

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  
должность, уч. степень, звание

подпись, дата

П. А. Степанов  
инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

по дисциплине: УПРАВЛЕНИЕ КАЧЕСТВОМ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

РАБОТУ ВЫПОЛНИЛ  
СТУДЕНТ ГР.

4631  
№ группы

подпись, дата

Д. С. Воробьев  
инициалы, фамилия

Санкт-Петербург

2019 г.

## Содержание

<b>1</b>	<b>Задание . . . . .</b>	<b>3</b>
<b>2</b>	<b>Выполнение . . . . .</b>	<b>3</b>
<b>3</b>	<b>Пример решения задачи . . . . .</b>	<b>3</b>
<b>4</b>	<b>План по тестированию . . . . .</b>	<b>5</b>
<b>5</b>	<b>Система непрерывной интеграции . . . . .</b>	<b>6</b>
	<b>Заключение . . . . .</b>	<b>7</b>
	<b>Приложение. . . . .</b>	<b>8</b>
	Исходный код программы . . . . .	8
	Тесты . . . . .	27

## 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 РМВОК)
7. Настроить систему непрерывной интеграции. Сделать так, чтобы тесты на реализованный функционал выполнялись в ответ на интеграцию.
8. Создать отчет по курсовому проектированию.

## 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`

```
1 package org.phdeh.a.astar
2
3 import java.lang.RuntimeException
4 import java.util.*
5
6 object AStar {
7     class VertexBelongsToDifferentGraphsException(msg: String) : RuntimeException(msg)
8     class VertexDoesntExistException(msg: String) : RuntimeException(msg)
9
10    fun findPath(from: Graph.Vertex, to: Graph.Vertex): List<Graph.Vertex> {
11        if (from === Graph.NON_EXISTENT_VERTEX && to === Graph.NON_EXISTENT_VERTEX)
```

```

12         throw VertexDoesntExistException("both from and to")
13     if (from == Graph.NON_EXISTENT_VERTEX)
14         throw VertexDoesntExistException("from")
15     if (to == Graph.NON_EXISTENT_VERTEX)
16         throw VertexDoesntExistException("to")
17     if (from.graph != to.graph)
18         throw VerticeBelongToDifferentGraphsException("$from and $to")
19     if (from == to)
20         return listOf(to)
21     val graph = from.graph
22
23     val frontier = PriorityQueue(from)
24     val cameFrom = Array<Graph.Vertex?>(graph.size) { null }
25     val handled = BitSet(graph.size)
26     val costSoFar = DoubleArray(graph.size) { 0.0 }
27
28     handled[from.ordinal] = true
29
30     fun directDist(v: Graph.Vertex) = Math.sqrt(
31         Math.pow((to.y - v.y), 2.0) +
32         Math.pow((to.x - v.x), 2.0)
33     )
34
35     var last = from
36     while (frontier.isNotEmpty()) {
37         val current = frontier.pop()
38         last = current
39         if (current == to)
40             break
41
42         current.edgesTo.forEach {
43             val newCost = costSoFar[current.ordinal] + it.distanceTo(current)
44             if (!handled[it.ordinal] || newCost < costSoFar[it.ordinal]) {
45                 handled[it.ordinal] = true
46                 costSoFar[it.ordinal] = newCost
47                 cameFrom[it.ordinal] = current
48                 val priority = newCost + directDist(it)
49                 frontier.push(it, priority)
50             }
51         }
52     }
53
54     if (last == to) {
55         val ml = mutableList<Graph.Vertex>()
56         var curr: Graph.Vertex? = last
57         while (curr != null) {
58             ml.add(curr)
59             curr = cameFrom[curr.ordinal]
60         }
61         return ml.reversed()
62     }
63     return listOf()
64 }
65
66 fun findAllPaths(from: Graph.Vertex, to: Graph.Vertex): List<List<Graph.Vertex>> {
67     val paths = mutableSetOf(findPath(from, to))
68     if (from != to && paths.isEmpty()) {
69         var minLength = 0.0

```

```

70     paths.first().forEachPair { a, b ->
71         minLength += a.distanceTo b
72     }
73     fun depthFirst(current: Graph.Vertex, visited: List<Graph.Vertex>, length:
74         ⇨ Double) {
75         if (current == to && length <= minLength + 0.001)
76             paths += visited
77         if (length <= minLength + 0.001)
78             current.edgesTo.forEach {
79                 if (it !in visited)
80                     depthFirst(it, visited + it, length + current.distanceTo(
81                         ⇨ it))
82             }
83         depthFirst(from, listOf(from), 0.0)
84     }
85     return paths.toList()
86 }

```

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

```
1 language: java
2 install: true
3
4 jdk:
5   - oraclejdk8
6
7 script:
8   - ./gradlew wrapper --gradle-version=5.4.1 --distribution-type=bin
9   - ./gradlew test
```

Листинг 2 – .travis.yml

Текст конфигурации Gradle:

Листинг 3 – build.gradle

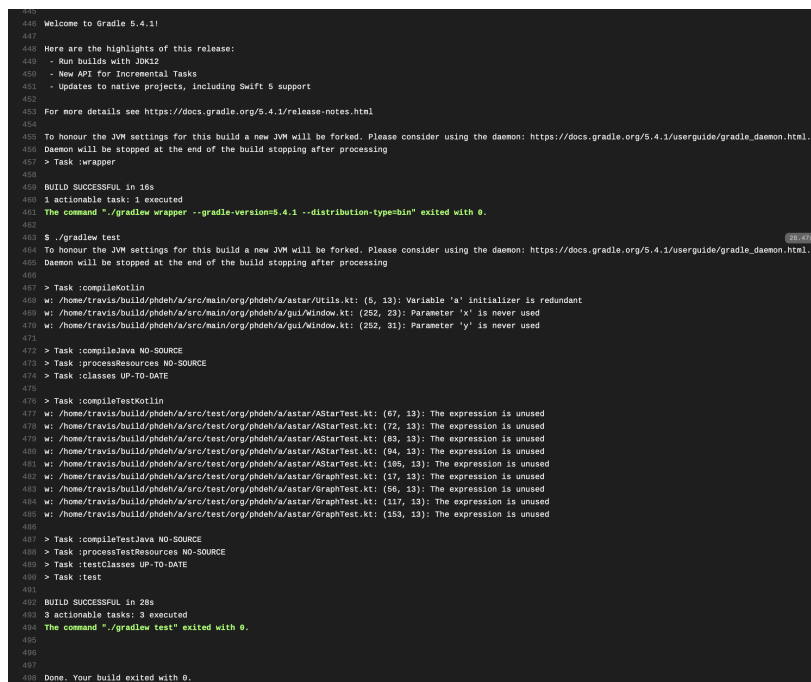
```
1 buildscript {
2     repositories {
3         mavenCentral()
4     }
5
6     dependencies {
7         classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:1.3.31"
8     }
9 }
10
11 plugins {
12     id 'java'
13     id 'org.jetbrains.kotlin.jvm' version '1.3.21'
14 }
15
16 group 'AStarCW'
17 version '1.0-SNAPSHOT'
18
19 sourceCompatibility = 1.8
20
21 repositories {
22     mavenCentral()
23 }
24
25 dependencies {
26     compile "org.jetbrains.kotlin:kotlin-stdlib:1.3.31"
27     compile 'junit:junit:4.12'
28     testCompile 'junit:junit:4.12'
29     testImplementation "org.jetbrains.kotlin:kotlin-stdlib-jdk8"
30 }
31
32 compileKotlin {
33     kotlinOptions.jvmTarget = "1.8"
34 }
35 compileTestKotlin {
36     kotlinOptions.jvmTarget = "1.8"
37 }
38
```

```

39 sourceSets {
40     main.java.srcDirs += 'src/main'
41     main.kotlin.srcDirs += 'src/main'
42     test.java.srcDirs += 'src/test'
43     test.kotlin.srcDirs += 'src/test'
44 }
45
46
47 test {
48     useJUnit()
49
50     maxHeapSize = '1G'
51 }

```

Листинг 3 – build.gradle



```

440 Welcome to Gradle 5.4.1!
441
442 Here are the highlights of this release:
443 - Run builds with JDK12
444 - New API for Incremental Tasks
445 - Updates to native projects, including Swift 5 support
446
447 For more details see https://docs.gradle.org/5.4.1/release-notes.html
448
449 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.
450 Daemon will be stopped at the end of the build stopping after processing
451 > Task :wrapper
452
453 BUILD SUCCESSFUL in 16s
454 1 actionable task: 1 executed
455 The command "/gradlew wrapper --gradle-version=5.4.1 --distribution-type=bin" exited with 0.
456
457 $ ./gradlew test
458 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.
459 Daemon will be stopped at the end of the build stopping after processing
460 > Task :compileKotlin
461 w: /home/travis/build/phdeh/a/src/main/org/phdeh/a/astar/Utils.kt: (5, 13): Variable 'a' initializer is redundant
462 w: /home/travis/build/phdeh/a/src/main/org/phdeh/a/gui/Window.kt: (252, 23): Parameter 'x' is never used
463 w: /home/travis/build/phdeh/a/src/main/org/phdeh/a/gui/Window.kt: (252, 31): Parameter 'y' is never used
464
465 > Task :compileJava NO-SOURCE
466 > Task :processResources NO-SOURCE
467 > Task :classes UP-TO-DATE
468
469 > Task :compileTestKotlin
470 w: /home/travis/build/phdeh/a/src/test/org/phdeh/a/astar/AStarTest.kt: (67, 13): The expression is unused
471 w: /home/travis/build/phdeh/a/src/test/org/phdeh/a/astar/AStarTest.kt: (72, 13): The expression is unused
472 w: /home/travis/build/phdeh/a/src/test/org/phdeh/a/astar/AStarTest.kt: (83, 13): The expression is unused
473 w: /home/travis/build/phdeh/a/src/test/org/phdeh/a/astar/AStarTest.kt: (94, 13): The expression is unused
474 w: /home/travis/build/phdeh/a/src/test/org/phdeh/a/astar/AStarTest.kt: (105, 13): The expression is unused
475 w: /home/travis/build/phdeh/a/src/test/org/phdeh/a/astar/GraphTest.kt: (17, 13): The expression is unused
476 w: /home/travis/build/phdeh/a/src/test/org/phdeh/a/astar/GraphTest.kt: (56, 13): The expression is unused
477 w: /home/travis/build/phdeh/a/src/test/org/phdeh/a/astar/GraphTest.kt: (117, 13): The expression is unused
478 w: /home/travis/build/phdeh/a/src/test/org/phdeh/a/astar/GraphTest.kt: (153, 13): The expression is unused
479
480 > Task :compileTestJava NO-SOURCE
481 > Task :processTestResources NO-SOURCE
482 > Task :testClasses UP-TO-DATE
483 > Task :test
484
485 BUILD SUCCESSFUL in 28s
486 3 actionable tasks: 3 executed
487 The command "./gradlew test" exited with 0.
488
489 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

```
1 package org.phdeh.a.astar
2
3 import java.lang.RuntimeException
4 import java.util.*
5
6 object AStar {
7     class VertexBelongToDifferentGraphsException(msg: String) : RuntimeException(msg)
8     class VertexDoesntExistException(msg: String) : RuntimeException(msg)
9
10    fun findPath(from: Graph.Vertex, to: Graph.Vertex): List<Graph.Vertex> {
11        if (from === Graph.NON_EXISTENT_VERTEX && to === Graph.NON_EXISTENT_VERTEX)
12            throw VertexDoesntExistException("both from and to")
13        if (from === Graph.NON_EXISTENT_VERTEX)
14            throw VertexDoesntExistException("from")
15        if (to === Graph.NON_EXISTENT_VERTEX)
16            throw VertexDoesntExistException("to")
17        if (from.graph != to.graph)
18            throw VertexBelongToDifferentGraphsException("$from and $to")
19        if (from == to)
20            return listOf(to)
21        val graph = from.graph
22
23        val frontier = PriorityQueue(from)
24        val cameFrom = Array<Graph.Vertex?>(graph.size) { null }
25        val handled = BitSet(graph.size)
26        val costSoFar = DoubleArray(graph.size) { 0.0 }
27
28        handled[from.ordinal] = true
29
30        fun directDist(v: Graph.Vertex) = Math.sqrt(
31            Math.pow((to.y - v.y), 2.0) +
32            Math.pow((to.x - v.x), 2.0)
33        )
34
35        var last = from
36        while (frontier.isNotEmpty()) {
37            val current = frontier.pop()
38            last = current
39            if (current == to)
40                break
41
42            current.edgesTo.forEach {
43                val newCost = costSoFar[current.ordinal] + it.distanceTo(current)
44                if (!handled[it.ordinal] || newCost < costSoFar[it.ordinal]) {
45                    handled[it.ordinal] = true
46                    costSoFar[it.ordinal] = newCost
47                    cameFrom[it.ordinal] = current
48                    val priority = newCost + directDist(it)
49                    frontier.push(it, priority)
50                }
51            }
52        }
53    }
```



```

54         if (last == to) {
55             val ml = mutableListOf<Graph.Vertex>()
56             var curr: Graph.Vertex? = last
57             while (curr != null) {
58                 ml.add(curr)
59                 curr = cameFrom[curr.ordinal]
60             }
61             return ml.reversed()
62         }
63         return listOf()
64     }
65
66     fun findAllPaths(from: Graph.Vertex, to: Graph.Vertex): List<List<Graph.Vertex>> {
67         val paths = mutableSetOf(findPath(from, to))
68         if (from != to && paths.isNotEmpty()) {
69             var minLength = 0.0
70             paths.first().forEachPair { a, b ->
71                 minLength += a.distanceTo b
72             }
73             fun depthFirst(current: Graph.Vertex, visited: List<Graph.Vertex>, length:
74                 ↳ Double) {
75                 if (current == to && length <= minLength + 0.001)
76                     paths += visited
77                 if (length <= minLength + 0.001)
78                     current.edgesTo.forEach {
79                         if (it !in visited)
80                             depthFirst(it, visited + it, length + current.distanceTo(
81                                 ↳ it))
82                     }
83             }
84             depthFirst(from, listOf(from), 0.0)
85         }
86         return paths.toList()
87     }
88 }

```

Листинг 4 – main/org/phdeh/a/astar/AStar.kt  
 Листинг 5 – main/org/phdeh/a/astar/Graph.kt

```

1 package org.phdeh.a.astar
2
3 import kotlin.reflect.KProperty
4
5 class Graph private constructor(builder: GraphBuilder) {
6
7     companion object {
8         fun build(action: (GraphBuilder) -> Unit): Graph {
9             val builder = GraphBuilder()
10             action(builder)
11             return Graph(builder)
12         }
13
14         val NON_EXISTENT_GRAPH = build { }
15         val NON_EXISTENT_VERTEX = Vertex(
16             NON_EXISTENT_GRAPH,
17             "NON_EXISTENT_GRAPH",
18             Double.NaN,
19             Double.NaN,
20             -1,
21             mutableListOf()

```

```

22     )
23 }
24
25 private val vertices: Map<String, Vertex>
26 private val verticesIndexed: List<Vertex>
27
28 operator fun get(name: String) = vertices[name] ?: NON_EXISTENT_VERTEX
29
30 operator fun get(index: Int) = if (index in this) verticesIndexed[index] else
    ↪ NON_EXISTENT_VERTEX
31
32 val size get() = vertices.size
33
34 operator fun contains(name: String) = vertices.containsKey(name)
35
36 operator fun contains(vertex: Vertex) = vertices.containsValue(vertex)
37
38 operator fun contains(index: Int) = index in 0..(vertices.size - 1)
39
40 override fun equals(other: Any?): Boolean {
41     return other != null &&
42         other is Graph &&
43         this !== NON_EXISTENT_GRAPH &&
44         other !== NON_EXISTENT_GRAPH &&
45         this === other
46 }
47
48 override fun hashCode(): Int = vertices.hashCode()
49
50 init {
51     synchronized(builder) {
52         val vb = List(builder.vertices.size) { builder.vertices[it].vertexBuilder
53             ↪ }
54         val relations = List(vb.size) { mutableListOf<Vertex>() }
55         val names = mutableSetOf<String>()
56         val verticesList = List(vb.size) {
57             val b = vb[it]
58             if (b.name !in names) {
59                 names += b.name
60                 Vertex(this, b.name, b.x, b.y, it, relations[it])
61             } else {
62                 var i = 2
63                 while ("${b.name}_$i" in names)
64                     i++
65                 names += "${b.name}_$i"
66                 Vertex(this, "${b.name}_$i", b.x, b.y, it, relations[it])
67             }
68         }
69         relations.forEachIndexed { it, list ->
70             vb[it].currentlyEdgesTo.forEach { e ->
71                 list += verticesList[vb.indexOf(e)]
72             }
73         }
74         val verticesMap = mutableMapOf<String, Vertex>()
75         verticesList.forEach {
76             verticesMap[it.name] = it
77         }
78         this.verticesIndexed = verticesList

```

```

78         this.vertices = verticesMap
79     }
80 }
81
82 data class Vertex internal constructor(
83     val graph: Graph,
84     val name: String,
85     val x: Double,
86     val y: Double,
87     val ordinal: Int,
88     private val hasEdgesTo: MutableList<Vertex>
89 ) {
90     val edgesTo by lazy { List(hasEdgesTo.size, { hasEdgesTo[it] }).toSet() }
91
92     infix fun hasEdgeTo(other: Vertex?) = this !== NON_EXISTENT_VERTEX &&
93         other != null &&
94         other !== NON_EXISTENT_VERTEX &&
95         other in edgesTo
96
97     override fun equals(other: Any?): Boolean {
98         return this !== NON_EXISTENT_VERTEX && other !== NON_EXISTENT_VERTEX &&
99             ↪ super.equals(other)
100     }
101
102     override fun hashCode(): Int {
103         return name.hashCode() xor x.hashCode() xor y.hashCode()
104     }
105
106     infix fun distanceTo(other: Vertex): Double =
107         if (this === NON_EXISTENT_VERTEX || other === NON_EXISTENT_VERTEX)
108             Double.NaN
109         else
110             Math.sqrt((x - other.x) * (x - other.x) + (y - other.y) * (y - other.y)
111                 ↪ )
112
113     override fun toString(): String = name
114 }
115
116 class GraphBuilder {
117
118     internal val vertices = mutableListOf<VertexDelegate>()
119
120     operator fun invoke(x: Double, y: Double): VertexDelegate {
121         val vd = VertexDelegate(null, x, y)
122         vertices += vd
123         return vd
124     }
125
126     operator fun invoke(name: String, x: Double, y: Double): VertexDelegate {
127         val vd = VertexDelegate(name, x, y)
128         vertices += vd
129         return vd
130     }
131
132     class VertexBuilder(val x: Double, val y: Double) {
133         private var actualName: String? = null
134         var name
135             get() = actualName ?: "UNDEFINED"
136     }
137 }

```

```

134         set(value) {
135             if (actualName == null)
136                 actualName = value
137         }
138
139         private val actuallyEdgesTo = mutableSetOf<VertexBuilder>()
140
141         val currentlyEdgesTo get() = actuallyEdgesTo.toSet()
142
143         operator fun minus(other: VertexBuilder): VertexBuilder {
144             hasEdgeTo(other)
145             return other
146         }
147
148         fun hasEdgeTo(other: VertexBuilder) {
149             synchronized(this) {
150                 actuallyEdgesTo += other
151             }
152             synchronized(other) {
153                 other.actuallyEdgesTo += this
154             }
155         }
156     }
157
158     class VertexDelegate(name: String?, x: Double, y: Double) {
159         val vertexBuilder = VertexBuilder(x, y).let {
160             if (name != null)
161                 it.name = name
162             it
163         }
164
165         operator fun getValue(thisRef: Any?, property: KProperty<*>):
166             ↪ VertexBuilder {
167             vertexBuilder.name = property.name
168             return vertexBuilder
169         }
170     }
171 }
172
173 }

```

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

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

```

1 package org.phdeh.a.astar
2
3 import java.lang.RuntimeException
4
5 class PriorityQueue<T>(vararg ts: T) {
6     class EmptyQueueException : RuntimeException()
7     private val mp = mutableListOf<Double>()
8     private val mt = mutableListOf<T>()
9
10     init {
11         ts.forEach {
12             push(it, 0.0)
13         }
14     }
15 }

```

```

16     fun push(t: T, priority: Double) {
17         synchronized(this) {
18             mp.add(priority)
19             mt.add(t)
20         }
21     }
22
23     fun isEmpty() = mp.isEmpty()
24     fun isEmpty() = !isEmpty()
25     fun pop(): T {
26         if (isEmpty())
27             throw EmptyQueueException()
28         synchronized(this) {
29             var max = mp[0]
30             var index = 0
31             var i = 0
32             do {
33                 if (mp[i] < max) {
34                     index = i
35                     max = mp[i]
36                 }
37             } while (++i < mp.size)
38             mp.removeAt(index)
39             val result = mt[index]
40             mt.removeAt(index)
41             return result
42         }
43     }
44 }

```

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

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

```

1 package org.phdeh.a.astar
2
3
4 fun<K,T : Iterable<K>> T.forEachPair(action: (a: K, b: K) -> Unit) {
5     var a = null as K?
6     var b = null as K?
7     val iter = this.iterator()
8     while (iter.hasNext()) {
9         a = b
10        b = iter.next()
11        if (a != null && b != null)
12            action(a, b)
13    }
14 }

```

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

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

```

1 package org.phdeh.a.gui
2
3 import java.awt.BasicStroke
4 import java.awt.Color
5 import java.awt.Graphics2D
6
7 data class Edge(
8     val from: Node,
9     val to: Node,
10    val win: Window

```

```

11 ) : ScreenObject {
12     var shortest = false
13
14     override fun isVisible(r: Screen): Boolean {
15         val x = (from.x + to.x) / 2
16         val y = (from.y + to.y) / 2
17         val width = Math.abs(from.x - to.x)
18         val height = Math.abs(from.y - to.y)
19         return Math.abs(x - r.x) < (r.width + width) / 2 &&
20             Math.abs(y - r.y) < (r.height + height) / 2
21     }
22
23     override fun draw(g: Graphics2D, r: Screen) {
24         val dx = (-r.width / 2 + r.x).toInt()
25         val dy = (-r.height / 2 + r.y).toInt()
26         if (shortest)
27             g.stroke = BasicStroke(6f)
28         else
29             g.stroke = BasicStroke(3f)
30         g.color = Color.BLACK
31         g.drawLine(
32             from.x.toInt() - dx,
33             from.y.toInt() - dy,
34             to.x.toInt() - dx,
35             to.y.toInt() - dy
36         )
37
38         if (shortest) {
39             g.stroke = BasicStroke(3f)
40             g.color = Color.GREEN
41             g.drawLine(
42                 from.x.toInt() - dx,
43                 from.y.toInt() - dy,
44                 to.x.toInt() - dx,
45                 to.y.toInt() - dy
46             )
47         }
48     }
49 }

```

Листинг 8 – main/org/phdeh/a/gui/Edge.kt  
 Листинг 9 – main/org/phdeh/a/gui/Keyboard.kt

```

1 package org.phdeh.a.gui
2
3 import java.awt.event.KeyEvent
4 import java.awt.event.KeyListener
5 import java.util.*
6
7 class Keyboard(
8     private val window: Window,
9     private val action: (Keyboard) -> Unit
10 ) {
11     private val keys = BitSet(256)
12     private val pkeys = BitSet(256)
13     private var _typed = '\u0000'
14     val typed get() = _typed
15
16     operator fun get(keyCode: Int) =
17         if (keyCode in 0..255)

```

```

18         keys[keyCode]
19     else false
20
21     fun pressed(keyCode: Int) =
22         if (keyCode in 0..255)
23             keys[keyCode] && !pkeys[keyCode]
24         else false
25
26     fun released(keyCode: Int) =
27         if (keyCode in 0..255)
28             !keys[keyCode] && pkeys[keyCode]
29         else false
30
31     val command
32     get() = this[KeyEvent.VK_CONTROL] ||
33             this[KeyEvent.VK_ALT] ||
34             this[KeyEvent.VK_META]
35
36     val handle
37     get() = this[KeyEvent.VK_SPACE]
38
39     val remove
40     get() = this[KeyEvent.VK_BACK_SPACE] ||
41             this[KeyEvent.VK_DELETE]
42
43     init {
44         fun handle() {
45             for (i in 0..255)
46                 pkeys[i] = keys[i]
47         }
48
49         window.addKeyListener(object : KeyListener {
50             override fun keyPressed(e: KeyEvent) {
51                 handle()
52                 keys[e.keyCode] = true
53                 action(this@Keyboard)
54             }
55
56             override fun keyReleased(e: KeyEvent) {
57                 handle()
58                 keys[e.keyCode] = false
59                 action(this@Keyboard)
60             }
61
62             override fun keyTyped(e: KeyEvent) {
63                 _typed = e.keyChar
64                 action(this@Keyboard)
65             }
66         })
67     }
68 }

```

Листинг 9 – main/org/phdeh/a/gui/Keyboard.kt  
 Листинг 10 – main/org/phdeh/a/gui/Mouse.kt

```

1 package org.phdeh.a.gui
2
3 import java.awt.event.*
4
5 class Mouse(

```

```

6     private val window: Window,
7     private val action: (Mouse) -> Unit
8 ) {
9     private var _x = 0
10    private var _y = 0
11    private var _dx = 0
12    private var _dy = 0
13    private var _lb = false
14    private var _rb = false
15    private var _mb = false
16    private var _lb_r = false
17    private var _rb_r = false
18    private var _mb_r = false
19    private var _lb_p = false
20    private var _rb_p = false
21    private var _mb_p = false
22    private var _hw = 0
23    private var _vw = 0
24
25    val x get() = _x
26    val y get() = _y
27    val dx get() = _dx
28    val dy get() = _dy
29    val leftButton get() = _lb
30    val rightButton get() = _rb
31    val middleButton get() = _mb
32    val leftButtonReleased get() = _lb_r
33    val rightButtonReleased get() = _rb_r
34    val middleButtonReleased get() = _mb_r
35    val leftButtonPressed get() = _lb_p
36    val rightButtonPressed get() = _rb_p
37    val middleButtonPressed get() = _mb_p
38    val horizontalWheel get() = _hw
39    val verticalWheel get() = _vw
40
41    init {
42        fun handle(e: MouseEvent, mb: Boolean? = null) {
43            val px = _x
44            val py = _y
45            _x = e.x
46            _y = e.y
47            _dx = _x - px
48            _dy = _y - py
49
50            val plb = _lb
51            val prb = _rb
52            val pmb = _mb
53
54            if (mb != null)
55                when (e.button) {
56                    MouseEvent.BUTTON1 -> {
57                        _lb = mb
58                    }
59                    MouseEvent.BUTTON3 -> {
60                        _rb = mb
61                    }
62                    MouseEvent.BUTTON2 -> {
63                        _mb = mb

```



```

64         }
65     }
66
67     _lb_p = !plb && !_lb
68     _rb_p = !prb && !_rb
69     _mb_p = !pmb && !_mb
70
71     _lb_r = plb && !_lb
72     _rb_r = prb && !_rb
73     _mb_r = pmb && !_mb
74
75     _hw = 0
76     _vw = 0
77 }
78
79 val ml = object : MouseListener {
80     override fun mouseClicked(e: MouseEvent) {
81         handle(e, null)
82         action(this@Mouse)
83     }
84
85     override fun mouseEntered(e: MouseEvent) {
86         handle(e, null)
87         action(this@Mouse)
88     }
89
90     override fun mouseExited(e: MouseEvent) {
91         handle(e, null)
92         action(this@Mouse)
93     }
94
95     override fun mousePressed(e: MouseEvent) {
96         handle(e, true)
97         action(this@Mouse)
98     }
99
100    override fun mouseReleased(e: MouseEvent) {
101        handle(e, false)
102        action(this@Mouse)
103    }
104 }
105 val mml = object : MouseMotionListener {
106     override fun mouseDragged(e: MouseEvent) {
107         handle(e, true)
108         action(this@Mouse)
109     }
110
111     override fun mouseMoved(e: MouseEvent) {
112         handle(e, false)
113         action(this@Mouse)
114     }
115 }
116 val mwl = object : MouseWheelListener {
117     override fun mouseWheelMoved(e: MouseWheelEvent) {
118         handle(e, null)
119         if (e.isShiftDown)
120             _hw = e.wheelRotation
121         else

```

```

122         _vw = e.wheelRotation
123         action(this@Mouse)
124     }
125 }
126 window.addMouseListener(ml)
127 window.addMouseMotionListener(mml)
128 window.addMouseWheelListener(mwl)
129 }
130 }

```

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

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

```

1 package org.phdeh.a.gui
2
3 import java.util.*
4
5 class NameLobby {
6     private var letter = 0
7     private var number = 0
8     private val stack = Stack<String>()
9
10    fun getName(): String {
11        if (stack.isEmpty())
12            return stack.pop()
13        val cl = (letter + 'A'.toInt()).toChar()
14        val name = if (number == 0)
15            "$cl"
16        else
17            "$cl$number"
18
19        if (cl == 'Z') {
20            letter = 0
21            number++
22        } else
23            letter++
24
25        return name
26    }
27
28    fun returnName(name: String) {
29        stack.push(name)
30    }
31 }

```

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

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

```

1 package org.phdeh.a.gui
2
3 import java.awt.Color
4 import java.awt.Graphics2D
5
6 data class Node(
7     var name: String,
8     var x: Double,
9     var y: Double,
10    val win: Window,
11    var tx: Double = 0.0,
12    var ty: Double = 0.0
13) : ScreenObject {

```

```

14     val width = 50
15     val height = 50
16
17     override fun isVisible(r: Screen): Boolean {
18         return Math.abs(x - r.x) < (r.width + width) / 2 &&
19             Math.abs(y - r.y) < (r.height + height) / 2
20     }
21
22     override fun draw(g: Graphics2D, r: Screen) {
23         tx = Math.round(x / win.squareSize.toDouble() * win.squareSize
24         ty = Math.round(y / win.squareSize.toDouble() * win.squareSize
25
26         val alpha = Math.max(160 - (Math.sqrt(
27             (tx - x) * (tx - x) + (ty - y) * (ty - y)
28         ) / win.squareSize * 255).toInt(), 0)
29         g.color = Color(0, 0, 0, alpha)
30         g.fillOval(
31             (tx - r.x).toInt() - (width - r.width) / 2,
32             (ty - r.y).toInt() - (height - r.height) / 2,
33             width,
34             height
35         )
36         if (win.lastTouch === this)
37             g.color = Color.RED
38         else if (win.prevTouch === this)
39             g.color = Color.RED
40         else
41             g.color = Color.BLUE
42         g.fillOval(
43             (x - r.x).toInt() - (width - r.width) / 2,
44             (y - r.y).toInt() - (height - r.height) / 2,
45             width,
46             height
47         )
48         g.color = Color.WHITE
49         g.drawString(
50             name,
51             (x - r.x).toInt() + (r.width) / 2
52                 - g.getFontMetrics().stringWidth(name) / 2,
53             (y - r.y).toInt() + (r.height + 28) / 2
54         )
55     }
56 }

```

Листинг 12 – main/org/phdeh/a/gui/Node.kt  
 Листинг 13 – main/org/phdeh/a/gui/Screen.kt

```

1 package org.phdeh.a.gui
2
3 data class Screen(
4     private val camera: Vector,
5     private val window: Window
6 ) {
7     val x get() = camera.x
8     val y get() = camera.y
9
10    val width get() = window.width
11    val height get() = window.height
12 }

```

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

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

```
1 package org.phdeh.a.gui
2
3 import java.awt.Graphics2D
4 import java.awt.Rectangle
5
6 interface ScreenObject {
7     fun draw(g: Graphics2D, r: Screen)
8
9     fun isVisible(r: Screen): Boolean
10 }
```

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

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

```
1 package org.phdeh.a.gui
2
3 data class Vector(
4     var x: Double = 0.0,
5     var y: Double = 0.0
6 )
```

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

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

```
1 package org.phdeh.a.gui
2
3 import org.phdeh.a.astar.AStar
4 import org.phdeh.a.astar.Graph
5 import org.phdeh.a.astar.forEachPair
6 import java.awt.*
7 import java.awt.Color.*
8 import javax.swing.JFrame
9 import java.awt.image.BufferedImage
10 import java.awt.Rectangle
11 import java.awt.event.ComponentEvent
12 import java.awt.event.ComponentAdapter
13 import java.awt.RenderingHints
14 import java.awt.event.KeyEvent
15
16 fun main() {
17     Window.isVisible = true
18 }
19
20 object Window : JFrame("AStar Test") {
21
22     @Volatile
23     var renderBuffer = BufferedImage(1, 1, BufferedImage.TYPE_3BYTE_BGR)
24
25     val nodeFont = Font("Helvetica", Font.BOLD, 40)
26
27     val cursorOnScreen = Vector()
28
29     val camera = Vector()
30     val screen = Screen(camera, this)
31
32     val edges = mutableListOf<Edge>()
33     val nodes = mutableListOf<Node>()
34
35     val nameLobby = NameLobby()
36 }
```

```

37     var imaginaryNode = null as Node?
38     var currentNode = null as Node?
39     var lastTouch = null as Node?
40     var prevTouch = null as Node?
41     var solution = false
42
43     val lineColor = Color(160, 128, 255, 100)
44     val squareSize = 75
45     val deleteCorner = 100
46     val lineWidth = 6
47     val selectDistance = 25
48     val wheelMultiplier = 5
49
50     var state = WindowState.NONE
51
52     init {
53         setSize(500, 500)
54         centreWindow()
55         initResizeListener()
56         renderBuffer.graphics.fontMetrics.getLineMetrics("", renderBuffer.graphics)
57     }
58
59     val keyboardHandler = Keyboard(this) {
60         var shouldRepaint = false
61         if (it.remove)
62             if (lastTouch != null || prevTouch != null) {
63                 clearSolution()
64                 lastTouch = null
65                 prevTouch = null
66                 shouldRepaint = true
67             }
68         if (it.released(KeyEvent.VK_SPACE)) {
69             synchronized(this@Window) {
70                 val start = lastTouch
71                 val stop = prevTouch
72
73                 if (start == null || stop == null)
74                     return@synchronized
75
76                 val graph = Graph.build { graph ->
77                     val vertices = mutableMapOf<Node, Graph.GraphBuilder.VertexBuilder
78                         ↳ >()
79                     nodes.forEach {
80                         vertices[it] = graph(it.name, it.x, it.y).vertexBuilder
81                     }
82                     edges.forEach {
83                         val from = vertices[it.from]
84                         val to = vertices[it.to]
85                         if (from != null && to != null)
86                             from - to
87                     }
88                 }
89
90                 val paths = AStar.findAllPaths(graph[start.name], graph[stop.name])
91                 paths.forEach {
92                     it.forEachPair { a, b ->
93                         edges.forEach {
94                             if (it.from.name == a.name && it.to.name == b.name ||

```

```

94         it.from.name == b.name && it.to.name == a.name)
95         it.shortest = true
96     }
97 }
98 }
99     solution = true
100     shouldRepaint = true
101 }
102 }
103 if (shouldRepaint)
104     repaint()
105 }
106
107 val mouseHandler = Mouse(this) {
108     var shouldRepaint = false
109
110     if (it.horizontalWheel != 0 || it.verticalWheel != 0)
111         shouldRepaint = true
112     camera.x += it.horizontalWheel * wheelMultiplier
113     camera.y += it.verticalWheel * wheelMultiplier
114
115     if (it.leftButtonPressed) {
116         clearSolution()
117         val node = findCursorNode(it.x, it.y)
118         if (keyboardHandler.command) {
119             if (node != null) {
120                 prevTouch = lastTouch
121                 lastTouch = node
122                 shouldRepaint = true
123             }
124         } else {
125             if (node == null) {
126                 state = WindowState.SET_VERTEX
127                 imaginaryNode = Node(
128                     nameLobby.getName(),
129                     it.x + camera.x, it.y + camera.y, this
130                 )
131             } else {
132                 state = WindowState.SET_VERTEX
133                 nodes.removeAt(nodes.indexOf(node))
134                 imaginaryNode = node
135             }
136             shouldRepaint = true
137         }
138     }
139
140     if (it.rightButtonPressed) {
141         val node = findCursorNode(it.x, it.y)
142         if (node != null)
143             clearSolution()
144         else
145             removeEdgesAt(it.x, it.y)
146         currentNode = node
147         shouldRepaint = true
148     }
149
150     val cn = currentNode
151     if (cn != null) {

```

```

152         if (it.rightButtonReleased) {
153             val node = findCursorNode(it.x, it.y)
154             if (node != null) {
155                 edges += Edge(node, cn, this)
156             }
157             currentNode = null
158         }
159         shouldRepaint = true
160     }
161
162
163     val imn = imaginaryNode
164     if (imn != null) {
165         imn.x = it.x + camera.x - width / 2
166         imn.y = it.y + camera.y - height / 2
167         shouldRepaint = true
168     }
169
170     if (it.leftButtonReleased && imn != null) {
171         val close = findGlobalNode(imn.tx.toInt(), imn.ty.toInt())
172         if (close == null &&
173             it.x >= -deleteCorner &&
174             it.y >= -deleteCorner &&
175             it.x <= width + deleteCorner &&
176             it.y <= height + deleteCorner
177         ) {
178             imn.x = imn.tx
179             imn.y = imn.ty
180             nodes += imn
181         } else {
182             edges.removeAll { it.to === imn || it.from === imn }
183             nameLobby.returnName(imn.name)
184         }
185         imaginaryNode = null
186         shouldRepaint = true
187     }
188
189     if (shouldRepaint)
190         repaint()
191 }
192
193 fun draw(g: Graphics2D) {
194     listOf(edges, nodes).forEach {
195         it.forEach {
196             if (it.isVisible(screen))
197                 it.draw(g, screen)
198         }
199     }
200     val imn = imaginaryNode
201     if (imn != null)
202         imn.draw(g, screen)
203     drawCurrentEdge(g)
204 }
205
206 fun drawCurrentEdge(g: Graphics2D) {
207     val r = screen
208     val from = currentNode
209     if (from === null)

```

```

210         return
211     val to = mouseHandler
212     g.stroke = BasicStroke(6f)
213     g.color = Color.BLACK
214     val dx = (-r.width / 2 + r.x).toInt()
215     val dy = (-r.height / 2 + r.y).toInt()
216     val a = Math.atan2(to.y + dy - from.y, to.x + dx - from.x)
217     g.drawLine(
218         from.x.toInt() - dx + (Math.cos(a) * from.width / 2).toInt(),
219         from.y.toInt() - dy + (Math.sin(a) * from.height / 2).toInt(),
220         to.x.toInt(),
221         to.y.toInt()
222     )
223 }
224
225 fun clearSolution() {
226     if (solution) {
227         edges.forEach { it.shortest = false }
228         solution = false
229     }
230 }
231
232 fun findCursorNode(x: Int, y: Int): Node? {
233     return findGlobalNode(
234         (x - width / 2 + camera.x).toInt(),
235         (y - height / 2 + camera.y).toInt()
236     )
237 }
238
239 fun findGlobalNode(x: Int, y: Int): Node? {
240     var close = null as Node?
241     nodes.forEach { i ->
242         if ((x - i.x) * (x - i.x) +
243             (y - i.y) * (y - i.y) <=
244                 selectDistance * selectDistance
245             ) {
246             close = i
247         }
248     }
249     return close
250 }
251
252 fun removeEdgesAt(x: Int, y: Int) {
253     // val r = screen
254     //
255     // val dx = (-r.width / 2 + r.x).toInt()
256     // val dy = (-r.height / 2 + r.y).toInt()
257     //
258     // val delete = mutableListOf<Edge>()
259     //
260     // edges.forEach {
261     //     val x1 = it.from.x.toInt() - dx
262     //     val y1 = it.from.y.toInt() - dy
263     //
264     //     val x2 = it.to.x.toInt() - dx
265     //     val y2 = it.to.y.toInt() - dy
266     //
267     //     val x3 = x2 - x1

```



```

268 //          val y3 = y2 - y1
269 //
270 //          val a = Math.atan2(it.to.y - it.from.y, it.to.x - it.from.x)
271 //
272 //          val mx = x - x1
273 //          val my = y - y2
274 //
275 //          val mx1 = mx * Math.cos(a) + my * Math.sin(a)
276 //          val my1 = mx * Math.sin(a) + my * Math.cos(a)
277 //
278 //          cursorOnScreen.x = mx.toDouble()
279 //          cursorOnScreen.y = my.toDouble()
280 //          println("$mx1:$my1")
281 //          println("$mx:$my")
282 //
283 //          val x5 = Math.sqrt(x3 * x3 + y3 * y3 + 0.0) / 2
284 //
285 //          if (Math.abs(my1) < 10 && Math.abs(mx1 - x5) < x5)
286 //              delete += it
287 //      }
288 //
289 //      edges.removeAll { it in delete }
290 }
291
292 //
293
294 override fun paint(g: Graphics) {
295     val r = renderBuffer
296     val g2 = r.graphics as Graphics2D
297     g2.color = WHITE
298     g2.fillRect(0, 0, width, height)
299     var rh = RenderingHints(
300         RenderingHints.KEY_TEXT_ANTIALIASING,
301         RenderingHints.VALUE_TEXT_ANTIALIAS_ON
302     )
303     g2.setRenderingHints(rh)
304     rh = RenderingHints(
305         RenderingHints.KEY_ANTIALIASING,
306         RenderingHints.VALUE_ANTIALIAS_ON
307     )
308     g2.color = lineColor
309     val linesHorizontally = (width / squareSize) / 2 + 2
310     val linesVertically = (height / squareSize) / 2 + 2
311     for (x in -linesHorizontally..linesHorizontally)
312         g2.fillRect(
313             (-camera.x % squareSize).toInt() - lineWidth / 2
314             + x * squareSize - squareSize + width / 2, 0, lineWidth,
315             ↵ height
316         )
317     for (y in -linesVertically..linesVertically)
318         g2.fillRect(
319             0, (-camera.y % squareSize).toInt() - lineWidth / 2
320             + y * squareSize - squareSize + height / 2, width, lineWidth
321         )
322     g2.font = nodeFont
323     g2.setRenderingHints(rh)
324     draw(g2)
325     g2.color = BLACK

```

```

325 //      g2.fillRect(cursorOnScreen.x.toInt() - 25, cursorOnScreen.y.toInt() - 25, 50, 50)
326 g.drawImage(r, 0, 0, this)
327 }
328
329 override fun setSize(width: Int, height: Int) {
330     handleResize(width, height)
331     super.setSize(width, height)
332 }
333
334 override fun setSize(d: Dimension) {
335     handleResize(d.width, d.height)
336     super.setSize(d)
337 }
338
339 override fun setBounds(r: Rectangle) {
340     handleResize(r.width, r.height)
341     super.setBounds(r)
342 }
343
344 override fun setBounds(x: Int, y: Int, width: Int, height: Int) {
345     handleResize(width, height)
346     super.setBounds(x, y, width, height)
347 }
348
349 fun initResizeListener() {
350     addComponentListener(object : ComponentAdapter() {
351         override fun componentResized(evt: ComponentEvent?) {
352             handleResize(width, height)
353         }
354     })
355 }
356
357 fun handleResize(width: Int, height: Int) {
358     if (width > 0 && height > 0) {
359         renderBuffer = BufferedImage(width, height, BufferedImage.TYPE_3BYTE_BGR)
360     }
361 }
362
363 fun centreWindow() {
364     val dimension = Toolkit.getDefaultToolkit().getScreenSize()
365     val x = ((dimension.getWidth() - this.width) / 2).toInt()
366     val y = ((dimension.getHeight() - this.height) / 2).toInt()
367     this.setLocation(x, y)
368 }
369 }

```

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

```

1 package org.phdeh.a.gui
2
3 enum class WindowState {
4     NONE,
5     SET_VERTEX,
6     MOVE_VERTEX,
7     CONNECT_VERTEX
8 }

```

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

## Тесты

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

```
1 package org.phdeh.a.astar
2
3 import org.junit.Test
4
5 internal class AStarMultiplePathTest {
6     @Test
7     fun multiplePaths() {
8         val graph = Graph.build {
9             val a by it(1.0, 0.0)
10            val b by it(0.0, 1.0)
11            val c by it(-1.0, 0.0)
12            val d by it(0.0, -1.0)
13
14            a - b - c - d - a
15        }
16
17        AStar.findAllPaths(graph["a"], graph["c"]) shouldBe listOf(
18            listOf(graph["a"], graph["b"], graph["c"]),
19            listOf(graph["a"], graph["d"], graph["c"])
20        )
21    }
22 }
```

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

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

```
1 package org.phdeh.a.astar
2
3 import org.junit.Test
4
5 internal class AStarTest {
6
7     @Test
8     fun findPath() {
9         val graph = Graph.build {
10             val a by it(1.0, 0.0)
11             val b by it(0.0, 1.0)
12             val c by it(-1.0, 0.0)
13
14             a - b - c
15         }
16
17         AStar.findPath(graph["a"], graph["c"]) shouldBe listOf(graph["a"], graph["b"],
18             ↪ graph["c"])
19     }
20
21     @Test
22     fun findShortestPath() {
23         val graph = Graph.build {
24             val a by it(1.0, 0.0)
25             val b by it(0.0, 2.0)
26             val c by it(0.0, 1.0)
27             val d by it(-1.0, 0.0)
28
29             a - b - d - c - a
30         }
31     }
32 }
```

```

31     AStar.findPath(graph["a"], graph["d"]) shouldBe listOf(graph["a"], graph["c"],
32     ↪ graph["d"])
33 }
34 @Test
35 fun wayDoesntExist() {
36     val graph = Graph.build {
37         val a by it(1.0, 0.0)
38         val b by it(0.0, 2.0)
39         val c by it(0.0, 1.0)
40         val d by it(-1.0, 0.0)
41
42         a - b; c - d
43     }
44
45     AStar.findPath(graph["a"], graph["d"]) shouldBe listOf<Graph.Vertex>()
46 }
47
48 @Test
49 fun fromAToA() {
50     val graph = Graph.build {
51         val a by it(1.0, 0.0)
52         val b by it(0.0, 2.0)
53         val c by it(0.0, 1.0)
54         val d by it(-1.0, 0.0)
55
56         a - b - d - c - a
57     }
58
59     AStar.findPath(graph["a"], graph["a"]) shouldBe listOf(graph["a"])
60 }
61
62 @Test(expected = AStar.VerticeBelongToDifferentGraphsException::class)
63 fun differentGraphs() {
64     val graph1 = Graph.build {
65         val a by it(1.0, 0.0)
66
67         a
68     }
69     val graph2 = Graph.build {
70         val a by it(1.0, 0.0)
71
72         a
73     }
74
75     AStar.findPath(graph1["a"], graph2["a"])
76 }
77
78 @Test(expected = AStar.VertexDoesntExistException::class)
79 fun nonExistingVerticesFrom() {
80     val graph = Graph.build {
81         val a by it(1.0, 0.0)
82
83         a
84     }
85
86     AStar.findPath(graph["d"], graph["a"])
87 }

```

```

88
89     @Test(expected = AStar.VertexDoesntExistException::class)
90     fun nonExistingVerticesTo() {
91         val graph = Graph.build {
92             val a by it(1.0, 0.0)
93
94             a
95         }
96
97         AStar.findPath(graph["a"], graph["d"])
98     }
99
100     @Test(expected = AStar.VertexDoesntExistException::class)
101     fun nonExistingVerticesBothFromAndTo() {
102         val graph = Graph.build {
103             val a by it(1.0, 0.0)
104
105             a
106         }
107
108         AStar.findPath(graph["c"], graph["d"])
109     }
110 }

```

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

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

```

1 package org.phdeh.a.astar
2
3 import org.junit.Test
4
5 internal class GraphTest {
6     @Test
7     fun emptyGraphTest() {
8         val graph = Graph.build { }
9         graph.size shouldBe 0
10    }
11
12    @Test
13    fun correctValues() {
14        val graph = Graph.build {
15            val a by it(1.0, -1.0)
16
17            a
18        }
19        graph["a"].name shouldBe "a"
20        graph["a"].toString() shouldBe "a"
21        graph["a"].x shouldBe 1.0
22        graph["a"].y shouldBe -1.0
23    }
24    @Test
25    fun explicitNaming() {
26        val graph = Graph.build {
27            val n1 by it("Moscow", 1.0, -1.0)
28            val n2 by it("Saint-Petersburg", 1.0, -1.0)
29
30            n1 - n2
31        }
32        ("Moscow" in graph) shouldBe true
33        ("n1" in graph) shouldNotBe true

```

```

34     }
35
36     @Test
37     fun duplicatingName() {
38         val graph = Graph.build {
39             val node1 by it("Node", 1.0, -1.0)
40             val node2 by it("Node", 1.0, -1.0)
41             val node3 by it("Node", 1.0, -1.0)
42
43             node1 - node2 - node3
44         }
45         ("Node" in graph) shouldBe true
46         ("Node_2" in graph) shouldBe true
47         ("Node_3" in graph) shouldBe true
48         ("Node_4" in graph) shouldNotBe true
49     }
50
51     @Test
52     fun containsNameA() {
53         val graph = Graph.build {
54             val a by it(1.0, -1.0)
55
56             a
57         }
58         ("a" in graph) shouldBe true
59     }
60
61     @Test
62     fun aConnectedToB() {
63         val graph = Graph.build {
64             val a by it(1.0, 1.0)
65             val b by it(-1.0, -1.0)
66
67             a - b
68         }
69         graph["a"] hasEdgeTo graph["b"] shouldBe true
70     }
71
72     @Test
73     fun commutativeConnection() {
74         val graph = Graph.build {
75             val a by it(1.0, 1.0)
76             val b by it(-1.0, -1.0)
77
78             b - a
79         }
80         graph["a"] hasEdgeTo graph["b"] shouldBe true
81     }
82
83     @Test
84     fun aNotConnectedToB() {
85         val graph = Graph.build {
86             val a by it(1.0, 0.0)
87             val b by it(-1.0, 0.0)
88             val c by it(0.0, 1.0)
89
90             a - c - b
91         }

```

```

92         graph["a"] hasEdgeTo graph["b"] shouldBe false
93     }
94
95     @Test
96     fun indices() {
97         val graph = Graph.build {
98             val a by it(1.0, 0.0)
99             val b by it(-1.0, 0.0)
100             val c by it(0.0, 1.0)
101
102             a - c - b
103         }
104         graph[0] shouldBe graph["a"]
105         graph[1] shouldBe graph["b"]
106         graph[2] shouldBe graph["c"]
107         graph["a"].ordinal shouldBe 0
108         graph["b"].ordinal shouldBe 1
109         graph["c"].ordinal shouldBe 2
110     }
111
112     @Test
113     fun backtracking() {
114         val graph = Graph.build {
115             val a by it(1.0, 0.0)
116
117             a
118         }
119         graph["a"].graph shouldBe graph
120         graph["b"].graph shouldNotBe graph
121         graph["c"].graph shouldNotBe graph
122         graph["b"].graph shouldNotBe graph["c"].graph
123     }
124
125     @Test
126     fun dDoesntExists() {
127         val graph = Graph.build {
128             val a by it(1.0, 0.0)
129             val b by it(-1.0, 0.0)
130             val c by it(0.0, 1.0)
131
132             a - c - b
133         }
134         (graph["d"] !in graph) shouldBe true
135     }
136
137     @Test
138     fun distanceTest() {
139         val graph = Graph.build {
140             val a by it(1.0, 0.0)
141             val b by it(-1.0, 0.0)
142
143             a - b
144         }
145         (graph["a"] distanceTo graph["b"]) shouldBe 2.0
146     }
147
148     @Test
149     fun impossibleDistanceTest() {

```

```

150     val graph = Graph.build {
151         val a by it(1.0, 0.0)
152
153         a
154     }
155     (graph["a"] distanceTo graph["d"]).isNaN() shouldBe true
156 }
157
158 @Test
159 fun edgesAreCorrect() {
160     val graph = Graph.build {
161         val a by it(1.0, 0.0)
162         val b by it(-1.0, 0.0)
163         val c by it(0.0, 1.0)
164         val d by it(0.0, 2.0)
165
166         a - b - c - a; b - d - c
167     }
168     val edges = graph["a"].edgesTo
169     (graph["b"] in edges) shouldBe true
170     (graph["c"] in edges) shouldBe true
171     (graph["d"] in edges) shouldNotBe true
172     (graph["e"] in edges) shouldNotBe true
173 }
174
175 @Test
176 fun nonExistingVerticesAreNotEqual() {
177     val graph = Graph.build {}
178     (graph["a"] != graph["b"]) shouldBe true
179 }
180
181 @Test
182 fun sizeOfGraph() {
183     val graph = Graph.build {
184         val a by it(1.0, 0.0)
185         val b by it(-1.0, 0.0)
186         val c by it(0.0, 1.0)
187
188         a - b - c - a
189     }
190     graph.size shouldBe 3
191 }
192 }

```

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

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

```

1 package org.phdeh.a.astar
2
3 import org.junit.Test
4
5 internal class PriorityQueueTest {
6     @Test
7     fun emptyQueue() {
8         val pq = PriorityQueue<String>()
9         pq.isEmpty() shouldBe true
10        pq.isNotEmpty() shouldNotBe true
11    }
12
13    @Test

```



```

14 fun notEmptyQueue() {
15     val pq = PriorityQueue<String>()
16     pq.push("Test", 1.0)
17     pq.isNotEmpty() shouldBe true
18     pq.isEmpty() shouldNotBe true
19 }
20
21 @Test
22 fun priorityTest() {
23     val pq = PriorityQueue<String>()
24     pq.push("Foo", 2.0)
25     pq.push("Bar", 1.0)
26     pq.push("Baz", 3.0)
27     pq.pop() shouldBe "Bar"
28     pq.pop() shouldBe "Foo"
29     pq.pop() shouldBe "Baz"
30 }
31
32 @Test
33 fun startValues() {
34     val pq = PriorityQueue<String>("Foo", "Bar", "Baz")
35     pq.pop() shouldBe "Foo"
36     pq.pop() shouldBe "Bar"
37     pq.pop() shouldBe "Baz"
38 }
39
40 @Test(expected = PriorityQueue.EmptyQueueException::class)
41 fun emptyQueuePop() {
42     val pq = PriorityQueue<String>()
43     pq.pop()
44 }
45 }

```

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

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

```

1 package org.phdeh.a.gui
2
3 import org.junit.Test
4 import org.phdeh.a.astar.shouldBe
5 import org.phdeh.a.astar.shouldNotBe
6 import java.awt.GraphicsEnvironment
7 import java.awt.Robot
8 import java.awt.event.InputEvent
9 import java.awt.event.KeyEvent
10
11 internal class WindowTest {
12     @Test
13     fun testGui() {
14         if (GraphicsEnvironment.isHeadless())
15             return
16
17         val wheeling = Window.squareSize * 2 / Window.wheelMultiplier
18         val robotTimer = 100L
19
20         Window.isVisible = true
21         Window.toFront()
22         val robot = Robot()
23         Thread.sleep(robotTimer)
24         robot.mouseMove(0, 0)

```

```

25 Thread.sleep(robotTimer)
26 robot.mouseMove(Window.x + Window.width / 2, Window.y + Window.height / 2)
27 Thread.sleep(robotTimer)
28 // Set first
29 robot.mousePress(InputEvent.BUTTON1_DOWN_MASK)
30 Thread.sleep(robotTimer)
31 robot.mouseRelease(InputEvent.BUTTON1_DOWN_MASK)
32 Thread.sleep(robotTimer)
33 // Move to second
34 robot.mouseWheel(wheeling)
35 Thread.sleep(robotTimer)
36 robot.keyPress(KeyEvent.VK_SHIFT)
37 Thread.sleep(robotTimer)
38 robot.mouseWheel(wheeling)
39 Thread.sleep(robotTimer)
40 robot.keyRelease(KeyEvent.VK_SHIFT)
41 Thread.sleep(robotTimer)
42 // Set second
43 robot.mousePress(InputEvent.BUTTON1_DOWN_MASK)
44 Thread.sleep(robotTimer)
45 robot.mouseRelease(InputEvent.BUTTON1_DOWN_MASK)
46 Thread.sleep(robotTimer * 2)
47 // Begin edge
48 robot.mousePress(InputEvent.BUTTON3_DOWN_MASK)
49 Thread.sleep(robotTimer)
50 // Return to first
51 robot.mouseWheel(-wheeling)
52 Thread.sleep(robotTimer)
53 robot.keyPress(KeyEvent.VK_SHIFT)
54 Thread.sleep(robotTimer)
55 robot.mouseWheel(-wheeling)
56 Thread.sleep(robotTimer)
57 robot.keyRelease(KeyEvent.VK_SHIFT)
58 Thread.sleep(robotTimer)
59 // End edge
60 robot.mouseRelease(InputEvent.BUTTON3_DOWN_MASK)
61 Thread.sleep(robotTimer)
62
63 Window.setSize(600, 400)
64 Window.centreWindow()
65
66 Thread.sleep(robotTimer)
67 // Set third
68 robot.mouseMove(
69     Window.x + Window.width / 2 + Window.squareSize * 2,
70     Window.y + Window.height / 2 - Window.squareSize * 2
71 )
72 Thread.sleep(robotTimer)
73 robot.mousePress(InputEvent.BUTTON1_DOWN_MASK)
74 Thread.sleep(robotTimer)
75 robot.mouseRelease(InputEvent.BUTTON1_DOWN_MASK)
76 Thread.sleep(robotTimer)
77 // Replacing third
78 robot.mousePress(InputEvent.BUTTON1_DOWN_MASK)
79 Thread.sleep(robotTimer)
80 robot.mouseMove(
81     Window.x + Window.width / 2 + Window.squareSize * 2,
82     Window.y + Window.height / 2 + Window.squareSize * 2

```

```

83     )
84     Thread.sleep(robotTimer)
85     robot.mouseRelease(InputEvent.BUTTON1_DOWN_MASK)
86     Thread.sleep(robotTimer)
87     // Begin edge
88     robot.mousePress(InputEvent.BUTTON3_DOWN_MASK)
89     Thread.sleep(robotTimer)
90     robot.mouseMove(Window.x + Window.width / 2, Window.y + Window.height / 2)
91     Thread.sleep(robotTimer)
92     // End edge
93     robot.mouseRelease(InputEvent.BUTTON3_DOWN_MASK)
94     Thread.sleep(robotTimer)
95     // Setting first way point
96     robot.keyPress(KeyEvent.VK_META)
97     Thread.sleep(robotTimer)
98     robot.mousePress(InputEvent.BUTTON1_DOWN_MASK)
99     Thread.sleep(robotTimer)
100    robot.mouseRelease(InputEvent.BUTTON1_DOWN_MASK)
101    Thread.sleep(robotTimer)
102
103    Window.setSize(1000, 1000)
104    Window.centreWindow()
105
106    Thread.sleep(robotTimer)
107    // Setting second way point
108    robot.mouseMove(
109        Window.x + Window.width / 2 - Window.squareSize * 2,
110        Window.y + Window.height / 2 - Window.squareSize * 2
111    )
112    Thread.sleep(robotTimer)
113    robot.mousePress(InputEvent.BUTTON1_DOWN_MASK)
114    Thread.sleep(robotTimer)
115    robot.mouseRelease(InputEvent.BUTTON1_DOWN_MASK)
116    Thread.sleep(robotTimer)
117    // Searching path
118    robot.keyRelease(KeyEvent.VK_META)
119    Thread.sleep(robotTimer)
120    robot.keyPress(KeyEvent.VK_SPACE)
121    Thread.sleep(robotTimer)
122    robot.keyRelease(KeyEvent.VK_SPACE)
123    Thread.sleep(robotTimer)
124
125    Window.edges.size shouldBe 2
126
127    Window.edges.forEach {
128        if (it.from.name == "A" && it.to.name == "B" || it.from.name == "B" && it.
129            ↪ to.name == "A")
130            it.shortest shouldBe true
131        else
132            it.shortest shouldNotBe true
133    }
134
135    Thread.sleep(robotTimer)
136    robot.keyPress(KeyEvent.VK_BACK_SPACE)
137    Thread.sleep(robotTimer)
138    robot.keyRelease(KeyEvent.VK_BACK_SPACE)
139    Thread.sleep(robotTimer)

```

```

140         Window.edges.size shouldBe 2
141
142         Window.edges.forEach {
143             it.shortest shouldNotBe true
144         }
145
146         // Removing B
147         Thread.sleep(robotTimer)
148         robot.mousePress(InputEvent.BUTTON1_DOWN_MASK)
149         Thread.sleep(robotTimer)
150         robot.mouseMove(0, 0)
151         Thread.sleep(robotTimer)
152         robot.mouseRelease(InputEvent.BUTTON1_DOWN_MASK)
153         Thread.sleep(robotTimer)
154
155         Window.edges.size shouldBe 1
156
157         Window.edges.forEach {
158             it.shortest shouldNotBe true
159         }
160
161
162         Thread.sleep(robotTimer * 5)
163     }
164 }

```

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