Graph.Vertex> {
        if (from === Graph.NON_EXISTENT_VERTEX && to === Graph.NON_EXISTENT_VERTEX)
```

```
            throw VertexDoesntExistException("both from and to")
        if (from === Graph.NON_EXISTENT_VERTEX)
            throw VertexDoesntExistException("from")
        if (to === Graph.NON_EXISTENT_VERTEX)
            throw VertexDoesntExistException("to")
        if (from.graph != to.graph)
            throw VerticeBelongToDifferentGraphsException("$from and $to")
        if (from == to)
            return listOf(to)
        val graph = from.graph

        val frontier = PriorityQueue(from)
        val cameFrom = Array<Graph.Vertex?>(graph.size) { null }
        val handled = BitSet(graph.size)
        val costSoFar = DoubleArray(graph.size) { 0.0 }

        handled[from.ordinal] = true

        fun directDist(v: Graph.Vertex) = Math.sqrt(
            Math.pow((to.y - v.y), 2.0) +
                    Math.pow((to.x - v.x), 2.0)
        )

        var last = from
        while (frontier.isNotEmpty()) {
            val current = frontier.pop()
            last = current
            if (current == to)
                break

            current.edgesTo.forEach {
                val newCost = costSoFar[current.ordinal] + it.distanceTo(current)
                if (!handled[it.ordinal] || newCost < costSoFar[it.ordinal]) {
                    handled[it.ordinal] = true
                    costSoFar[it.ordinal] = newCost
                    cameFrom[it.ordinal] = current
                    val priority = newCost + directDist(it)
                    frontier.push(it, priority)
                }
            }
        }

        if (last == to) {
            val ml = mutableListOf<Graph.Vertex>()
            var curr: Graph.Vertex? = last
            while (curr != null) {
                ml.add(curr)
                curr = cameFrom[curr.ordinal]
            }
            return ml.reversed()
        }
        return listOf()
    }

    fun findAllPaths(from: Graph.Vertex, to: Graph.Vertex): List<List<Graph.Vertex>> {
        val paths = mutableSetOf(findPath(from, to))
        if (from != to && paths.isNotEmpty()) {
            var minLength = 0.0
            paths.first().forEachPair { a, b ->
                minLength += a distanceTo b
            }
            fun depthFirst(current: Graph.Vertex, visited: List<Graph.Vertex>, length:
            ↪  Double) {
                if (current == to && length <= minLength + 0.001)
                    paths += visited
                if (length <= minLength + 0.001)
                    current.edgesTo.forEach {
                        if (it !in visited)
                            depthFirst(it, visited + it, length + current.distanceTo(
                            ↪  it))
                    }
            }
            depthFirst(from, listOf(from), 0.0)
        }
        return paths.toList()
    }
}
```

Листинг 1 – main/org/phdeh/a/astar/AStar.kt

4. План по тестированию

Таблица 1 – План по тестированию

Название задачи	Длительность	Ответственный	Объект проверки
Добавить возможность найти полный список путей.	1 день	Д. С. Воробьев	Программа
Добавить тесты на реализованный функционал.	1 день	Д. С. Воробьев	Программа
Добавить тесты на нереализованный функционал.	1 день	Д. С. Воробьев	Программа
Добавить оконный интерфейс для удобства пользовательского ввода.	1 день	Д. С. Воробьев	Программа

5. Система непрерывной интеграции

Для реализации непрерывной интеграции была подключена система Travis CI. В корень репозитория был помещён файл с конфигурацией.

Текст файла конфигурации:

Листинг 2 – .travis.yml

```yaml
language: java
install: true

jdk:
  - oraclejdk8

script:
  - ./gradlew wrapper --gradle-version=5.4.1 --distribution-type=bin
  - ./gradlew test
```

Листинг 2 – .travis.yml

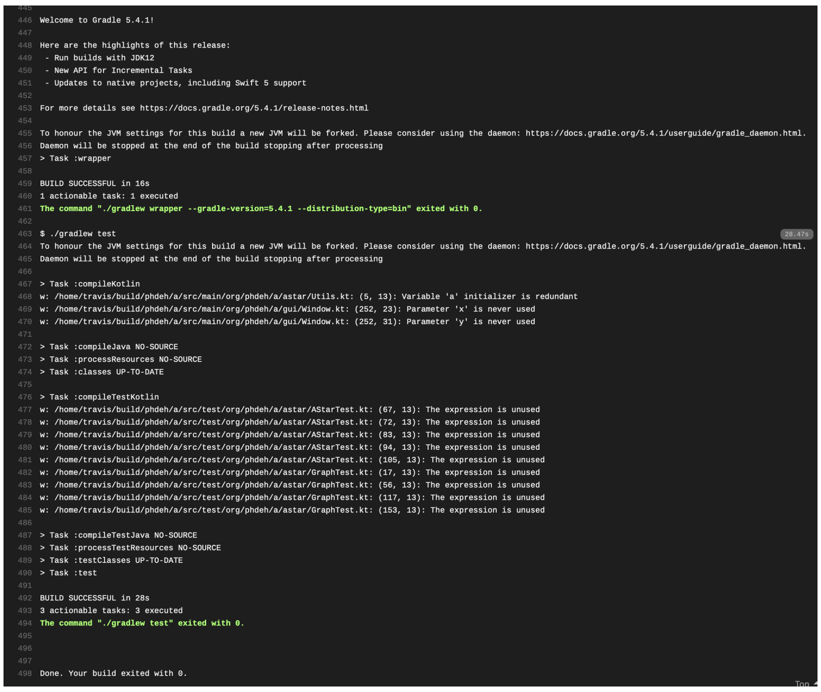
Текст конфигурации Gradle:

Листинг 3 – build.gradle

```groovy
buildscript {
    repositories {
        mavenCentral()
    }

    dependencies {
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:1.3.31"
    }
}

plugins {
    id 'java'
    id 'org.jetbrains.kotlin.jvm' version '1.3.21'
}

group 'AStarCW'
version '1.0-SNAPSHOT'

sourceCompatibility = 1.8

repositories {
    mavenCentral()
}

dependencies {
    compile "org.jetbrains.kotlin:kotlin-stdlib:1.3.31"
    compile 'junit:junit:4.12'
    testCompile 'junit:junit:4.12'
    testImplementation "org.jetbrains.kotlin:kotlin-stdlib-jdk8"
}

compileKotlin {
    kotlinOptions.jvmTarget = "1.8"
}
compileTestKotlin {
    kotlinOptions.jvmTarget = "1.8"
}

```

```
sourceSets {
    main.java.srcDirs += 'src/main'
    main.kotlin.srcDirs += 'src/main'
    test.java.srcDirs += 'src/test'
    test.kotlin.srcDirs += 'src/test'
}


test {
    useJUnit()

    maxHeapSize = '1G'
}
```

Листинг 3 – build.gradle



```
Welcome to Gradle 5.4.1!

Here are the highlights of this release:
 - Run builds with JDK12
 - New API for Incremental Tasks
 - Updates to native projects, including Swift 5 support

For more details see https://docs.gradle.org/5.4.1/release-notes.html

To honour the JVM settings for this build a new JVM will be forked. Please consider using the daemon: https://docs.gradle.org/5.4.1/userguide/gradle_daemon.html.
Daemon will be stopped at the end of the build stopping after processing
> Task :wrapper

BUILD SUCCESSFUL in 16s
1 actionable task: 1 executed
The command "./gradlew wrapper --gradle-version=5.4.1 --distribution-type=bin" exited with 0.

$ ./gradlew test                                                                                                                                  28.47s
To honour the JVM settings for this build a new JVM will be forked. Please consider using the daemon: https://docs.gradle.org/5.4.1/userguide/gradle_daemon.html.
Daemon will be stopped at the end of the build stopping after processing

> Task :compileKotlin
w: /home/travis/build/phdeh/a/src/main/org/phdeh/a/astar/Utils.kt: (5, 13): Variable 'a' initializer is redundant
w: /home/travis/build/phdeh/a/src/main/org/phdeh/a/gui/Window.kt: (252, 23): Parameter 'x' is never used
w: /home/travis/build/phdeh/a/src/main/org/phdeh/a/gui/Window.kt: (252, 31): Parameter 'y' is never used

> Task :compileJava NO-SOURCE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE

> Task :compileTestKotlin
w: /home/travis/build/phdeh/a/src/test/org/phdeh/a/astar/AStarTest.kt: (67, 13): The expression is unused
w: /home/travis/build/phdeh/a/src/test/org/phdeh/a/astar/AStarTest.kt: (72, 13): The expression is unused
w: /home/travis/build/phdeh/a/src/test/org/phdeh/a/astar/AStarTest.kt: (83, 13): The expression is unused
w: /home/travis/build/phdeh/a/src/test/org/phdeh/a/astar/AStarTest.kt: (94, 13): The expression is unused
w: /home/travis/build/phdeh/a/src/test/org/phdeh/a/astar/AStarTest.kt: (105, 13): The expression is unused
w: /home/travis/build/phdeh/a/src/test/org/phdeh/a/astar/GraphTest.kt: (17, 13): The expression is unused
w: /home/travis/build/phdeh/a/src/test/org/phdeh/a/astar/GraphTest.kt: (56, 13): The expression is unused
w: /home/travis/build/phdeh/a/src/test/org/phdeh/a/astar/GraphTest.kt: (117, 13): The expression is unused
w: /home/travis/build/phdeh/a/src/test/org/phdeh/a/astar/GraphTest.kt: (153, 13): The expression is unused

> Task :compileTestJava NO-SOURCE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test

BUILD SUCCESSFUL in 28s
3 actionable tasks: 3 executed
The command "./gradlew test" exited with 0.



Done. Your build exited with 0.
```

Рисунок 1 – Коммит удачно проходит тесты на Travis CI

Заключение

В ходе выполнения курсового проекта был разработан план выпуска новой версии. Работа по выпуску новой версии была оформлена в виде списка Issues. Разработаны тесты на новый функционал. Также была настроена система непрерывной интеграции так, чтобы тесты на реализованный функционал выполнялись в ответ на интеграцию.

Приложение

Исходный код программы

Листинг 4 – main/org/phdeh/a/astar/AStar.kt

```
package org.phdeh.a.astar

import java.lang.RuntimeException
import java.util.*

object AStar {
    class VerticeBelongToDifferentGraphsException(msg: String) : RuntimeException(msg)
    class VertexDoesntExistException(msg: String) : RuntimeException(msg)

    fun findPath(from: Graph.Vertex, to: Graph.Vertex): List<Graph.Vertex> {
        if (from === Graph.NON_EXISTENT_VERTEX && to === Graph.NON_EXISTENT_VERTEX)
            throw VertexDoesntExistException("both from and to")
        if (from === Graph.NON_EXISTENT_VERTEX)
            throw VertexDoesntExistException("from")
        if (to === Graph.NON_EXISTENT_VERTEX)
            throw VertexDoesntExistException("to")
        if (from.graph != to.graph)
            throw VerticeBelongToDifferentGraphsException("$from and $to")
        if (from == to)
            return listOf(to)
        val graph = from.graph

        val frontier = PriorityQueue(from)
        val cameFrom = Array<Graph.Vertex?>(graph.size) { null }
        val handled = BitSet(graph.size)
        val costSoFar = DoubleArray(graph.size) { 0.0 }

        handled[from.ordinal] = true

        fun directDist(v: Graph.Vertex) = Math.sqrt(
            Math.pow((to.y - v.y), 2.0) +
                    Math.pow((to.x - v.x), 2.0)
        )

        var last = from
        while (frontier.isNotEmpty()) {
            val current = frontier.pop()
            last = current
            if (current == to)
                break

            current.edgesTo.forEach {
                val newCost = costSoFar[current.ordinal] + it.distanceTo(current)
                if (!handled[it.ordinal] || newCost < costSoFar[it.ordinal]) {
                    handled[it.ordinal] = true
                    costSoFar[it.ordinal] = newCost
                    cameFrom[it.ordinal] = current
                    val priority = newCost + directDist(it)
                    frontier.push(it, priority)
                }
            }
        }

```

```kotlin
        if (last == to) {
            val ml = mutableListOf<Graph.Vertex>()
            var curr: Graph.Vertex? = last
            while (curr != null) {
                ml.add(curr)
                curr = cameFrom[curr.ordinal]
            }
            return ml.reversed()
        }
        return listOf()
    }

    fun findAllPaths(from: Graph.Vertex, to: Graph.Vertex): List<List<Graph.Vertex>> {
        val paths = mutableSetOf(findPath(from, to))
        if (from != to && paths.isNotEmpty()) {
            var minLength = 0.0
            paths.first().forEachPair { a, b ->
                minLength += a distanceTo b
            }
            fun depthFirst(current: Graph.Vertex, visited: List<Graph.Vertex>, length:
            ↪  Double) {
                if (current == to && length <= minLength + 0.001)
                    paths += visited
                if (length <= minLength + 0.001)
                    current.edgesTo.forEach {
                        if (it !in visited)
                            depthFirst(it, visited + it, length + current.distanceTo(
                            ↪  it))
                    }
            }
            depthFirst(from, listOf(from), 0.0)
        }
        return paths.toList()
    }
}
```

Листинг 4 – main/org/phdeh/a/astar/AStar.kt

Листинг 5 – main/org/phdeh/a/astar/Graph.kt

```kotlin
package org.phdeh.a.astar

import kotlin.reflect.KProperty

class Graph private constructor(builder: GraphBuilder) {

    companion object {
        fun build(action: (GraphBuilder) -> Unit): Graph {
            val builder = GraphBuilder()
            action(builder)
            return Graph(builder)
        }

        val NON_EXISTENT_GRAPH = build { }
        val NON_EXISTENT_VERTEX = Vertex(
            NON_EXISTENT_GRAPH,
            "NON_EXISTENT_GRAPH",
            Double.NaN,
            Double.NaN,
            -1,
            mutableListOf()
        )
    }

    private val vertices: Map<String, Vertex>
    private val verticesIndexed: List<Vertex>

    operator fun get(name: String) = vertices[name] ?: NON_EXISTENT_VERTEX

    operator fun get(index: Int) = if (index in this) verticesIndexed[index] else
    ↪  NON_EXISTENT_VERTEX

    val size get() = vertices.size

    operator fun contains(name: String) = vertices.containsKey(name)

    operator fun contains(vertex: Vertex) = vertices.containsValue(vertex)

    operator fun contains(index: Int) = index in 0..(vertices.size - 1)

    override fun equals(other: Any?): Boolean {
        return other != null &&
                other is Graph &&
                this !== NON_EXISTENT_GRAPH &&
                other !== NON_EXISTENT_GRAPH &&
                this === other
    }

    override fun hashCode(): Int = vertices.hashCode()

    init {
        synchronized(builder) {
            val vb = List(builder.vertices.size) { builder.vertices[it].vertexBuilder
            ↪  }
            val relations = List(vb.size) { mutableListOf<Vertex>() }
            val names = mutableSetOf<String>()
            val verticesList = List(vb.size) {
                val b = vb[it]
                if (b.name !in names) {
                    names += b.name
                    Vertex(this, b.name, b.x, b.y, it, relations[it])
                } else {
                    var i = 2
                    while ("${b.name}_$i" in names)
                        i++
                    names += "${b.name}_$i"
                    Vertex(this, "${b.name}_$i", b.x, b.y, it, relations[it])
                }
            }
            relations.forEachIndexed { it, list ->
                vb[it].currentlyEdgesTo.forEach { e ->
                    list += verticesList[vb.indexOf(e)]
                }
            }
            val verticesMap = mutableMapOf<String, Vertex>()
            verticesList.forEach {
                verticesMap[it.name] = it
            }
            this.verticesIndexed = verticesList
            this.vertices = verticesMap
        }
    }

    data class Vertex internal constructor(
        val graph: Graph,
        val name: String,
        val x: Double,
        val y: Double,
        val ordinal: Int,
        private val hasEdgesTo: MutableList<Vertex>
    ) {
        val edgesTo by lazy { List(hasEdgesTo.size, { hasEdgesTo[it] }).toSet() }

        infix fun hasEdgeTo(other: Vertex?) = this !== NON_EXISTENT_VERTEX &&
                other != null &&
                other !== NON_EXISTENT_VERTEX &&
                other in edgesTo

        override fun equals(other: Any?): Boolean {
            return this !== NON_EXISTENT_VERTEX && other !== NON_EXISTENT_VERTEX &&
            ↪  super.equals(other)
        }

        override fun hashCode(): Int {
            return name.hashCode() xor x.hashCode() xor y.hashCode()
        }

        infix fun distanceTo(other: Vertex): Double =
            if (this === NON_EXISTENT_VERTEX || other === NON_EXISTENT_VERTEX)
                Double.NaN
            else
                Math.sqrt((x - other.x) * (x - other.x) + (y - other.y) * (y - other.y
                ↪  ))

        override fun toString(): String = name
    }

    class GraphBuilder {

        internal val vertices = mutableListOf<VertexDelegate>()

        operator fun invoke(x: Double, y: Double): VertexDelegate {
            val vd = VertexDelegate(null, x, y)
            vertices += vd
            return vd
        }

        operator fun invoke(name: String, x: Double, y: Double): VertexDelegate {
            val vd = VertexDelegate(name, x, y)
            vertices += vd
            return vd
        }

        class VertexBuilder(val x: Double, val y: Double) {
            private var actualName: String? = null
            var name
                get() = actualName ?: "UNDEFINED"
                set(value) {
                    if (actualName == null)
                        actualName = value
                }

            private val actuallyEdgesTo = mutableSetOf<VertexBuilder>()

            val currentlyEdgesTo get() = actuallyEdgesTo.toSet()

            operator fun minus(other: VertexBuilder): VertexBuilder {
                hasEdgeTo(other)
                return other
            }

            fun hasEdgeTo(other: VertexBuilder) {
                synchronized(this) {
                    actuallyEdgesTo += other
                }
                synchronized(other) {
                    other.actuallyEdgesTo += this
                }
            }

        }

        class VertexDelegate(name: String?, x: Double, y: Double) {
            val vertexBuilder = VertexBuilder(x, y).let {
                if (name != null)
                    it.name = name
                it
            }

            operator fun getValue(thisRef: Any?, property: KProperty<*>):
            ↪   VertexBuilder {
                vertexBuilder.name = property.name
                return vertexBuilder
            }
        }
    }

}
```

Листинг 5 – main/org/phdeh/a/astar/Graph.kt

Листинг 6 – main/org/phdeh/a/astar/PriorityQueue.kt

```kotlin
package org.phdeh.a.astar

import java.lang.RuntimeException

class PriorityQueue<T>(vararg ts: T) {
    class EmptyQueueException : RuntimeException()
    private val mp = mutableListOf<Double>()
    private val mt = mutableListOf<T>()

    init {
        ts.forEach {
            push(it, 0.0)
        }
    }

    fun push(t: T, priority: Double) {
        synchronized(this) {
            mp.add(priority)
            mt.add(t)
        }
    }

    fun isEmpty() = mp.isEmpty()
    fun isNotEmpty() = !isEmpty()
    fun pop(): T {
        if (isEmpty())
            throw EmptyQueueException()
        synchronized(this) {
            var max = mp[0]
            var index = 0
            var i = 0
            do {
                if (mp[i] < max) {
                    index = i
                    max = mp[i]
                }
            } while (++i < mp.size)
            mp.removeAt(index)
            val result = mt[index]
            mt.removeAt(index)
            return result
        }
    }
}
```

Листинг 6 – `main/org/phdeh/a/astar/PriorityQueue.kt`

Листинг 7 – `main/org/phdeh/a/astar/Utils.kt`

```kotlin
package org.phdeh.a.astar


fun<K,T : Iterable<K>> T.forEachPair(action: (a: K, b: K) -> Unit) {
    var a = null as K?
    var b = null as K?
    val iter = this.iterator()
    while (iter.hasNext()) {
        a = b
        b = iter.next()
        if (a != null && b != null)
            action(a, b)
    }
}
```

Листинг 7 – `main/org/phdeh/a/astar/Utils.kt`

Листинг 8 – `main/org/phdeh/a/gui/Edge.kt`

```kotlin
package org.phdeh.a.gui

import java.awt.BasicStroke
import java.awt.Color
import java.awt.Graphics2D

data class Edge(
    val from: Node,
    var to: Node,
    val win: Window
) : ScreenObject {
    var shortest = false

    override fun isVisible(r: Screen): Boolean {
        val x = (from.x + to.x) / 2
        val y = (from.y + to.y) / 2
        val width = Math.abs(from.x - to.x)
        val height = Math.abs(from.y - to.y)
        return Math.abs(x - r.x) < (r.width + width) / 2 &&
                Math.abs(y - r.y) < (r.height + height) / 2
    }

    override fun draw(g: Graphics2D, r: Screen) {
        val dx = (-r.width / 2 + r.x).toInt()
        val dy = (-r.height / 2 + r.y).toInt()
        if (shortest)
            g.stroke = BasicStroke(6f)
        else
            g.stroke = BasicStroke(3f)
        g.color = Color.BLACK
        g.drawLine(
            from.x.toInt() - dx,
            from.y.toInt() - dy,
            to.x.toInt() - dx,
            to.y.toInt() - dy
        )

        if (shortest) {
            g.stroke = BasicStroke(3f)
            g.color = Color.GREEN
            g.drawLine(
                from.x.toInt() - dx,
                from.y.toInt() - dy,
                to.x.toInt() - dx,
                to.y.toInt() - dy
            )
        }
    }
}
```

Листинг 8 – main/org/phdeh/a/gui/Edge.kt

Листинг 9 – main/org/phdeh/a/gui/Keyboard.kt

```
package org.phdeh.a.gui

import java.awt.event.KeyEvent
import java.awt.event.KeyListener
import java.util.*

class Keyboard(
    private val window: Window,
    private val action: (Keyboard) -> Unit
) {
    private val keys = BitSet(256)
    private val pkeys = BitSet(256)
    private var _typed = '\u0000'
    val typed get() = _typed

    operator fun get(keyCode: Int) =
        if (keyCode in 0..255)
```

```kotlin
            keys[keyCode]
        else false

    fun pressed(keyCode: Int) =
        if (keyCode in 0..255)
            keys[keyCode] && !pkeys[keyCode]
        else false

    fun released(keyCode: Int) =
        if (keyCode in 0..255)
            !keys[keyCode] && pkeys[keyCode]
        else false

    val command
        get() = this[KeyEvent.VK_CONTROL] ||
                this[KeyEvent.VK_ALT] ||
                this[KeyEvent.VK_META]

    val handle
        get() = this[KeyEvent.VK_SPACE]

    val remove
        get() = this[KeyEvent.VK_BACK_SPACE] ||
                this[KeyEvent.VK_DELETE]

    init {
        fun handle() {
            for (i in 0..255)
                pkeys[i] = keys[i]
        }

        window.addKeyListener(object : KeyListener {
            override fun keyPressed(e: KeyEvent) {
                handle()
                keys[e.keyCode] = true
                action(this@Keyboard)
            }

            override fun keyReleased(e: KeyEvent) {
                handle()
                keys[e.keyCode] = false
                action(this@Keyboard)
            }

            override fun keyTyped(e: KeyEvent) {
                _typed = e.keyChar
                action(this@Keyboard)
            }
        })
    }
}
```

Листинг 9 – main/org/phdeh/a/gui/Keyboard.kt

Листинг 10 – main/org/phdeh/a/gui/Mouse.kt

```kotlin
package org.phdeh.a.gui

import java.awt.event.*

class Mouse(
    private val window: Window,
    private val action: (Mouse) -> Unit
) {
    private var _x = 0
    private var _y = 0
    private var _dx = 0
    private var _dy = 0
    private var _lb = false
    private var _rb = false
    private var _mb = false
    private var _lb_r = false
    private var _rb_r = false
    private var _mb_r = false
    private var _lb_p = false
    private var _rb_p = false
    private var _mb_p = false
    private var _hw = 0
    private var _vw = 0

    val x get() = _x
    val y get() = _y
    val dx get() = _dx
    val dy get() = _dy
    val leftButton get() = _lb
    val rightButton get() = _rb
    val middleButton get() = _mb
    val leftButtonReleased get() = _lb_r
    val rightButtonReleased get() = _rb_r
    val middleButtonReleased get() = _mb_r
    val leftButtonPressed get() = _lb_p
    val rightButtonPressed get() = _rb_p
    val middleButtonPressed get() = _mb_p
    val horisontalWheel get() = _hw
    val verticalWheel get() = _vw

    init {
        fun handle(e: MouseEvent, mb: Boolean? = null) {
            val px = _x
            val py = _y
            _x = e.x
            _y = e.y
            _dx = _x - px
            _dy = _y - py

            val plb = _lb
            val prb = _rb
            val pmb = _mb

            if (mb != null)
                when (e.button) {
                    MouseEvent.BUTTON1 -> {
                        _lb = mb
                    }
                    MouseEvent.BUTTON3 -> {
                        _rb = mb
                    }
                    MouseEvent.BUTTON2 -> {
                        _mb = mb
                    }
                }

            _lb_p = !plb && _lb
            _rb_p = !prb && _rb
            _mb_p = !pmb && _mb

            _lb_r = plb && !_lb
            _rb_r = prb && !_rb
            _mb_r = pmb && !_mb

            _hw = 0
            _vw = 0
        }

        val ml = object : MouseListener {
            override fun mouseClicked(e: MouseEvent) {
                handle(e, null)
                action(this@Mouse)
            }

            override fun mouseEntered(e: MouseEvent) {
                handle(e, null)
                action(this@Mouse)
            }

            override fun mouseExited(e: MouseEvent) {
                handle(e, null)
                action(this@Mouse)
            }

            override fun mousePressed(e: MouseEvent) {
                handle(e, true)
                action(this@Mouse)
            }

            override fun mouseReleased(e: MouseEvent) {
                handle(e, false)
                action(this@Mouse)
            }
        }
        val mml = object : MouseMotionListener {
            override fun mouseDragged(e: MouseEvent) {
                handle(e, true)
                action(this@Mouse)
            }

            override fun mouseMoved(e: MouseEvent) {
                handle(e, false)
                action(this@Mouse)
            }
        }
        val mwl = object : MouseWheelListener {
            override fun mouseWheelMoved(e: MouseWheelEvent) {
                handle(e, null)
                if (e.isShiftDown)
                    _hw = e.wheelRotation
                else
```

```
                      _vw = e.wheelRotation
                  action(this@Mouse)
              }
          }
          window.addMouseListener(ml)
          window.addMouseMotionListener(mml)
          window.addMouseWheelListener(mwl)
      }
  }
```

Листинг 10 – main/org/phdeh/a/gui/Mouse.kt

Листинг 11 – main/org/phdeh/a/gui/NameLobby.kt

```
package org.phdeh.a.gui

import java.util.*

class NameLobby {
    private var letter = 0
    private var number = 0
    private val stack = Stack<String>()

    fun getName(): String {
        if (stack.isNotEmpty())
            return stack.pop()
        val cl = (letter + 'A'.toInt()).toChar()
        val name = if (number == 0)
            "$cl"
        else
            "$cl$number"

        if (cl == 'Z') {
            letter = 0
            number++
        } else
            letter++

        return name
    }

    fun returnName(name: String) {
        stack.push(name)
    }
}
```

Листинг 11 – main/org/phdeh/a/gui/NameLobby.kt

Листинг 12 – main/org/phdeh/a/gui/Node.kt

```
package org.phdeh.a.gui

import java.awt.Color
import java.awt.Graphics2D

data class Node(
    var name: String,
    var x: Double,
    var y: Double,
    val win: Window,
    var tx: Double = 0.0,
    var ty: Double = 0.0
) : ScreenObject {
```

```kotlin
    val width = 50
    val height = 50

    override fun isVisible(r: Screen): Boolean {
        return Math.abs(x - r.x) < (r.width + width) / 2 &&
                Math.abs(y - r.y) < (r.height + height) / 2
    }

    override fun draw(g: Graphics2D, r: Screen) {
        tx = Math.round(x / win.squareSize).toDouble() * win.squareSize
        ty = Math.round(y / win.squareSize).toDouble() * win.squareSize

        val alpha = Math.max(160 - (Math.sqrt(
            (tx - x) * (tx - x) + (ty - y) * (ty - y)
        ) / win.squareSize * 255).toInt(), 0)
        g.color = Color(0, 0, 0, alpha)
        g.fillOval(
            (tx - r.x).toInt() - (width - r.width) / 2,
            (ty - r.y).toInt() - (height - r.height) / 2,
            width,
            height
        )
        if (win.lastTouch === this)
            g.color = Color.RED
        else if (win.prevTouch === this)
            g.color = Color.RED
        else
            g.color = Color.BLUE
        g.fillOval(
            (x - r.x).toInt() - (width - r.width) / 2,
            (y - r.y).toInt() - (height - r.height) / 2,
            width,
            height
        )
        g.color = Color.WHITE
        g.drawString(
            name,
            (x - r.x).toInt() + (r.width) / 2
                    - g.getFontMetrics().stringWidth(name) / 2,
            (y - r.y).toInt() + (r.height + 28) / 2
        )
    }
}
```

Листинг 12 – main/org/phdeh/a/gui/Node.kt

Листинг 13 – main/org/phdeh/a/gui/Screen.kt

```kotlin
package org.phdeh.a.gui

data class Screen(
    private val camera: Vector,
    private val window: Window
) {
    val x get() = camera.x
    val y get() = camera.y

    val width get() = window.width
    val height get() = window.height
}
```

Листинг 13 – main/org/phdeh/a/gui/Screen.kt

Листинг 14 – main/org/phdeh/a/gui/ScreenObject.kt

```kotlin
package org.phdeh.a.gui

import java.awt.Graphics2D
import java.awt.Rectangle

interface ScreenObject {
    fun draw(g: Graphics2D, r: Screen)

    fun isVisible(r: Screen): Boolean
}
```

Листинг 14 – main/org/phdeh/a/gui/ScreenObject.kt

Листинг 15 – main/org/phdeh/a/gui/Vector.kt

```kotlin
package org.phdeh.a.gui

data class Vector(
    var x: Double = 0.0,
    var y: Double = 0.0
)
```

Листинг 15 – main/org/phdeh/a/gui/Vector.kt

Листинг 16 – main/org/phdeh/a/gui/Window.kt

```kotlin
package org.phdeh.a.gui

import org.phdeh.a.astar.AStar
import org.phdeh.a.astar.Graph
import org.phdeh.a.astar.forEachPair
import java.awt.*
import java.awt.Color.*
import javax.swing.JFrame
import java.awt.image.BufferedImage
import java.awt.Rectangle
import java.awt.event.ComponentEvent
import java.awt.event.ComponentAdapter
import java.awt.RenderingHints
import java.awt.event.KeyEvent

fun main() {
    Window.isVisible = true
}

object Window : JFrame("AStar Test") {

    @Volatile
    var renderBuffer = BufferedImage(1, 1, BufferedImage.TYPE_3BYTE_BGR)

    val nodeFont = Font("Helvetica", Font.BOLD, 40)

    val cursorOnScreen = Vector()

    val camera = Vector()
    val screen = Screen(camera, this)

    val edges = mutableListOf<Edge>()
    val nodes = mutableListOf<Node>()

    val nameLobby = NameLobby()

    var imaginaryNode = null as Node?
    var currentNode = null as Node?
    var lastTouch = null as Node?
    var prevTouch = null as Node?
    var solution = false

    val lineColor = Color(160, 128, 255, 100)
    val squareSize = 75
    val deleteCorner = 100
    val lineWidth = 6
    val selectDistance = 25
    val wheelMultiplier = 5

    var state = WindowState.NONE

    init {
        setSize(500, 500)
        centreWindow()
        initResizeListener()
        renderBuffer.graphics.fontMetrics.getLineMetrics("", renderBuffer.graphics)
    }

    val keyboardHandler = Keyboard(this) {
        var shouldRepaint = false
        if (it.remove)
            if (lastTouch !== null || prevTouch !== null) {
                clearSolution()
                lastTouch = null
                prevTouch = null
                shouldRepaint = true
            }
        if (it.released(KeyEvent.VK_SPACE)) {
            synchronized(this@Window) {
                val start = lastTouch
                val stop = prevTouch

                if (start == null || stop == null)
                    return@synchronized

                val graph = Graph.build { graph ->
                    val vertices = mutableMapOf<Node, Graph.GraphBuilder.VertexBuilder
                    ↪  >()
                    nodes.forEach {
                        vertices[it] = graph(it.name, it.x, it.y).vertexBuilder
                    }
                    edges.forEach {
                        val from = vertices[it.from]
                        val to = vertices[it.to]
                        if (from != null && to != null)
                            from - to
                    }
                }

                val paths = AStar.findAllPaths(graph[start.name], graph[stop.name])
                paths.forEach {
                    it.forEachPair { a, b ->
                        edges.forEach {
                            if (it.from.name == a.name && it.to.name == b.name ||
                                it.from.name == b.name && it.to.name == a.name)
                                it.shortest = true
                        }
                    }
                }
                solution = true
                shouldRepaint = true
            }
        }
        if (shouldRepaint)
            repaint()
    }

    val mouseHandler = Mouse(this) {
        var shouldRepaint = false

        if (it.horisontalWheel != 0 || it.verticalWheel != 0)
            shouldRepaint = true
        camera.x += it.horisontalWheel * wheelMultiplier
        camera.y += it.verticalWheel * wheelMultiplier

        if (it.leftButtonPressed) {
            clearSolution()
            val node = findCursorNode(it.x, it.y)
            if (keyboardHandler.command) {
                if (node != null) {
                    prevTouch = lastTouch
                    lastTouch = node
                    shouldRepaint = true
                }
            } else {
                if (node == null) {
                    state = WindowState.SET_VERTEX
                    imaginaryNode = Node(
                        nameLobby.getName(),
                        it.x + camera.x, it.y + camera.y, this
                    )
                } else {
                    state = WindowState.SET_VERTEX
                       nodes.removeAt(nodes.indexOf(node))
                    imaginaryNode = node
                }
                shouldRepaint = true
            }
        }

        if (it.rightButtonPressed) {
            val node = findCursorNode(it.x, it.y)
            if (node != null)
                clearSolution()
            else
                removeEdgesAt(it.x, it.y)
            currentNode = node
            shouldRepaint = true
        }

        val cn = currentNode
        if (cn != null) {
            if (it.rightButtonReleased) {
                val node = findCursorNode(it.x, it.y)
                if (node != null) {
                    edges += Edge(node, cn, this)
                }
                currentNode = null
            }
            shouldRepaint = true
        }


        val imn = imaginaryNode
        if (imn != null) {
            imn.x = it.x + camera.x - width / 2
            imn.y = it.y + camera.y - height / 2
            shouldRepaint = true
        }

        if (it.leftButtonReleased && imn != null) {
            val close = findGlobalNode(imn.tx.toInt(), imn.ty.toInt())
            if (close == null &&
                it.x >= -deleteCorner &&
                it.y >= -deleteCorner &&
                it.x <= width + deleteCorner &&
                it.y <= height + deleteCorner
            ) {
                imn.x = imn.tx
                imn.y = imn.ty
                nodes += imn
            } else {
                edges.removeAll { it.to === imn || it.from === imn }
                nameLobby.returnName(imn.name)
            }
            imaginaryNode = null
            shouldRepaint = true
        }

        if (shouldRepaint)
            repaint()
    }

    fun draw(g: Graphics2D) {
        listOf(edges, nodes).forEach {
            it.forEach {
                if (it.isVisible(screen))
                    it.draw(g, screen)
            }
        }
        val imn = imaginaryNode
        if (imn != null)
            imn.draw(g, screen)
        drawCurrentEdge(g)
    }

    fun drawCurrentEdge(g: Graphics2D) {
        val r = screen
        val from = currentNode
        if (from === null)
```

```
                return
            val to = mouseHandler
            g.stroke = BasicStroke(6f)
            g.color = Color.BLACK
            val dx = (-r.width / 2 + r.x).toInt()
            val dy = (-r.height / 2 + r.y).toInt()
            val a = Math.atan2(to.y + dy - from.y, to.x + dx - from.x)
            g.drawLine(
                from.x.toInt() - dx + (Math.cos(a) * from.width / 2).toInt(),
                from.y.toInt() - dy + (Math.sin(a) * from.height / 2).toInt(),
                to.x.toInt(),
                to.y.toInt()
            )
        }

        fun clearSolution() {
            if (solution) {
                edges.forEach { it.shortest = false }
                solution = false
            }
        }

        fun findCursorNode(x: Int, y: Int): Node? {
            return findGlobalNode(
                (x - width / 2 + camera.x).toInt(),
                (y - height / 2 + camera.y).toInt()
            )
        }

        fun findGlobalNode(x: Int, y: Int): Node? {
            var close = null as Node?
            nodes.forEach { i ->
                if ((x - i.x) * (x - i.x) +
                    (y - i.y) * (y - i.y) <=
                    selectDistance * selectDistance
                ) {
                    close = i
                }
            }
            return close
        }

        fun removeEdgesAt(x: Int, y: Int) {
    //        val r = screen
    //
    //        val dx = (-r.width / 2 + r.x).toInt()
    //        val dy = (-r.height / 2 + r.y).toInt()
    //
    //        val delete = mutableListOf<Edge>()
    //
    //        edges.forEach {
    //            val x1 = it.from.x.toInt() - dx
    //            val y1 = it.from.y.toInt() - dy
    //
    //            val x2 = it.to.x.toInt() - dx
    //            val y2 = it.to.y.toInt() - dy
    //
    //            val x3 = x2 - x1
```

```
//            val y3 = y2 - y1
//
//            val a = Math.atan2(it.to.y - it.from.y, it.to.x - it.from.x)
//
//            val mx = x - x1
//            val my = y - y2
//
//            val mx1 = mx * Math.cos(a) + my * Math.sin(a)
//            val my1 = mx * Math.sin(a) + my * Math.cos(a)
//
//            cursorOnScreen.x = mx.toDouble()
//            cursorOnScreen.y = my.toDouble()
//            println("$mx1:$my1")
//            println("$mx:$my")
//
//            val x5 = Math.sqrt(x3 * x3 + y3 * y3 + 0.0) / 2
//
//            if (Math.abs(my1) < 10 && Math.abs(mx1 - x5) < x5)
//                delete += it
//        }
//
//        edges.removeAll { it in delete }
    }

    ///////

    override fun paint(g: Graphics) {
        val r = renderBuffer
        val g2 = r.graphics as Graphics2D
        g2.color = WHITE
        g2.fillRect(0, 0, width, height)
        var rh = RenderingHints(
            RenderingHints.KEY_TEXT_ANTIALIASING,
            RenderingHints.VALUE_TEXT_ANTIALIAS_ON
        )
        g2.setRenderingHints(rh)
        rh = RenderingHints(
            RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON
        )
        g2.color = lineColor
        val linesHorizontally = (width / squareSize) / 2 + 2
        val linesVertically = (height / squareSize) / 2 + 2
        for (x in -linesHorizontally..linesHorizontally)
            g2.fillRect(
                (-camera.x % squareSize).toInt() - lineWidth / 2
                        + x * squareSize - squareSize + width / 2, 0, lineWidth,
                        ↪   height
            )
        for (y in -linesVertically..linesVertically)
            g2.fillRect(
                0, (-camera.y % squareSize).toInt() - lineWidth / 2
                        + y * squareSize - squareSize + height / 2, width, lineWidth
            )
        g2.font = nodeFont
        g2.setRenderingHints(rh)
        draw(g2)
//        g2.color = BLACK
//        g2.fillRect(cursorOnScreen.x.toInt() - 25, cursorOnScreen.y.toInt() - 25, 50, 50)
        g.drawImage(r, 0, 0, this)
    }

    override fun setSize(width: Int, height: Int) {
        handleResize(width, height)
        super.setSize(width, height)
    }

    override fun setSize(d: Dimension) {
        handleResize(d.width, d.height)
        super.setSize(d)
    }

    override fun setBounds(r: Rectangle) {
        handleResize(r.width, r.height)
        super.setBounds(r)
    }

    override fun setBounds(x: Int, y: Int, width: Int, height: Int) {
        handleResize(width, height)
        super.setBounds(x, y, width, height)
    }

    fun initResizeListener() {
        addComponentListener(object : ComponentAdapter() {
            override fun componentResized(evt: ComponentEvent?) {
                handleResize(width, height)
            }
        })
    }

    fun handleResize(width: Int, height: Int) {
        if (width > 0 && height > 0) {
            renderBuffer = BufferedImage(width, height, BufferedImage.TYPE_3BYTE_BGR)
        }
    }

    fun centreWindow() {
        val dimension = Toolkit.getDefaultToolkit().getScreenSize()
        val x = ((dimension.getWidth() - this.width) / 2).toInt()
        val y = ((dimension.getHeight() - this.height) / 2).toInt()
        this.setLocation(x, y)
    }
}
```

Листинг 16 – main/org/phdeh/a/gui/Window.kt

Листинг 17 – main/org/phdeh/a/gui/WindowState.kt

```
package org.phdeh.a.gui

enum class WindowState {
    NONE,
    SET_VERTEX,
    MOVE_VERTEX,
    CONNECT_VERTEX
}
```

Листинг 17 – main/org/phdeh/a/gui/WindowState.kt

Тесты

Листинг 18 – test/org/phdeh/a/astar/AStarMiltiplePathTest.kt

```kotlin
package org.phdeh.a.astar

import org.junit.Test

internal class AStarMiltiplePathTest {
    @Test
    fun multiplePaths() {
        val graph = Graph.build {
            val a by it(1.0, 0.0)
            val b by it(0.0, 1.0)
            val c by it(-1.0, 0.0)
            val d by it(0.0, -1.0)

            a - b - c - d - a
        }

        AStar.findAllPaths(graph["a"], graph["c"]) shouldBe listOf(
            listOf(graph["a"], graph["b"], graph["c"]),
            listOf(graph["a"], graph["d"], graph["c"])
            )
    }
}
```

Листинг 18 – test/org/phdeh/a/astar/AStarMiltiplePathTest.kt

Листинг 19 – test/org/phdeh/a/astar/AStarTest.kt

```kotlin
package org.phdeh.a.astar

import org.junit.Test

internal class AStarTest {

    @Test
    fun findPath() {
        val graph = Graph.build {
            val a by it(1.0, 0.0)
            val b by it(0.0, 1.0)
            val c by it(-1.0, 0.0)

            a - b - c
        }

        AStar.findPath(graph["a"], graph["c"]) shouldBe listOf(graph["a"], graph["b"],
        ↪  graph["c"])
    }

    @Test
    fun findShortestPath() {
        val graph = Graph.build {
            val a by it(1.0, 0.0)
            val b by it(0.0, 2.0)
            val c by it(0.0, 1.0)
            val d by it(-1.0, 0.0)

            a - b - d - c - a
        }

        AStar.findPath(graph["a"], graph["d"]) shouldBe listOf(graph["a"], graph["c"],
        ↪  graph["d"])
    }

    @Test
    fun wayDoesntExist() {
        val graph = Graph.build {
            val a by it(1.0, 0.0)
            val b by it(0.0, 2.0)
            val c by it(0.0, 1.0)
            val d by it(-1.0, 0.0)

            a - b; c - d
        }

        AStar.findPath(graph["a"], graph["d"]) shouldBe listOf<Graph.Vertex>()
    }

    @Test
    fun fromAToA() {
        val graph = Graph.build {
            val a by it(1.0, 0.0)
            val b by it(0.0, 2.0)
            val c by it(0.0, 1.0)
            val d by it(-1.0, 0.0)

            a - b - d - c - a
        }

        AStar.findPath(graph["a"], graph["a"]) shouldBe listOf(graph["a"])
    }

    @Test(expected = AStar.VerticeBelongToDifferentGraphsException::class)
    fun differentGraphs() {
        val graph1 = Graph.build {
            val a by it(1.0, 0.0)

            a
        }
        val graph2 = Graph.build {
            val a by it(1.0, 0.0)

            a
        }

        AStar.findPath(graph1["a"], graph2["a"])
    }

    @Test(expected = AStar.VertexDoesntExistException::class)
    fun nonExistingVerticesFrom() {
        val graph = Graph.build {
            val a by it(1.0, 0.0)

            a
        }

        AStar.findPath(graph["d"], graph["a"])
    }

    @Test(expected = AStar.VertexDoesntExistException::class)
    fun nonExistingVerticesTo() {
        val graph = Graph.build {
            val a by it(1.0, 0.0)

            a
        }

        AStar.findPath(graph["a"], graph["d"])
    }

    @Test(expected = AStar.VertexDoesntExistException::class)
    fun nonExistingVerticesBothFromAndTo() {
        val graph = Graph.build {
            val a by it(1.0, 0.0)

            a
        }

        AStar.findPath(graph["c"], graph["d"])
    }
}
```

Листинг 19 – test/org/phdeh/a/astar/AStarTest.kt

Листинг 20 – test/org/phdeh/a/astar/GraphTest.kt

```
package org.phdeh.a.astar

import org.junit.Test

internal class GraphTest {
    @Test
    fun emptyGraphTest() {
        val graph = Graph.build { }
        graph.size shouldBe 0
    }

    @Test
    fun correctValues() {
        val graph = Graph.build {
            val a by it(1.0, -1.0)

            a
        }
        graph["a"].name shouldBe "a"
        graph["a"].toString() shouldBe "a"
        graph["a"].x shouldBe 1.0
        graph["a"].y shouldBe -1.0
    }
    @Test
    fun explicitNaming() {
        val graph = Graph.build {
            val n1 by it("Moscow", 1.0, -1.0)
            val n2 by it("Saint-Petersburg", 1.0, -1.0)

            n1 - n2
        }
        ("Moscow" in graph) shouldBe true
        ("n1" in graph) shouldNotBe true
    }

    @Test
    fun duplicatingName() {
        val graph = Graph.build {
            val node1 by it("Node", 1.0, -1.0)
            val node2 by it("Node", 1.0, -1.0)
            val node3 by it("Node", 1.0, -1.0)

            node1 - node2 - node3
        }
        ("Node" in graph) shouldBe true
        ("Node_2" in graph) shouldBe true
        ("Node_3" in graph) shouldBe true
        ("Node_4" in graph) shouldNotBe true
    }

    @Test
    fun containsNameA() {
        val graph = Graph.build {
            val a by it(1.0, -1.0)

            a
        }
        ("a" in graph) shouldBe true
    }

    @Test
    fun aConnectedToB() {
        val graph = Graph.build {
            val a by it(1.0, 1.0)
            val b by it(-1.0, -1.0)

            a - b
        }
        graph["a"] hasEdgeTo graph["b"] shouldBe true
    }

    @Test
    fun commutativeConnection() {
        val graph = Graph.build {
            val a by it(1.0, 1.0)
            val b by it(-1.0, -1.0)

            b - a
        }
        graph["a"] hasEdgeTo graph["b"] shouldBe true
    }

    @Test
    fun aNotConnectedToB() {
        val graph = Graph.build {
            val a by it(1.0, 0.0)
            val b by it(-1.0, 0.0)
            val c by it(0.0, 1.0)

            a - c - b
        }
```

```kotlin
        graph["a"] hasEdgeTo graph["b"] shouldBe false
    }

    @Test
    fun indices() {
        val graph = Graph.build {
            val a by it(1.0, 0.0)
            val b by it(-1.0, 0.0)
            val c by it(0.0, 1.0)

            a - c - b
        }
        graph[0] shouldBe graph["a"]
        graph[1] shouldBe graph["b"]
        graph[2] shouldBe graph["c"]
        graph["a"].ordinal shouldBe 0
        graph["b"].ordinal shouldBe 1
        graph["c"].ordinal shouldBe 2
    }

    @Test
    fun backtracking() {
        val graph = Graph.build {
            val a by it(1.0, 0.0)

            a
        }
        graph["a"].graph shouldBe graph
        graph["b"].graph shouldNotBe graph
        graph["c"].graph shouldNotBe graph
        graph["b"].graph shouldNotBe graph["c"].graph
    }

    @Test
    fun dDoesntExists() {
        val graph = Graph.build {
            val a by it(1.0, 0.0)
            val b by it(-1.0, 0.0)
            val c by it(0.0, 1.0)

            a - c - b
        }
        (graph["d"] !in graph) shouldBe true
    }

    @Test
    fun distanceTest() {
        val graph = Graph.build {
            val a by it(1.0, 0.0)
            val b by it(-1.0, 0.0)

            a - b
        }
        (graph["a"] distanceTo graph["b"]) shouldBe 2.0
    }

    @Test
    fun impossibleDistanceTest() {
        val graph = Graph.build {
            val a by it(1.0, 0.0)

            a
        }
        (graph["a"] distanceTo graph["d"]).isNaN() shouldBe true
    }

    @Test
    fun edgesAreCorrect() {
        val graph = Graph.build {
            val a by it(1.0, 0.0)
            val b by it(-1.0, 0.0)
            val c by it(0.0, 1.0)
            val d by it(0.0, 2.0)

            a - b - c - a; b - d - c
        }
        val edges = graph["a"].edgesTo
        (graph["b"] in edges) shouldBe true
        (graph["c"] in edges) shouldBe true
        (graph["d"] in edges) shouldNotBe true
        (graph["e"] in edges) shouldNotBe true
    }

    @Test
    fun nonExistingVerticesAreNotEqual() {
        val graph = Graph.build {}
        (graph["a"] != graph["b"]) shouldBe true
    }

    @Test
    fun sizeOfGraph() {
        val graph = Graph.build {
            val a by it(1.0, 0.0)
            val b by it(-1.0, 0.0)
            val c by it(0.0, 1.0)

            a - b - c - a
        }
        graph.size shouldBe 3
    }
}
```

Листинг 20 – test/org/phdeh/a/astar/GraphTest.kt

Листинг 21 – test/org/phdeh/a/astar/PriorityQueueTest.kt

```kotlin
package org.phdeh.a.astar

import org.junit.Test

internal class PriorityQueueTest {
    @Test
    fun emptyQueue() {
        val pq = PriorityQueue<String>()
        pq.isEmpty() shouldBe true
        pq.isNotEmpty() shouldNotBe  true
    }

    @Test
    fun notEmptyQueue() {
        val pq = PriorityQueue<String>()
        pq.push("Test", 1.0)
        pq.isNotEmpty() shouldBe true
        pq.isEmpty() shouldNotBe  true
    }

    @Test
    fun priorityTest() {
        val pq = PriorityQueue<String>()
        pq.push("Foo", 2.0)
        pq.push("Bar", 1.0)
        pq.push("Baz", 3.0)
        pq.pop() shouldBe "Bar"
        pq.pop() shouldBe "Foo"
        pq.pop() shouldBe "Baz"
    }

    @Test
    fun startValues() {
        val pq = PriorityQueue<String>("Foo", "Bar", "Baz")
        pq.pop() shouldBe "Foo"
        pq.pop() shouldBe "Bar"
        pq.pop() shouldBe "Baz"
    }

    @Test(expected = PriorityQueue.EmptyQueueException::class)
    fun emptyQueuePop() {
        val pq = PriorityQueue<String>()
        pq.pop()
    }
}
```

Листинг 21 – test/org/phdeh/a/astar/PriorityQueueTest.kt

Листинг 22 – test/org/phdeh/a/gui/WIndowTest.kt

```
package org.phdeh.a.gui

import org.junit.Test
import org.phdeh.a.astar.shouldBe
import org.phdeh.a.astar.shouldNotBe
import java.awt.GraphicsEnvironment
import java.awt.Robot
import java.awt.event.InputEvent
import java.awt.event.KeyEvent

internal class WindowTest {
    @Test
    fun testGui() {
        if (GraphicsEnvironment.isHeadless())
            return

        val wheeling = Window.squareSize * 2 / Window.wheelMultiplier
        val robotTimer = 100L

        Window.isVisible = true
        Window.toFront()
        val robot = Robot()
        Thread.sleep(robotTimer)
        robot.mouseMove(0, 0)
        Thread.sleep(robotTimer)
        robot.mouseMove(Window.x + Window.width / 2, Window.y + Window.height / 2)
        Thread.sleep(robotTimer)
        // Set first
        robot.mousePress(InputEvent.BUTTON1_DOWN_MASK)
        Thread.sleep(robotTimer)
        robot.mouseRelease(InputEvent.BUTTON1_DOWN_MASK)
        Thread.sleep(robotTimer)
        // Move to second
        robot.mouseWheel(wheeling)
        Thread.sleep(robotTimer)
        robot.keyPress(KeyEvent.VK_SHIFT)
        Thread.sleep(robotTimer)
        robot.mouseWheel(wheeling)
        Thread.sleep(robotTimer)
        robot.keyRelease(KeyEvent.VK_SHIFT)
        Thread.sleep(robotTimer)
        // Set second
        robot.mousePress(InputEvent.BUTTON1_DOWN_MASK)
        Thread.sleep(robotTimer)
        robot.mouseRelease(InputEvent.BUTTON1_DOWN_MASK)
        Thread.sleep(robotTimer * 2)
        // Begin edge
        robot.mousePress(InputEvent.BUTTON3_DOWN_MASK)
        Thread.sleep(robotTimer)
        // Return to first
        robot.mouseWheel(-wheeling)
        Thread.sleep(robotTimer)
        robot.keyPress(KeyEvent.VK_SHIFT)
        Thread.sleep(robotTimer)
        robot.mouseWheel(-wheeling)
        Thread.sleep(robotTimer)
        robot.keyRelease(KeyEvent.VK_SHIFT)
        Thread.sleep(robotTimer)
        // End edge
        robot.mouseRelease(InputEvent.BUTTON3_DOWN_MASK)
        Thread.sleep(robotTimer)

        Window.setSize(600, 400)
        Window.centreWindow()

        Thread.sleep(robotTimer)
        // Set third
        robot.mouseMove(
            Window.x + Window.width / 2 + Window.squareSize * 2,
            Window.y + Window.height / 2 - Window.squareSize * 2
        )
        Thread.sleep(robotTimer)
        robot.mousePress(InputEvent.BUTTON1_DOWN_MASK)
        Thread.sleep(robotTimer)
        robot.mouseRelease(InputEvent.BUTTON1_DOWN_MASK)
        Thread.sleep(robotTimer)
        // Replacing third
        robot.mousePress(InputEvent.BUTTON1_DOWN_MASK)
        Thread.sleep(robotTimer)
        robot.mouseMove(
            Window.x + Window.width / 2 + Window.squareSize * 2,
            Window.y + Window.height / 2 + Window.squareSize * 2
        )
        Thread.sleep(robotTimer)
        robot.mouseRelease(InputEvent.BUTTON1_DOWN_MASK)
        Thread.sleep(robotTimer)
        // Begin edge
        robot.mousePress(InputEvent.BUTTON3_DOWN_MASK)
        Thread.sleep(robotTimer)
        robot.mouseMove(Window.x + Window.width / 2, Window.y + Window.height / 2)
        Thread.sleep(robotTimer)
        // End edge
        robot.mouseRelease(InputEvent.BUTTON3_DOWN_MASK)
        Thread.sleep(robotTimer)
        // Setting first way point
        robot.keyPress(KeyEvent.VK_META)
        Thread.sleep(robotTimer)
        robot.mousePress(InputEvent.BUTTON1_DOWN_MASK)
        Thread.sleep(robotTimer)
        robot.mouseRelease(InputEvent.BUTTON1_DOWN_MASK)
        Thread.sleep(robotTimer)

        Window.setSize(1000, 1000)
        Window.centreWindow()

        Thread.sleep(robotTimer)
        // Setting second way point
        robot.mouseMove(
            Window.x + Window.width / 2 - Window.squareSize * 2,
            Window.y + Window.height / 2 - Window.squareSize * 2
        )
        Thread.sleep(robotTimer)
        robot.mousePress(InputEvent.BUTTON1_DOWN_MASK)
        Thread.sleep(robotTimer)
        robot.mouseRelease(InputEvent.BUTTON1_DOWN_MASK)
        Thread.sleep(robotTimer)
        // Searching path
        robot.keyRelease(KeyEvent.VK_META)
        Thread.sleep(robotTimer)
        robot.keyPress(KeyEvent.VK_SPACE)
        Thread.sleep(robotTimer)
        robot.keyRelease(KeyEvent.VK_SPACE)
        Thread.sleep(robotTimer)

        Window.edges.size shouldBe 2

        Window.edges.forEach {
            if (it.from.name == "A" && it.to.name == "B" || it.from.name == "B" && it.
            ↪  to.name == "A")
                it.shortest shouldBe true
            else
                it.shortest shouldNotBe true
        }

        Thread.sleep(robotTimer)
        robot.keyPress(KeyEvent.VK_BACK_SPACE)
        Thread.sleep(robotTimer)
        robot.keyRelease(KeyEvent.VK_BACK_SPACE)
        Thread.sleep(robotTimer)

        Window.edges.size shouldBe 2

        Window.edges.forEach {
            it.shortest shouldNotBe true
        }

        // Removing B
        Thread.sleep(robotTimer)
        robot.mousePress(InputEvent.BUTTON1_DOWN_MASK)
        Thread.sleep(robotTimer)
        robot.mouseMove(0, 0)
        Thread.sleep(robotTimer)
        robot.mouseRelease(InputEvent.BUTTON1_DOWN_MASK)
        Thread.sleep(robotTimer)

        Window.edges.size shouldBe 1

        Window.edges.forEach {
            it.shortest shouldNotBe true
        }


        Thread.sleep(robotTimer * 5)
    }
}
```

Листинг 22 – `test/org/phdeh/a/gui/WIndowTest.kt`