Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение высшего образования  
«Рыбинский государственный авиационный технический университет имени П. А. Соловьёва»

Факультет радиоэлектроники и информатики

Кафедра вычислительных систем

**КУРСОВАЯ РАБОТА**

по дисциплине  
«Тестирование и отладка програмного обеспечения»

Студент группы ИПБ-20: Батурин К. А.

Руководитель: Пруктишина В.А.

Рыбинск 2023

Содержание

[1. Описание тестируемой системы 4](#_Toc155770642)

[1.1. Функции приложения 5](#_Toc155770643)

[1.2. Используемые технологии 5](#_Toc155770644)

[2. Структурное тестирование 6](#_Toc155770645)

[2.1. Управляющий граф 6](#_Toc155770646)

[2.2. Тестирование по критериям C0, C1, C2 7](#_Toc155770647)

[2.2.1. Критерий тестирования C0 7](#_Toc155770648)

[2.2.2. Критерий тестирования C1 7](#_Toc155770649)

[2.2.3. Критерий тестирования C2 9](#_Toc155770650)

[3. Модульное тестирование 9](#_Toc155770651)

[3.1. Модули 9](#_Toc155770652)

[3.2. Тестирование 10](#_Toc155770653)

[4. Интеграционное тестирование 13](#_Toc155770654)

[4.1. Модули 13](#_Toc155770655)

[4.2. Тестирование 14](#_Toc155770656)

[5. Системное тестирование 15](#_Toc155770657)

[5.1. Функциональные требования 15](#_Toc155770658)

[5.2. Тестирование 16](#_Toc155770659)

[6. Вывод 20](#_Toc155770660)

[7. Приложение 21](#_Toc155770661)

[7.1. Код тестов модуля математики 21](#_Toc155770662)

[7.2. Код тестов модуля генерации фонового изображения капчи 21](#_Toc155770663)

[7.3. Код тестов модуля генерации текстового растра 23](#_Toc155770664)

[7.4. Код тестов модуля загрузки ini-файлов 24](#_Toc155770665)

[7.5. Код теста вспомогательного функционала 25](#_Toc155770666)

[7.6. Конфигурационный файл для GitHub actions 25](#_Toc155770667)

# Описание тестируемой системы

Для проведения тестирования была выбрана программа, реализованная для курсовой работы на тему «Реализация CAPTCHA» по предмету Объектно-ориентированное программирование. Данная программа представляет собой оконное приложение, которое позволяет просматривать и решать различные реализованные виды капчи.

Ссылка на GitHub: <https://github.com/KiRiLL2001REAL/CaptchaTD>

Вид типичных окон приложения представлен на рисунках 1, 2 и 3.

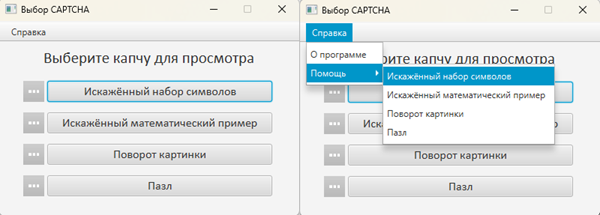


Рисунок 1 – вид главного окна приложения (хаба)

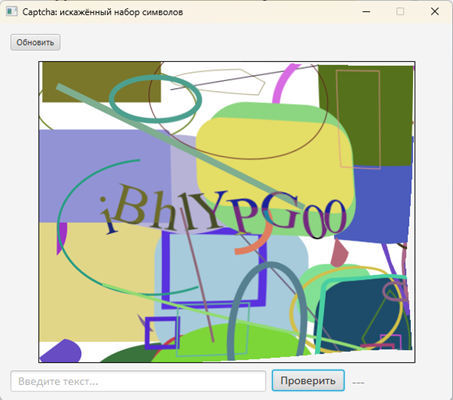


Рисунок 2 – вид одного из окон капчи

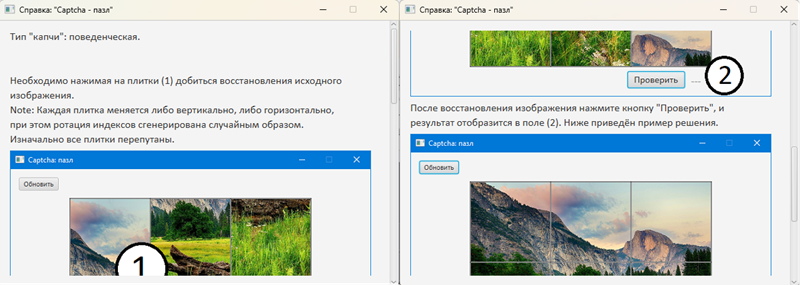


Рисунок 3 – вид одного из окон справки

## Функции приложения

* решение графической капчи (с искажённым набором символов);
* решение логически-графической капчи (с искажённым математическим примером);
* решение поведенческой капчи (с поворотом изображения и с решением паззла);
* просмотр сведений о программе;
* просмотр справки по решению вышеуказанных видов капчи.

## Используемые технологии

Разработка программного обеспечения производилась на языке Java с использованием графической библиотеки JavaFX.

Справка конструируется по описаниям в ini-файлах, работа с которыми производится посредством библиотеки org.ini4j.

# Структурное тестирование

## Управляющий граф

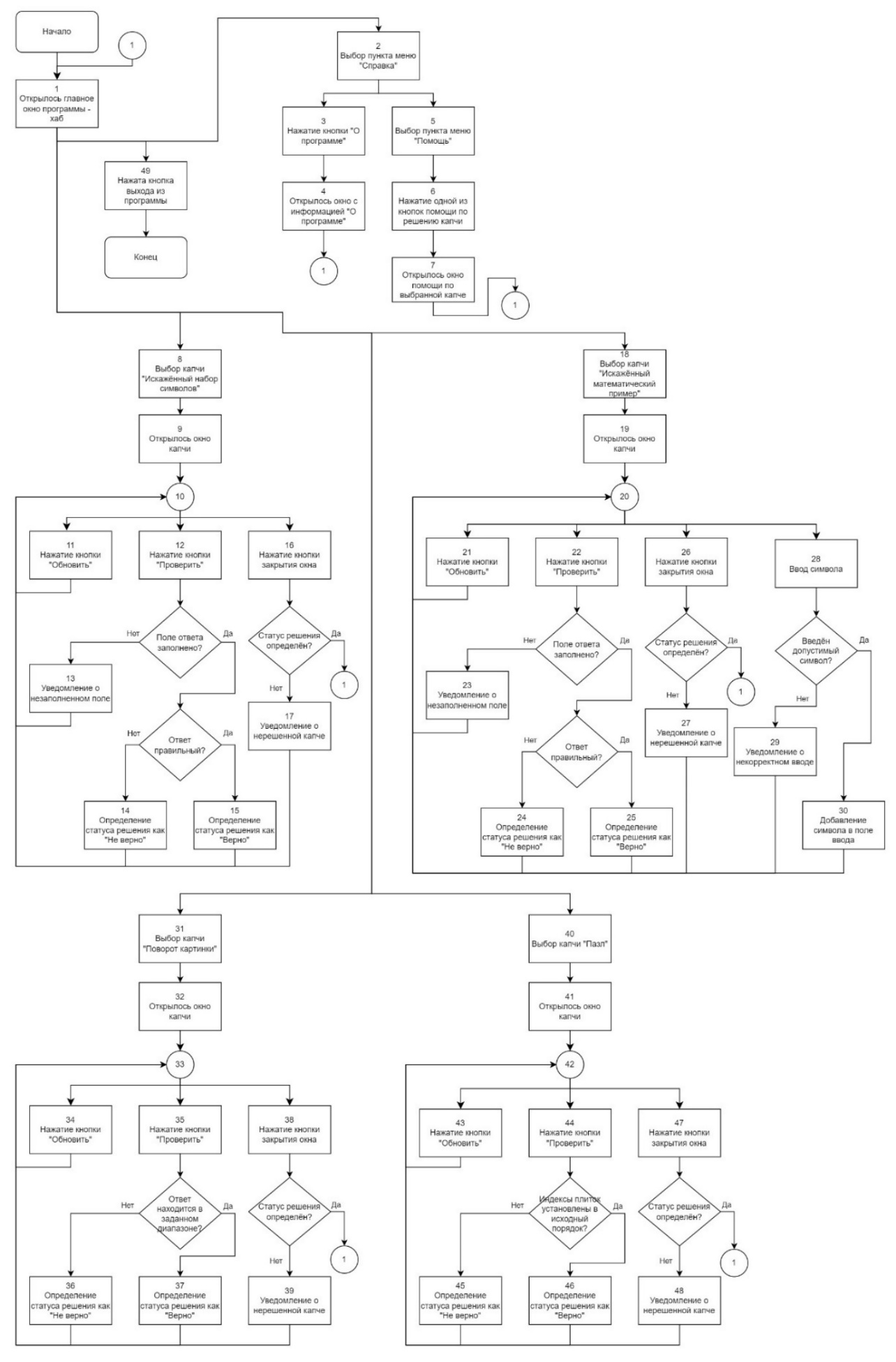


Рисунок – управляющий граф программы «Captcha»

## Тестирование по критериям C0, C1, C2

Кртерии тестирования основаны на управляющем графе программы, представленном на рисунке 4.

### Критерий тестирования C0

Критерий С0 – набор тестов в совокупности должен обеспечить прохождение каждой команды не менее 1 раза.

1-2-3-4-1-2-5-6-7-1-8-9-10-11-10-12-13-10-16-17-10-12-14-10-12-15-10-16-1-18-19-20-21-20-22-23-20-26-27-20-28-29-20-28-30-20-22-24-20-28-30-20-22-25-20-26-1-31-32-33-34-33-38-39-33-35-36-33-35-37-33-38-1-40-41-42-43-42-47-48-42-44-45-42-44-46-42-47-1-45

### Критерий тестирования C1

Критерий С1 – набор тестов в совокупности должен обеспечить прохождение каждой ветви не менее 1 раза.

1-49

1-2

1-2-3

1-2-5

1-8-9-10

9-10-11

9-10-12

12-13

12-14

12-15

9-10-16

16-17

16-1

1-18

19-20

19-20-21

19-20-22

22-23

22-24

22-25

19-20-26

26-27

26-1

19-20-28

28-29

28-30

1-31

32-33

32-33-34

32-33-35

35-36

35-37

32-33-38

38-39

38-1

1-40

41-42

41-42-43

41-42-44

44-45

44-46

41-42-47

47-48

47-1

### Критерий тестирования C2

Критерий С2 – набор тестов в совокупности должен обеспечить прохождение каждого пути не менее одного раза. Если программа содержит цикл (в особенности с неявно заданным числом итераций), то число итераций ограничивается константой.

1-49

1-2-3-4

1-2-5-6-7

1-8-9-10-12-13-10-16-17-10-12-15-10-16

1-8-9-10-12-15-10-11-10-12-14-10-16

1-8-9-10-16-17-10-12-14

1-18-19-20-28-30-20-22-24-20-26

1-18-19-20-28-29-20-22-23-20-28-30-20-26-27-20-22-25-20-26

1-18-19-20-26-27-20-28-30-20-22-25-20-26

1-18-19-20-28-30-20-22-25-20-21-20-26-27-20-28-30-20-22-24-20-26

1-31-32-33-35-36-33-38

1-31-32-33-38-39-33-35-37-33-38

1-31-32-33-38-39-33-35-36-33-34-33-38-39-33-35-37-33-38

1-40-41-42-44-45-42-47

1-40-41-42-47-48-42-44-46-42-47

1-40-41-42-47-48-42-44-45-42-43-42-47-48-42-44-46-42-47

# Модульное тестирование

## Модули

* Модуль математики

cw.utils.imgenerator.AdditionalMath.java

Необходимо убедиться, что методы работают исправно, иначе могут возникнуть проблемы с генерацией изображений к «искажённому набору символов» и «искажённому математическому примеру».

* Вспомогательные модули для работы с изображениями

cw.utils.imgenerator.ImageUtils.java

cw.utils.imgenerator.ImageTextUtils.java

Необходимо проверить работоспособность функций, отвечающих за модификацию изображений (рисование примитивов, растеризацию текста), а также функций генерации и смешения цветов.

* Модуль загрузки ini-файлов

cw.utils.HelpConfiguration.java

Будет выполнена проверка хелпер-класса, используемого для получения информации о контенте, отображаемого в справочных окнах программы.

## Тестирование

При тестировании использовалась зависимость JUnit4 версии 4.12. В некоторых случаях применялась в том числе библиотека TestFX версии 4.0.17.

* Модуль математики

Для данного модуля написано 3 теста: тест на маппинг числа из одного числового диапазона в другой, а также тесты нахождения евклидового расстояния между двумя точками в двумерном и трёхмерном пространстве. Код приведён в приложении 8.1. Как можно видеть на рисунке 5, тесты успешно пройдены, а значит, модуль в целом работает корректно.

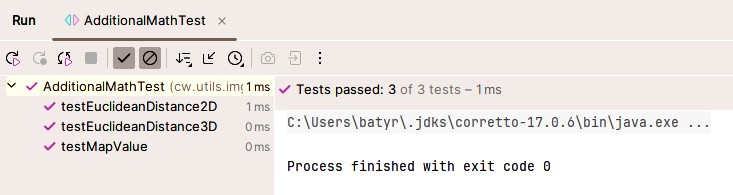


Рисунок – результат тестирования модуля математики

* Модули для работы с изображениями
  + Модуль генерации фонового изображения капчи

cw.utils.imgenerator.ImageUtils.java

Для данного модуля выполнено тестирование: билинейной интерполяции цвета, генерации случайного цвета, генерации нескольких уникальных цветов, функций смешения, рисования примитивов.

Стоит раскрыть некоторые подробности реализации тестирования функции отрисовки графических примитивов на изображении. Используемая графическая библиотека JavaFX допускает вызовы отрисовки только внутри так называемого «fx» потока. Чтобы данный поток был создан при проведении тестов, необходимо было подключить особую зависимость TestFX, в рамках которой предполагается наследование класса от ApplicationTest, а также добавление статического метода setUpClass, в котором инициируется запуск приложения. Код тестов приведён в приложении 8.2, результат тестирования приведён ниже, на рисунке 6.

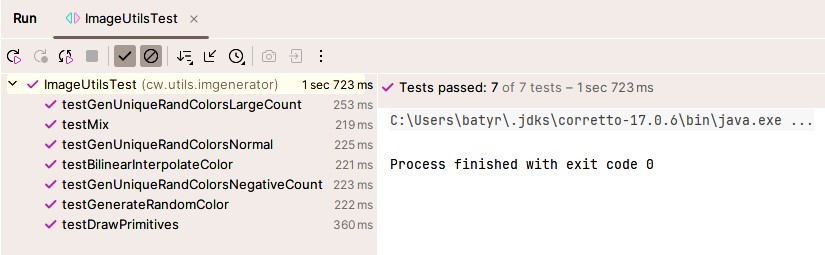


Рисунок – результат тестирования модуля генерации фонового изображения капчи

* + Модуль генерации текстового растра

cw.utils.imgenerator.ImageTextUtils.java

Данный модуль содержит в себе 2 функции: метод создания растрированного текста, и метод отображения этого текста на другое изображение. Для них написаны тесты в соответствующем количестве. Как и в случае с предыдущим тестировочным модулем, класс наследуется от ApplicationTest. Код приведён в приложении 8.3, езультат тестирования приведён на рисунке 7.

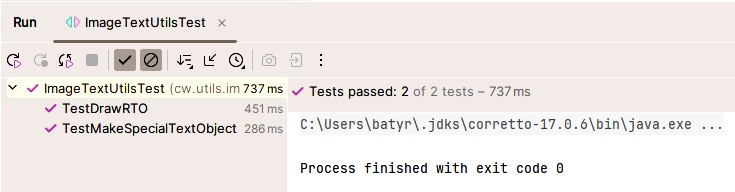


Рисунок – результат тестирования модуля генерации текстового растра

* Модуль загрузки ini-файлов

Для этого модуля было написано 3 unit-теста. По сути, данный класс является хранилищем данных, из которого извлекается информация для построения окна помощи. Проверяется соответствие заголовка, равенство количества строк текста / их индексов, а также количества путей к изображениям / их индексов. Код тестов представлен в приложении 8.4, результаты тестирования приведены на рисунке 8. В данном тестовом классе также присутствует интеграционный тест, который будет рассмотрен в следующем разделе.

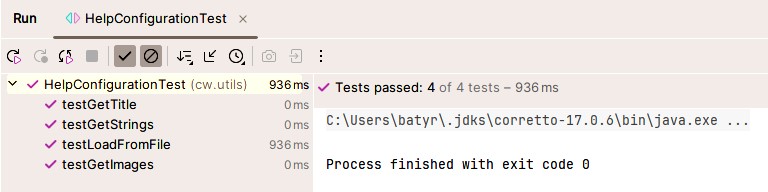


Рисунок – результат тестирования модуля загрузки ini-файлов

* Вспомогательный модуль для тестирования

Тестирование функций, связанных с графической составляющей, осуществляется таким образом: выбирается случайный цвет, проверяется, что данным цветом ещё ничего не нарисовано на экране, затем примитив / текст отображается и выполняется проверка наличия нового цвета на изображении. JavaFX не предлагает особых функций, способных быстро сравнивать цвет выбранной точки с искомой, однако есть функция его получения в байтовом формате ARGB (число типа int). Проведя некоторое исследование, было выяснено, что хеш-код компонента Color составляется из RGBA представления.

Основываясь на этом знании, была реализована вспомогательная функция, переводящая цвет формата ARGB в хеш путём левой ротации крайнего байта на 1 позицию. Чтобы быть уверенным в эквивалентности встроенного и разработанного метода получения хеш-кода, написан тест. Его текст представлен в приложении 8.5, результат выполения представлен на рисунке 9.

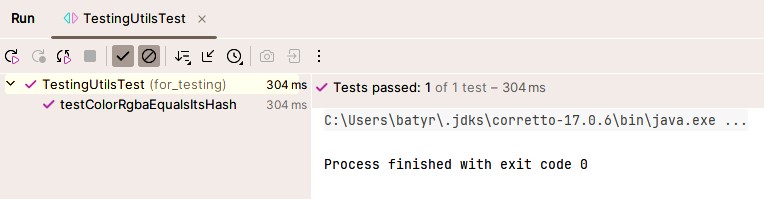


Рисунок – результат тестирования вспомогательного функционала

# Интеграционное тестирование

## Модули

* Модуль загрузки ini-файлов

Необходимо проверить корректность взаимодействия модуля с системой ввода / вывода.

* Модуль математики, модули для работы с изображениями

Необходимо проверить, корректно ли зваимодействуют модули. Генератор фонового изображения использует модуль математики. На полученную картинку накладывается текст, каждая буква которого отклоняется от своего базового положения и поворачивается. При этом вновь используется модуль математики. В конце, к результирующему изображению применяется бочкообразная дисторсия.

## Тестирование

* Модуль загрузки ini-файлов

Для данного модуля написан тест на предмет корректного взаимодействия с системой ввода/вывода. Код теста представлен в приложении 8.4, результаты тестирования приведены на рисунке 8. Убедиться в работоспособности модуля можно также открыв справку в программе (рисунок 10)

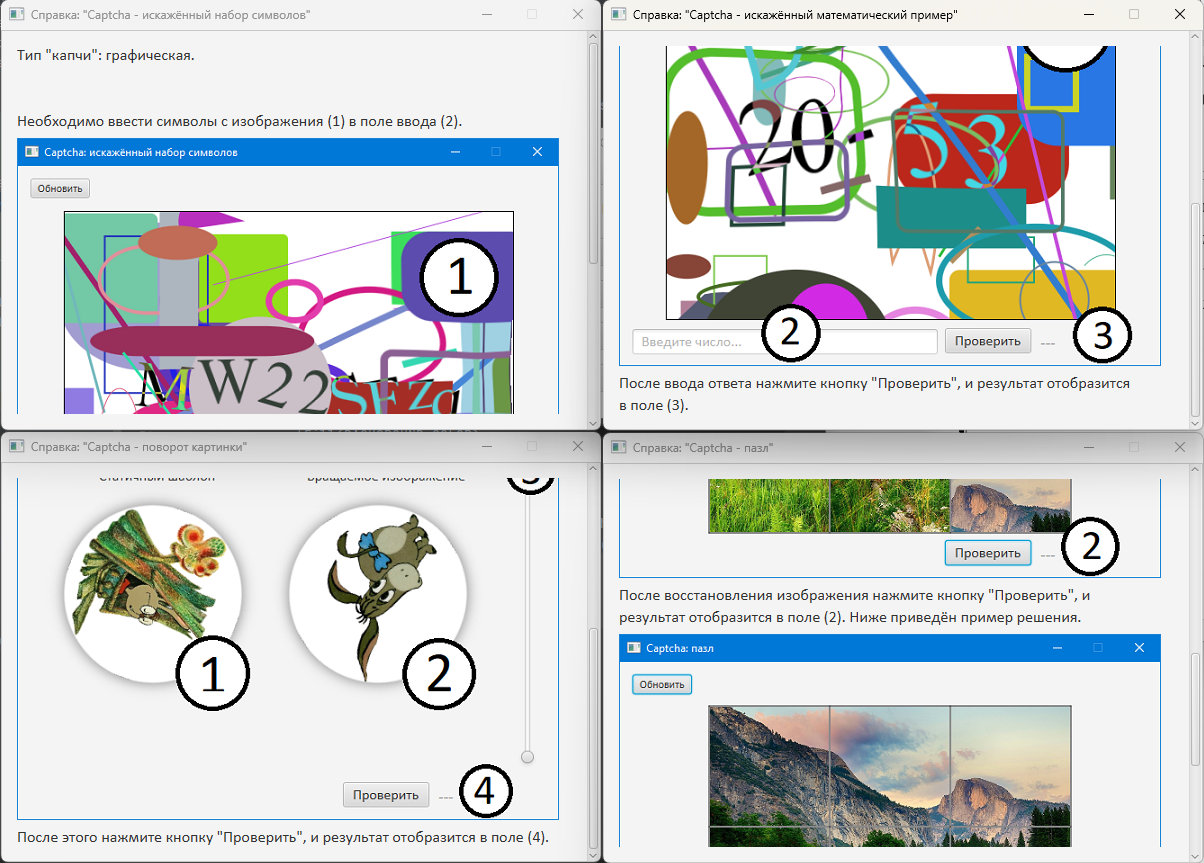


Рисунок 10 – ручная проверка взаимодействия модуля загрузки ini-файлов с системой ввода / вывода

* Модули математики, модули для работы с изображениями

Тестирование проводилось вручную. Рисунок 11 демонстрирует корректное взаимодействие модулей между собой.

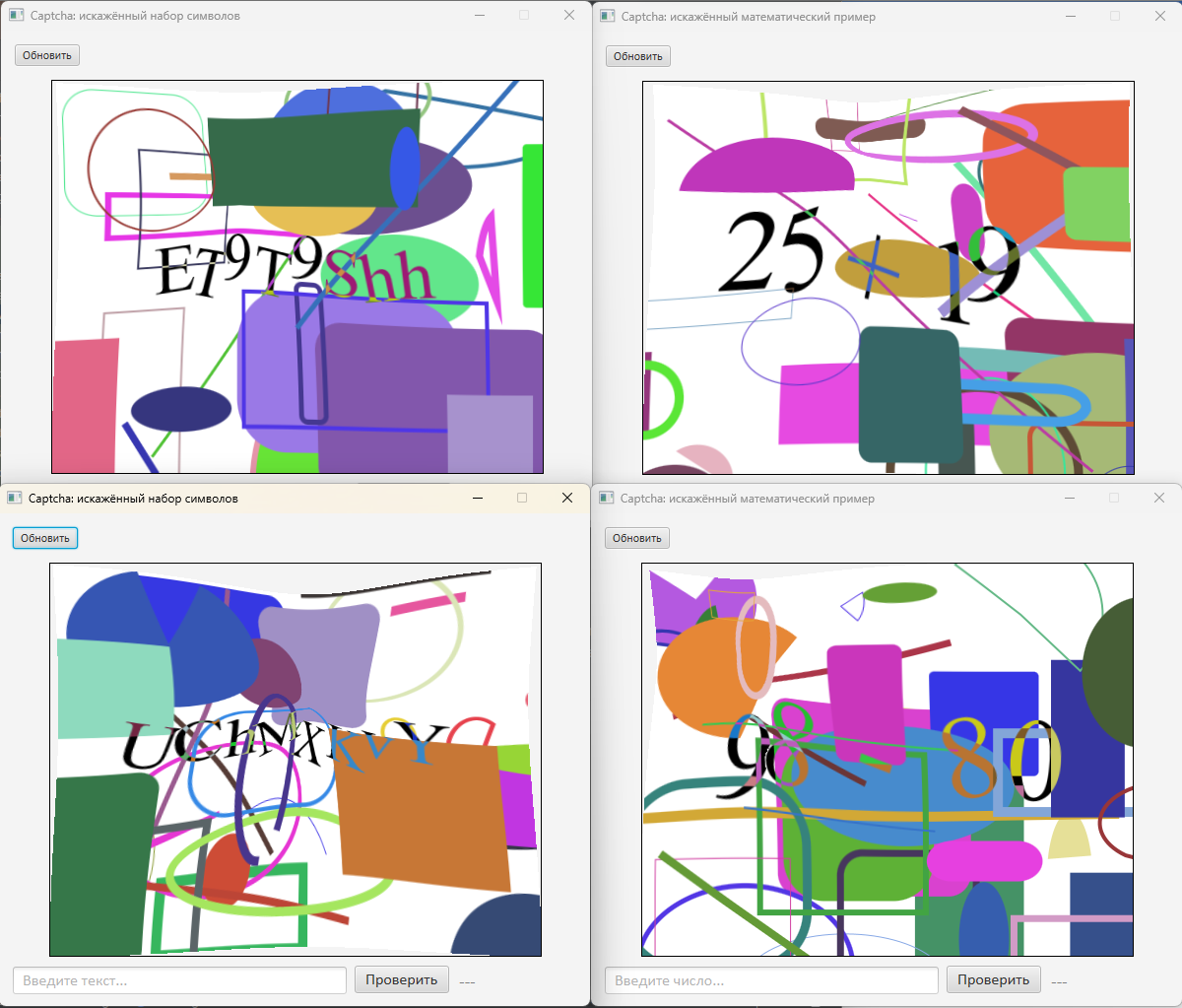


Рисунок – ручная проверка взаимодействия модулей генерации изображений и модуля математики

# Системное тестирование

Основной задачей системного тестирования является проверка функциональных требований к системе. При этом выявляются дефекты, такие как неверное использование ресурсов системы, непредусмотренный комбинации данных пользовательского уровня, несовместимость с окружением, непредусмотренные сценарии использования, отсутствующая или неверная функциональность и т.д.

## Функциональные требования

Для системного тестирования могут быть выделены следующие функциональные требования к программе:

1. программа должна позволять пользователю проходить капчу всех представленных видов (всего 4 штуки), отображая результат решения в главном окне;
2. при инициации прохождении капчи пользователем, программа должна открывать новое окно с запрошенным типом теста;
3. программа должна фиксировать ответ пользователя только после того, как он нажмёт кнопку проверки решения;
4. программа не должна позволять пользователю закрывать окно решения капчи, если статус ответа не определён;
5. при решении математической капчи, программа не должна позволять пользователю вводить недопустимые символы, отличные от цифр и знака «минус»; при проверке решения она должна проверять, является ли введённый пользователем ответ числом, и, в случае необходимости, информировать пользователя ю этом несоответствии, не засчитывая ответ;
6. при решении капчи поворота картинки, программа должна позволять пользователю допускать некоторую погрешность ввода, на уровне 10 градусов.

Поскольку функционал приложения не очень большой, тестирование будет проведено вручную.

## Тестирование

1. При открытии приложения, пользователя встречает окно-хаб с возможностью выбора типа тестирования. На рисунке 12 слева можно видеть начальное состояние экрана, а справа – если пользователь правильно решит первую и четвёртую капчу, и ошибётся в третьей. Ошибок не выявлено.

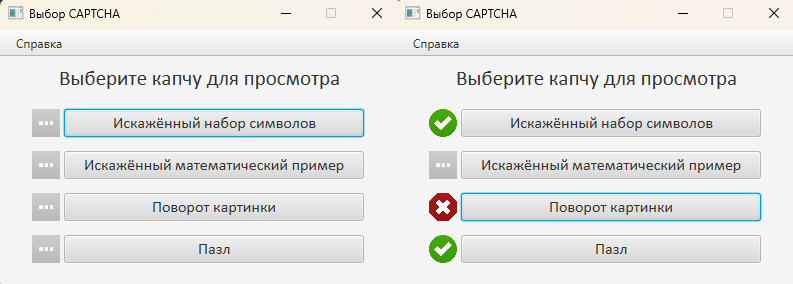


Рисунок – ручная проверка отображения прохождения капчи

1. Удерживая окно-хаб в области видимости, можно открывать окна всех видов капч сколь угодное количество раз. На рисунке 13 показано открытие всех 4х видов. Ошибок не выявлено.

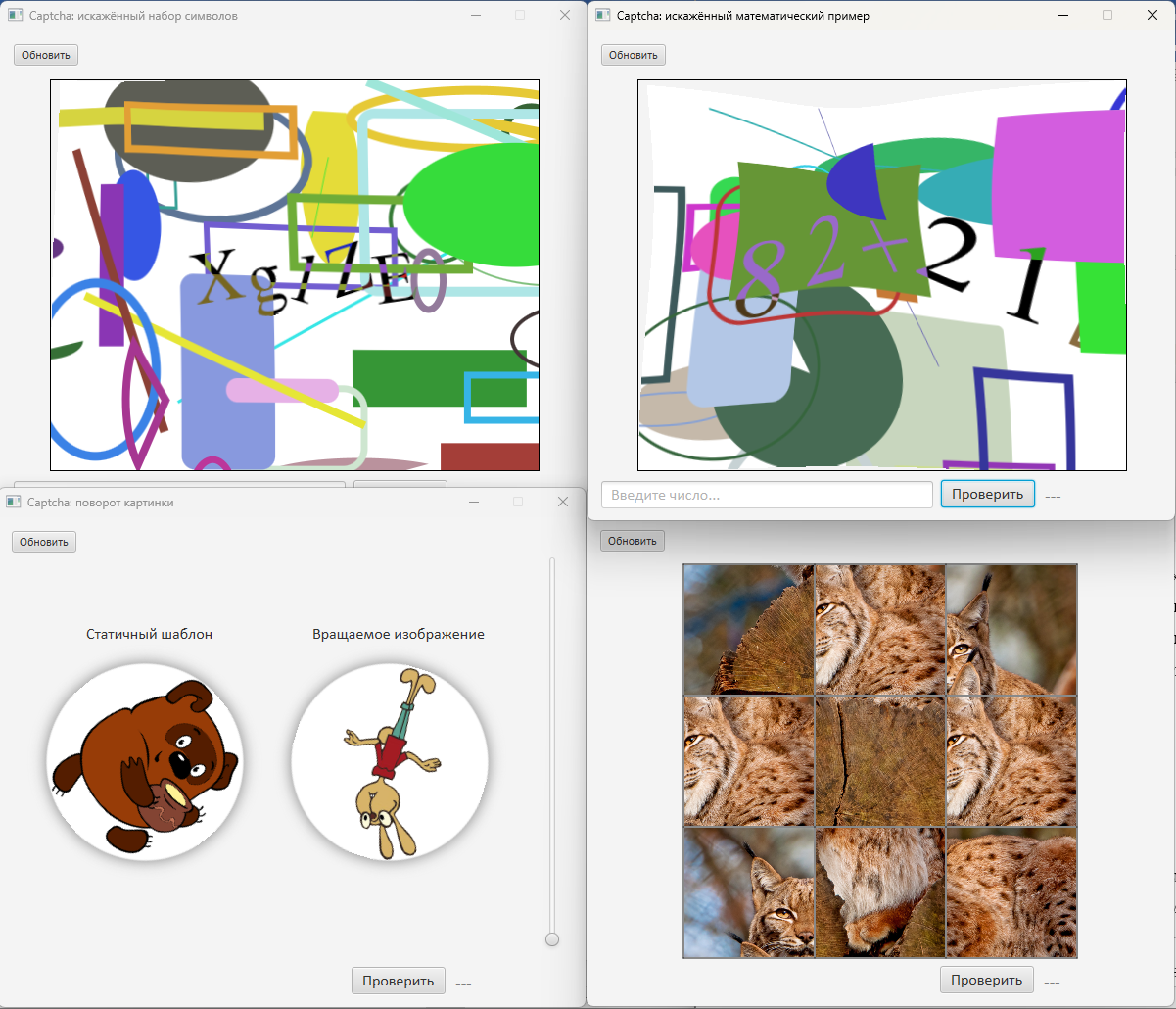


Рисунок 13 – ручная проверка открытия новых окон

1. На риснуке 14 слева ответ введён, но кнопка проверки ещё не нажата, справа – после проверки результат отобразился в соответствующей графе. Ошибок не выявлено.

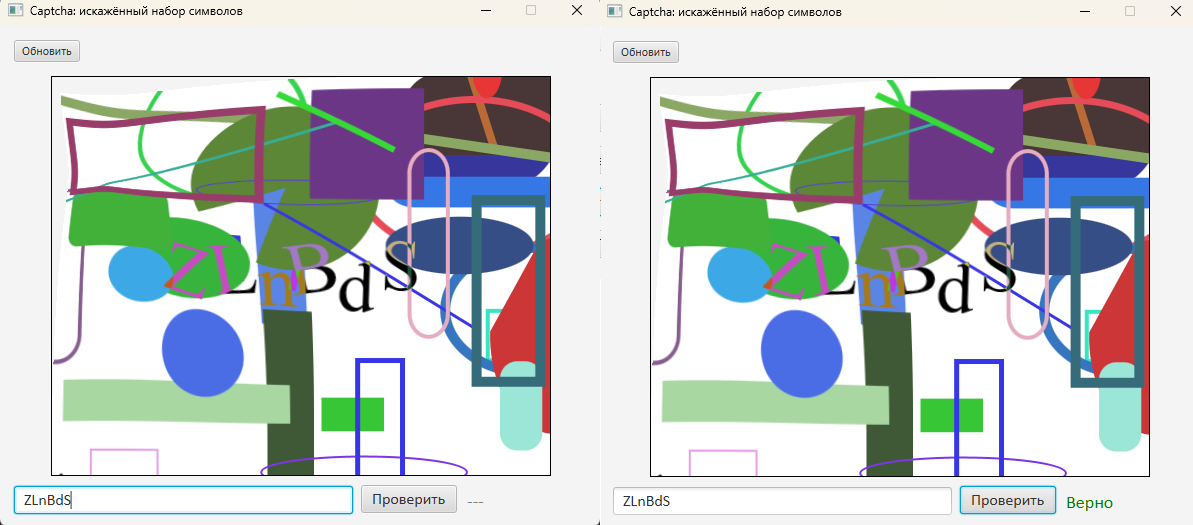


Рисунок 14 – ручная проверка фиксации ответа после нажатия кнопки подтверждения

1. При попытке закрыть окно без подтверждения статуса решения, программа выдаёт информационное сообщение о невозможности закрытия окна в данный момент (рисунок 15). Ошибок не выявлено.

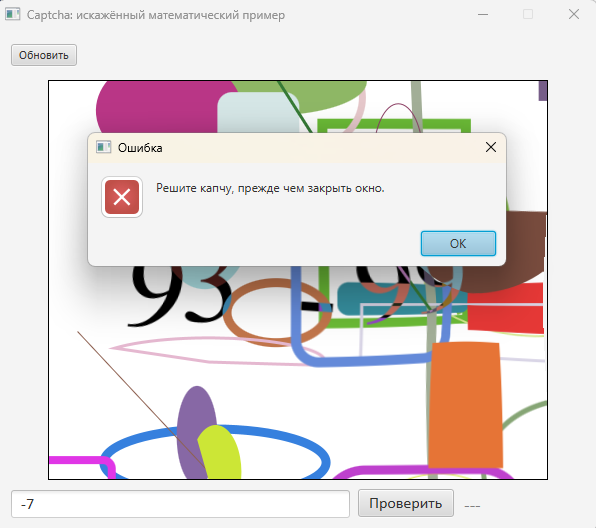


Рисунок 15 – ручное тестирование закрытия окна, когда решение не подтверждено

1. При попытке ввести неразрешенный символ программа сообщит пользователю об этом, к тому же символ не отобразится в поле ввода (рисунок 16, слева); если пользователь введёт значение, не являющееся числом, программа также сообщит об ошибке (тот же рисунок, справа). Ошибок не выявлено.

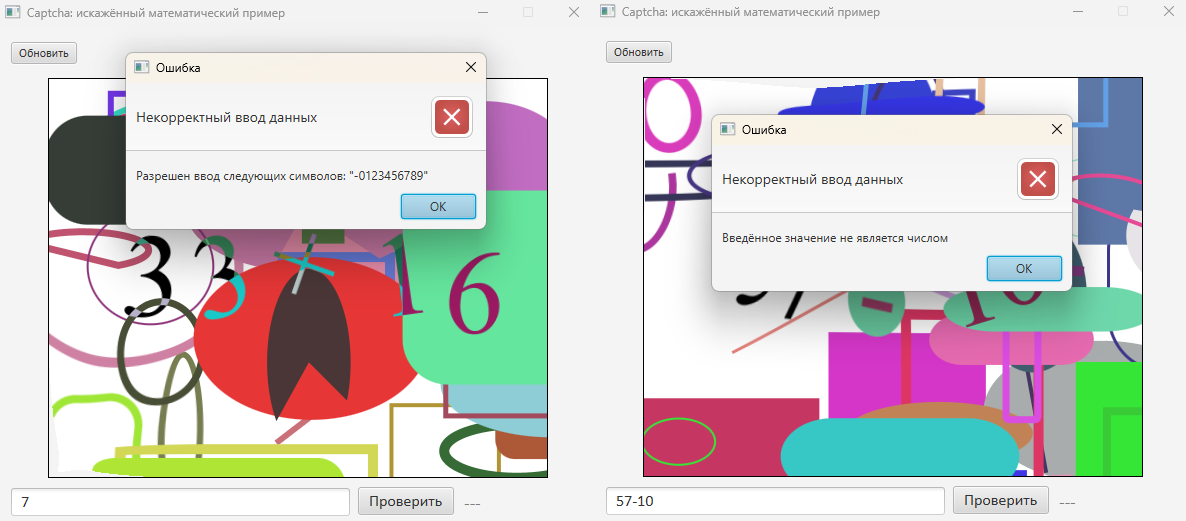


Рисунок 16 – ручная проверка ввода недопустимого символа и «не числа» в окне математической капчи

1. При решении были найдены крайние положения, в которых ответ засчитывался как верный. Как и ожидалось, отклонение более чем на 10 градусов засчитывается как неверный ответ (рисунок 17).

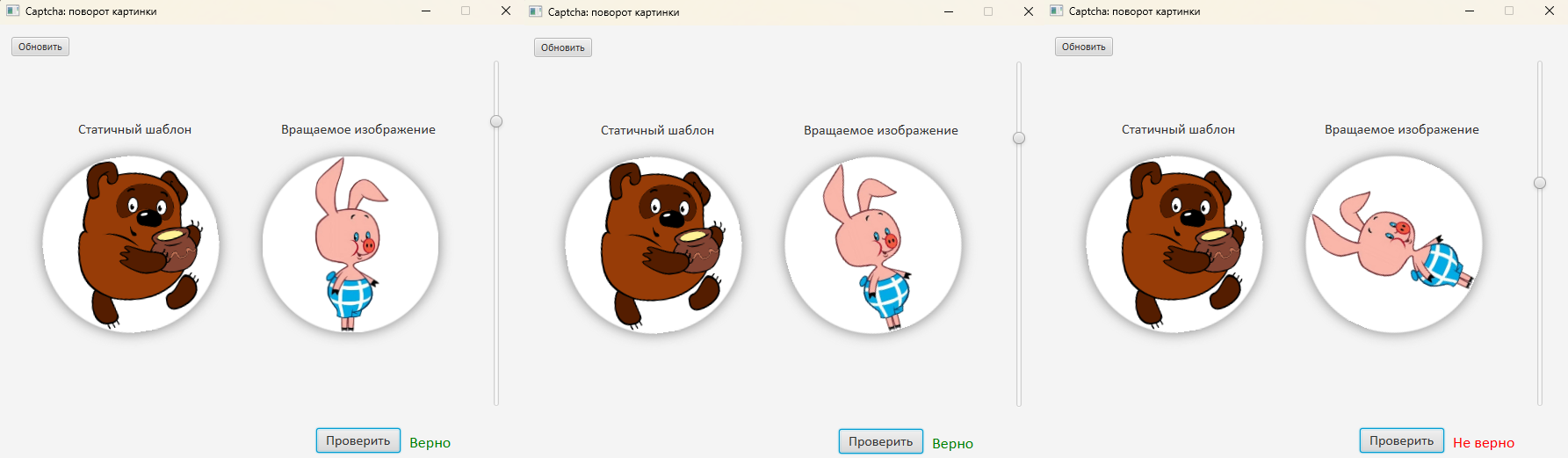


Рисунок 17 – ручная проверка погрешности ввода при решении капчи с поворотом картинки

# Вывод

В ходе курсовой работы были получены навыки разработки тестов программного обеспечения, разработанного на Java с использованием графического фреймворка JavaFX. В процессе работы были изучены библиотеки JUnit4 и TestFX. Проведено структурное, модульное, интеграционное и системное тестирование.

Во время структурного тестирования был спроектирован управляющий граф программы и конкретизированы условия критериев тестирования команд, ветвей и путей. Поскольку пути тестов покрывали критерии C0, C1, C2, это дало возможность провести максимально полное тестирование графа.

В ходе модульного и интеграционного тестирования, модули не всегда проходили тесты. В таком случае, код подвергался модификации, чтобы избавиться от непредвиденного поведения программы. Для автоматического тестирования при загрузке кода на GitHub, написан конфигурационный файл, позволяющий удалённо выполнить сборку и тестирование приложения (см приложение 7.6).

При выполнении системного тестирования были проверены функциональные требования к программе.

Можно сделать вывод, что программа полностью протестирована и готова к дальнейшему использованию.

# Приложение

В целях экономии места, приведён только код самих тестов. Импорт зависимостей рассмотрен не будет.

## Код тестов модуля математики

**public** **class** AdditionalMathTest {

@Test *// unit*

**public** **void** testMapValue() {

**double** value = 1;

**double** srcFrom = 0;

**double** srcTo = 4;

**double** dstFrom = 0;

**double** dstTo = 10;

**double** actual = AdditionalMath.mapValue(value, srcFrom, srcTo, dstFrom, dstTo);

assertEquals(2.5, actual, 0.0001);

}

@Test *// unit*

**public** **void** testEuclideanDistance2D() {

**double**[] point1 = {0, 0};

**double**[] point2 = {1, 2};

**double** actual = AdditionalMath.euclideanDistance(point1, point2);

assertEquals(Math.sqrt(5), actual, 0.0001);

}

@Test *// unit*

**public** **void** testEuclideanDistance3D() {

**double**[] point1 = {1, 2, 3};

**double**[] point2 = {0, 0, 0};

**double** actual = AdditionalMath.euclideanDistance(point1, point2);

assertEquals(Math.sqrt(14), actual, 0.0001);

}

}

## Код тестов модуля генерации фонового изображения капчи

**public** **class** ImageUtilsTest **extends** ApplicationTest {

@BeforeClass

**public** **static** **void** setUpClass() **throws** Exception {

ApplicationTest.launch(Main.**class**);

}

@Test *// unit*

**public** **void** testBilinearInterpolateColor() {

Color[] pixels = **new** Color[4];

pixels[0] = **new** Color(0.04, 0.08, 0.92, 1.0); // ul

pixels[1] = **new** Color(0.56, 0.29, 0.33, 1.0); // ur

pixels[2] = **new** Color(0.09, 0.56, 0.06, 1.0); // br

pixels[3] = **new** Color(0.77, 0.23, 0.77, 1.0); // bl

**double** dx = 0.2;

**double** dy = 0.7;

Color actual = ImageUtils.bilinearInterpolateColor(pixels, dx, dy);

Color expected = **new** Color(0.4870, 0.2438, 0.6802, 1.0);

assertEquals(expected, actual);

}

@Test *// unit*

**public** **void** testGenerateRandomColor() {

Color color = ImageUtils.generateRandomColor();

assertNotNull("Object is **null**", color);

}

@Test *// unit*

**public** **void** testGenUniqueRandColorsNegativeCount() {

**var** randThread = ThreadLocalRandom.current();

**var** colorToAvoid = ImageUtils.generateRandomColor();

**int** count = -Math.abs(randThread.nextInt());

ArrayList<Color> colors = ImageUtils.generateUniqueRandomColors(count, colorToAvoid);

assertNotNull("Object is **null**", colors);

assertTrue("List of colors is not empty, meanwhile count is negative", colors.isEmpty());

}

@Test *// unit*

**public** **void** testGenUniqueRandColorsNormal() {

**var** randThread = ThreadLocalRandom.current();

**var** colorToAvoid = ImageUtils.generateRandomColor();

**int** count = Math.abs(randThread.nextInt()) % 1024;

ArrayList<Color> colors = ImageUtils.generateUniqueRandomColors(count, colorToAvoid);

assertNotNull("Object is **null**", colors);

assertEquals("Number of generated colors didn't match with passed count", count, colors.size());

assertEquals("Avoided color is in resulting list", -1, colors.indexOf(colorToAvoid));

**boolean** duplicate = **false**;

**var** hashes = **new** HashSet<Integer>();

**for** (**var** color : colors) {

**int** hash = color.hashCode();

**if** (hashes.contains(hash)) {

duplicate = **true**;

**break**;

}

hashes.add(hash);

}

assertFalse("Found duplicate color", duplicate);

}

@Test *// unit*

**public** **void** testGenUniqueRandColorsLargeCount() {

**var** randThread = ThreadLocalRandom.current();

**var** colorToAvoid = ImageUtils.generateRandomColor();

**int** count;

**do** {

count = Math.abs(randThread.nextInt());

} **while** (count < 1024);

**int** finalCount = count;

**boolean** exceptionThrown = **false**;

**try** { ImageUtils.generateUniqueRandomColors(finalCount, colorToAvoid); }

**catch** (IllegalArgumentException e) { exceptionThrown = **true**; }

assertTrue("Exception wasn't thrown", exceptionThrown);

}

@Test *// unit*

**public** **void** testMix() {

Color c1 = **new** Color(1.0, 0.0, 0.2, 1.0);

Color c2 = **new** Color(0.0, 1.0, 0.8, 1.0);

**double** factor = 0.5;

Color actual = ImageUtils.mix(c1, c2, factor);

Color expected = **new** Color(0.5, 0.5, 0.5, 1.0);

assertEquals(expected, actual);

}

@Test *// unit*

**public** **void** testDrawPrimitives() {

**var** snapshotRef = **new** AtomicReference<WritableImage>();

**var** canvasRef = **new** AtomicReference<Canvas>();

GraphicsContext gc;

**final** Color BACKGROUND\_COLOR = ImageUtils.generateRandomColor();

*// Зальём изображение указанным цветом, и убедимся, что заливка выполнена*

**final** **int** WIDTH, HEIGHT; WIDTH = HEIGHT = 128;

canvasRef.set(**new** Canvas(WIDTH, HEIGHT));

gc = canvasRef.get().getGraphicsContext2D();

gc.setFill(BACKGROUND\_COLOR);

gc.fillRect(0, 0, WIDTH, HEIGHT);

interact(() -> snapshotRef.set(canvasRef.get().snapshot(**new** SnapshotParameters(), **null**)));

assertFalse("There is color, different from background",

findDifferentColor(BACKGROUND\_COLOR, snapshotRef.get()));

*// Сгенерируем случайный набор различных цветов в количестве доступных примитивов, не схожих с цветом фона.*

*// Нарисуем несколько примитивов каждого цвета, и убедимся, что что-либо выводится.*

**int** count = ImageUtils.PrimitiveType.values().length;

ArrayList<Color> colors = ImageUtils.generateUniqueRandomColors(count, BACKGROUND\_COLOR);

ArrayList<ImageUtils.PrimitiveType> primitives =

**new** ArrayList<>(Arrays.asList(ImageUtils.PrimitiveType.values()));

Collections.shuffle(primitives);

**for** (**int** i = 0; i < count; i++) {

**var** primitiveType = primitives.get(i);

**if** (primitiveType == ImageUtils.PrimitiveType.none) *// пропускаем, если тип 'none'*

**continue**;

*// рисуем примитив*

ImageUtils.drawPrimitive(gc, primitiveType, List.of(colors.get(i), colors.get(i)));

}

*// проверка того, что примитивы нарисованы*

interact(() -> snapshotRef.set(canvasRef.get().snapshot(**new** SnapshotParameters(), **null**)));

**int** foundedColors = 0;

**for** (**int** i = 0; i < count; i++) {

**var** primitiveType = primitives.get(i);

**if** (primitiveType == ImageUtils.PrimitiveType.none) *// пропускаем, если тип 'none'*

**continue**;

**if** (findThisColor(colors.get(i), snapshotRef.get()))

foundedColors++;

}

assertTrue("No primitives are visible. May be it is **false**-positive detection?",

foundedColors > 0);

}

}

## Код тестов модуля генерации текстового растра

**public** **class** ImageTextUtilsTest **extends** ApplicationTest {

@BeforeClass

**public** **static** **void** setUpClass() **throws** Exception {

ApplicationTest.launch(Main.**class**);

}

@Test *// unit*

**public** **void** TestMakeSpecialTextObject() {

**var** rtoRef = **new** AtomicReference<ImageTextUtils.RotatedTextObject>();

interact(() -> rtoRef.set(ImageTextUtils.makeSpecialTextObject("123",

"Times New Roman", 16, 15, 30)));

assertNotNull("Returned object is **null**", rtoRef.get());

assertTrue("Width <= 0", rtoRef.get().getWidth() > 0);

assertTrue("Height <= 0", rtoRef.get().getHeight() > 0);

}

@Test *// unit*

**public** **void** TestDrawRTO() {

**var** snapshotRef = **new** AtomicReference<WritableImage>();

**var** canvasRef = **new** AtomicReference<Canvas>();

GraphicsContext gc;

**var** rtoRef = **new** AtomicReference<ImageTextUtils.RotatedTextObject>();

interact(() -> rtoRef.set(ImageTextUtils.makeSpecialTextObject("123",

"Times New Roman", 16, 15, 30)));

**double** deviation = 16;

**int** width = (**int**) (rtoRef.get().getWidth() + deviation \* 2);

**int** height = (**int**) (rtoRef.get().getHeight() + deviation \* 2);

canvasRef.set(**new** Canvas(width, height));

gc = canvasRef.get().getGraphicsContext2D();

gc.setFill(Color.WHITE);

gc.fillRect(0, 0, width, height);

interact(() -> snapshotRef.set(canvasRef.get().snapshot(**new** SnapshotParameters(), **null**)));

assertFalse("There is color, different from WHITE",

findDifferentColor(Color.WHITE, snapshotRef.get()));

ImageTextUtils.drawRTO(gc.getPixelWriter(), rtoRef.get(),

deviation, deviation, deviation / width, deviation / height);

interact(() -> snapshotRef.set(canvasRef.get().snapshot(**new** SnapshotParameters(), **null**)));

assertTrue("Text was not rasterized",

findThisColor(Color.BLACK, snapshotRef.get()));

}

}

## Код тестов модуля загрузки ini-файлов

**public** **class** HelpConfigurationTest {

**private** **static** HelpConfiguration helpConf = **null**;

@BeforeClass

**public** **static** **void** setUpClass() {

helpConf = **new** HelpConfiguration(

"TESTtitle123",

**new** ArrayList<>(Arrays.asList("TESTstring1", "TESTstring2", "TESTstring3")),

**new** ArrayList<>(Arrays.asList("TESTpath1", "TESTpath2")),

**new** ArrayList<>(Arrays.asList(0, 1, 4)),

**new** ArrayList<>(Arrays.asList(2, 3))

);

}

@Test *// unit*

**public** **void** testGetTitle() {

assertEquals("Title mismatch",

"TESTtitle123", helpConf.getTitle());

}

@Test *// unit*

**public** **void** testGetStrings() {

assertEquals("Strings and its indices arrays length mismatch",

helpConf.getStringsIndices().size(), helpConf.getStrings().size());

}

@Test *// unit*

**public** **void** testGetImages() {

assertEquals("Images paths and its indices arrays length mismatch",

helpConf.getImagesIndices().size(), helpConf.getImages().size());

}

@Test *// integration (with system I/O)*

**public** **void** testLoadFromFile() {

**var** helpConf = **new** HelpConfiguration();

**try** {

helpConf.loadFromFile("/help/config/testHelp.ini");

}

**catch** (Exception e) {

e.printStackTrace();

fail("Failed to open resource test file");

}

assertEquals("Expected title mismatch",

"Test: \"Тест - test\"", helpConf.getTitle());

assertEquals("Strings count mismatch",

2, helpConf.getStrings().size());

assertEquals("Strings and its indices arrays length mismatch",

helpConf.getStringsIndices().size(), helpConf.getStrings().size());

assertEquals("Images paths count mismatch",

2, helpConf.getImages().size());

assertEquals("Images paths and its indices arrays length mismatch",

helpConf.getImagesIndices().size(), helpConf.getImages().size());

assertArrayEquals("String content mismatch",

helpConf.strings.toArray(),

**new** String[]{

"Тест \"test\": тест.\n\n\nTest тест test тест (1) test тест test.",

"Test тест test тест test \"тест\", test\nтест test тест test (2). Тест test тест test."});

assertArrayEquals("Images paths content mismatch",

helpConf.images.toArray(),

**new** String[]{

"/path/to/image/in/resource/folder/1.png",

"/path/to/image/in/resource/folder/2.png"});

}

}

## Код теста вспомогательного функционала

**public** **class** TestingUtilsTest **extends** ApplicationTest {

@BeforeClass

**public** **static** **void** setUpClass() **throws** Exception {

ApplicationTest.launch(Main.**class**);

}

@Test *// unit (для cw.utils.imgenerator.testDrawPrimitives)*

**public** **void** testColorRgbaEqualsItsHash() {

**final** Color BACKGROUND\_COLOR = ImageUtils.generateRandomColor();

AtomicReference<WritableImage> snapshotRef = **new** AtomicReference<>();

**var** canvas = **new** Canvas(1, 1);

GraphicsContext gc = canvas.getGraphicsContext2D();

Color color = ImageUtils.generateRandomColor();

gc.setFill(BACKGROUND\_COLOR);

gc.fillRect(0, 0, 1, 1);

gc.setFill(color);

gc.fillRect(0, 0, 1, 1);

interact(() -> snapshotRef.set(canvas.snapshot(**new** SnapshotParameters(), **null**)));

*// цвет забирается из снапшота в формате argb*

**int** argb = snapshotRef.get().getPixelReader().getArgb(0, 0

assertEquals("Check Color::hashCode() and fix argbToHash function.",

color.hashCode(), argbToHash(argb));

}

}

## Конфигурационный файл для GitHub actions

**name**: Java CI with Maven

**on**:

**push**:

**branches**: [ "master" ]

**pull\_request**:

**branches**: [ "master" ]

**jobs**:

**build**:

**runs-on**: ubuntu-latest

**steps**:

- **uses**: actions/checkout@v3

- **name**: Set up JDK 17

**uses**: actions/setup-java@v3

**with**:

**java-version**: '17'

**distribution**: 'temurin'

**cache**: maven

- **name**: Maven clean and verify

**run**: mvn -B clean verify --file Program/pom.xml -DskipTests

- **name**: Build with Maven

**run**: mvn -B package --file Program/pom.xml -DskipTests

**test**:

**runs-on**: ubuntu-latest

**steps**:

- **uses**: actions/checkout@v3

- **name**: Set up JDK 17

**uses**: actions/setup-java@v3

**with**:

**java-version**: '17'

**distribution**: 'temurin'

**cache**: maven

- **name**: Maven clean and verify

**run**: mvn -B clean verify --file Program/pom.xml -DskipTests

- **name**: Run tests

**run**: xvfb-run mvn -B test --file Program/pom.xml