**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

Лабораторная работа №2 по дисциплине «Инжиринг программного обеспечения»

**Графический интерфейс с динамической разметкой.**

Группа: **АВТ-342**

Студенты: **Вострецов Н.С., Фадеев В.А.** Преподаватель: **Дыминский И.И.**

НОВОСИБИРСК 2025

**1.Постановка задачи**

**Вариант 5.**

Игровой автомат с прокруткой картинок. Анимация прокрутки с плавным перемещением картинки в панели.

**2. Описание архитектуры**

Программа написана на Java с использованием JavaFX.

Основные компоненты:

1. Графический интерфейс

casino.fxml - XML-описание интерфейса; Main.java – файл запуска приложения; CasinoController.java - основной контроллер, управляющий логикой приложения

2. Логика

GameState.java – показатель состояний игры (IDLE, SPINNING, STOPPING)

Command.java - интерфейс для реализации паттерна "Команда"

StartSpinCommand - конкретная реализация команды запуска вращения

3. Вероятности

RandomNumberGenerator.java - интерфейс генератора случайных чисел

DefaultRandomGenerator.java - реализация по умолчанию

ProbabilityCalculator.java - интерфейс калькулятора вероятностей

DefaultProbabilityCalculator.java - реализация по умолчанию

Реализован гибкий интерфейс с динамическим количеством колонок (1-10), настройка количества используемых картинок (1-8), автоматическое определение количества рядов (1 при ≤ 5 колонках, 2 при >5 колонках.

Сделана плавная анимация прокрутки картинок в колонке. Каждая колонка – vbox, содержащий картинки, которые выбираются случайно в зависимости от количества подгруженных изображений. Область отображения ограничена центральной картинкой, на которой в итоге и останавливается барабан.

Реализован автомат с тремя состояниями:

IDLE - состояние покоя, SPINNING - состояние прокрутки, STOPPING - состояние остановки,

Используются классы JavaFX Animation API:

Timeline для создания последовательности кадров анимации

PauseTransition для создания задержек между запуском колонок

KeyFrame для определения состояний в разные моменты времени

**Описание работы программы и архитектурных приемов:**

Запуск программы:

Программа начинается с метода main в классе Main, запускается метод start. В методе start создается FXMLLoader, который загружает FXML-файл casino.fxml. При загрузке FXML автоматически создается экземпляр CasinoController и вызывается его метод initialize.

Инициализация контроллера:

В методе initialize выполняются следующие действия:

setupSettingsPanel - устанавливает начальные значения в текстовые поля columnsField и symbolsToUseField

setupBindings - настраивает привязки между свойствами columnsProperty, rowsProperty, symbolCountProperty, symbolsToUseProperty и isSpinningProperty

createGameArea - создает игровую область с барабанами

setupEventHandlers - настраивает обработчики событий для кнопок applySettingsButton, startButton и loadImagesButton

loadDefaultImages - загружает изображения из директории images

applyCasinoStyle - применяет стили казино ко всем элементам интерфейса

Загрузка изображений:

Метод loadDefaultImages проверяет существование директории images. Если директория не существует, она создаётся и выводится сообщение в statusLabel. Если директория существует, вызывается метод loadImagesFromDirectory, который загружает все изображения форматов PNG, JPG. Количество загруженных изображений устанавливается в symbolCountProperty.

Создание игровой области:

Метод createGameArea очищает игровую область и создает новую сетку для барабанов. В зависимости от значения columnsProperty определяется количество рядов: если колонок 5 или меньше - один ряд, если больше - два ряда. Для каждой колонки вызывается метод createColumn, который создает VBox с тремя изображениями одинакового размера 110x110 пикселей. Созданные барабаны сохраняются в списки reels и reelImages.

Расчет вероятности выигрыша:

Метод updateProbability вызывается при изменении свойств columnsProperty, rowsProperty или symbolCountProperty. Он использует probabilityCalculator для вычисления вероятности выигрыша по формуле, которая учитывает количество символов, колонок и рядов. Результат отображается в probabilityLabel.

Обработка пользовательского ввода:

При нажатии кнопки applySettingsButton вызывается метод applySettings, который проверяет корректность введенных значений в columnsField и symbolsToUseField. Если значения правильные, они устанавливаются в свойства columnsProperty и symbolsToUseProperty, после чего при выполнении этого условия, обновляется интерфейс с помощью метода createGameArea.

При нажатии кнопки startButton создается экземпляр StartSpinCommand и передается в метод executeCommand, который проверяет возможность выполнения команды в текущем состоянии игры через метод canExecute.

Запуск вращения барабанов:

Метод startSpin изменяет состояние игры на SPINNING через currentState и isSpinningProperty. Для каждой колонки создается PauseTransition с увеличивающейся задержкой, и по окончании задержки вызывается метод spinColumn для соответствующей колонки.

Анимация вращения:

Метод spinColumn создает TranslateTransition, который смещает VBox барабана на -110 пикселей по оси Y. После завершения анимации первое изображение в VBox удаляется, меняется на случайное новое изображение и добавляется в конец VBox. Позиция VBox сбрасывается на 0. Если игра все еще в состоянии SPINNING, метод spinColumn вызывается рекурсивно.

Проверка результата:

После завершения всех анимаций вызывается метод checkWin, который сравнивает видимые изображения в первом ряду каждой колонки. Если все изображения одинаковые, выводится сообщение о выигрыше в resultLabel, иначе - сообщение о проигрыше.

Обновление состояния игры:

По завершении проверки результата состояние игры возвращается в IDLE через currentState и isSpinningProperty, что разрешает запуск нового вращения.

Математическая модель подсчёта вероятностей:

**Вероятность вычисляется по двухэтапной формуле:**

1. Вероятность совпадения в одной строке:

double prow = Math.pow(1.0 / symbolCount, columns - 1)

Это вероятность того, что все символы в одной строке совпадут с первым символом.

1. Вероятность хотя-бы одной выигрышной строки:

double pwin = 1 - Math.pow(1 - prow, rows)

Эта формула вычисляет вероятность того, что хотя бы одна из строк будет полностью совпадающей.

**Примеры расчётов:**

**Пример 1:** 3 колонки, 1 ряд, 5 картинок

prow = (1/5)^(3-1) = 0.2^2 = 0.04

pwin = 1 - (1 - 0.04)^1 = 0.04 = 4%

**Пример 2:** 5 колонок, 1 ряд, 8 картинок

prow = (1/8)^4 = 0.00024414

pwin = 1 - (1 - 0.00024414)^1 = 0.00024414 = 0.0244%

**Архитектурные приемы**

**Разделение на интерфейсы и реализации**

Правила генерации и расчёта вынесены в отдельные интерфейсы, а контроллер автомата работает только с абстракциями.

**Паттерн Command**  
Действия StartSpin и LoadImages оформлены как объекты-команды. У каждой команды есть метод execute() (что сделать) и canExecute() (можно ли сейчас выполнить). Это позволяет контролировать действия автомата через наши состояния.

**Состояния игры через enum GameState**  
Автомат всегда находится в одном из состояний: покой (IDLE), кручение (SPINNING) или остановка (STOPPING).

**Анимации**

Каждый барабан в автомате — это вертикальная коробка (VBox) с картинками (ImageView).  
Чтобы создать эффект вращения, смещают контейнер по оси Y.

За 200 миллисекунд контейнер плавно сдвигается вверх на высоту символа.

После завершения верхний элемент вырезается и перемещается вниз, при этом ему подставляется новая случайная картинка.

Чтобы автомат знал, когда пора проверить совпадения, используется Timeline.

Считается время (количество колонок \* 300 мс) + 2000 мс.

После этого времени вызывается метод checkWin(), который сравнивает верхние картинки в колонках.

В программу были добавлены два новых параметра, позволяющих гибко настраивать визуальное поведение игрового автомата:

* **Скорость прокрутки (интенсивность)** – определяет, насколько быстро сменяются кадры анимации в каждом барабане.
* **Длительность симуляции** – общее время в секундах, в течение которого длится вращение барабанов перед проверкой результата.

**Были** введены два свойства для реализации упомянутых параметров:

* DoubleProperty spinSpeedProperty – хранит коэффициент скорости прокрутки (значение 1.0 – базовая скорость, меньше 1.0 – медленнее, больше 1.0 – быстрее).
* DoubleProperty simulationSpeedProperty – хранит общую длительность вращения в секундах.

**Элементы управления в FXML:** В файл casino.fxml добавлены соответствующие текстовые поля (TextField):

* spinSpeedField – для ввода значения скорости прокрутки.
* simulationSpeedField – для ввода длительности симуляции.

**Обработка в контроллере (**CasinoController.java**):**

* В методе setupSettingsPanel() устанавливаются начальные значения в новые поля ввода.
* В методе setupBindings() добавлены слушатели (Listener) для свойств spinSpeedProperty и simulationSpeedProperty, которые обновляют текст в полях ввода при изменении свойств (например, из кода).
* Метод applySettings() расширен: теперь он валидирует введённые пользователем значения для новых параметров (скорость от 0.01 до 10, длительность от 0.5 до 30 секунд) и, в случае успеха, устанавливает их в соответствующие свойства.

**Интеграция с логикой анимации:**

В методе startSpin() рассчитываются фактические временные параметры анимации на основе текущих значений свойств:

* long totalSimulationMs = (long) (simulationSpeedProperty.get() \* 1000) – общее время вращения в миллисекундах.
* long stepDurationMs = (long) (200 / spinSpeedProperty.get()) – длительность одного шага анимации (смещения на один символ). Базовая длительность шага (200 мс) делится на коэффициент скорости. Таким образом, при spinSpeed = 2.0 шаг будет длиться 100 мс (анимация ускорится в 2 раза), а при spinSpeed = 0.5 – 400 мс (анимация замедлится в 2 раза).
* Эти рассчитанные значения (stepDurationMs и totalSimulationMs) передаются в метод spinColumn() для каждого барабана, обеспечивая управляемое вращение.

**Синхронизация завершения:**

Для корректной обработки окончания анимации при изменяемой длительности введён механизм отслеживания активных анимаций с помощью AtomicInteger activeAnimations.

После истечения общего времени симуляции (totalSimulationMs) состояние игры переходит в IDLE, запрещая запуск новых анимаций.

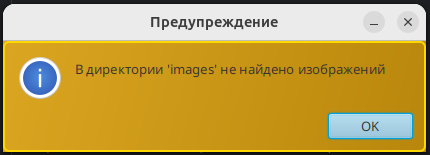
Запускается асинхронная проверка (waitForAnimationsToFinish()), которая ожидает завершения всех текущих анимаций (когда activeAnimations станет равен 0), и только после этого вызывается метод проверки результата showFinalResult(). Это гарантирует, что итоговый расклад символов будет определён после полной остановки всех барабанов, даже если их скорости были настроены по-разному или были очень медленными.

1. **UML-диаграмма**

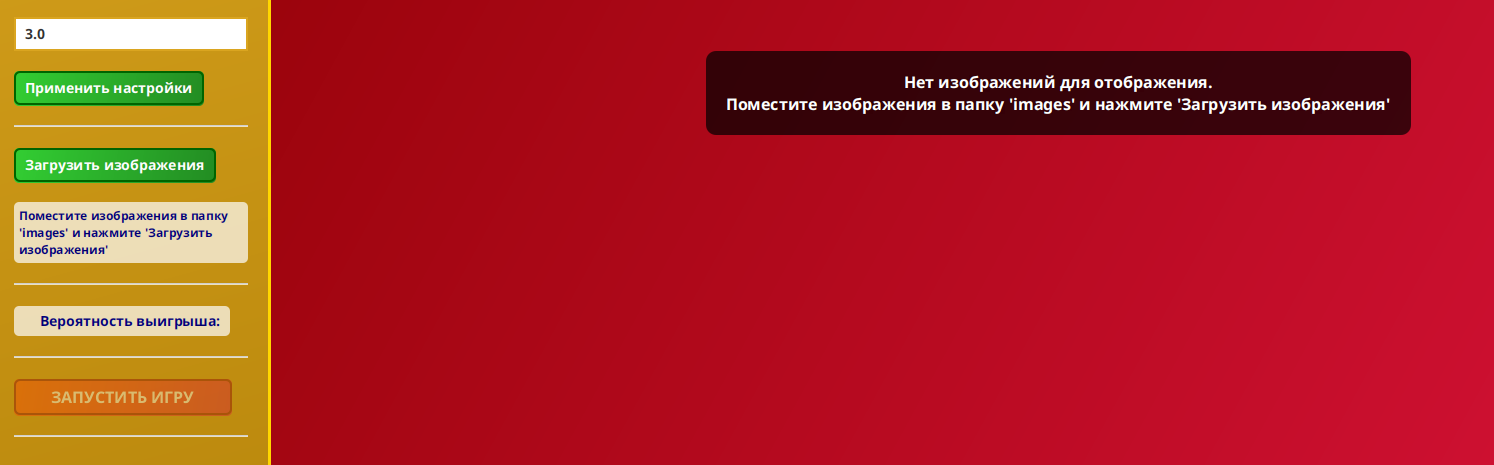
Изображение выглядит как текст, чек, дизайн

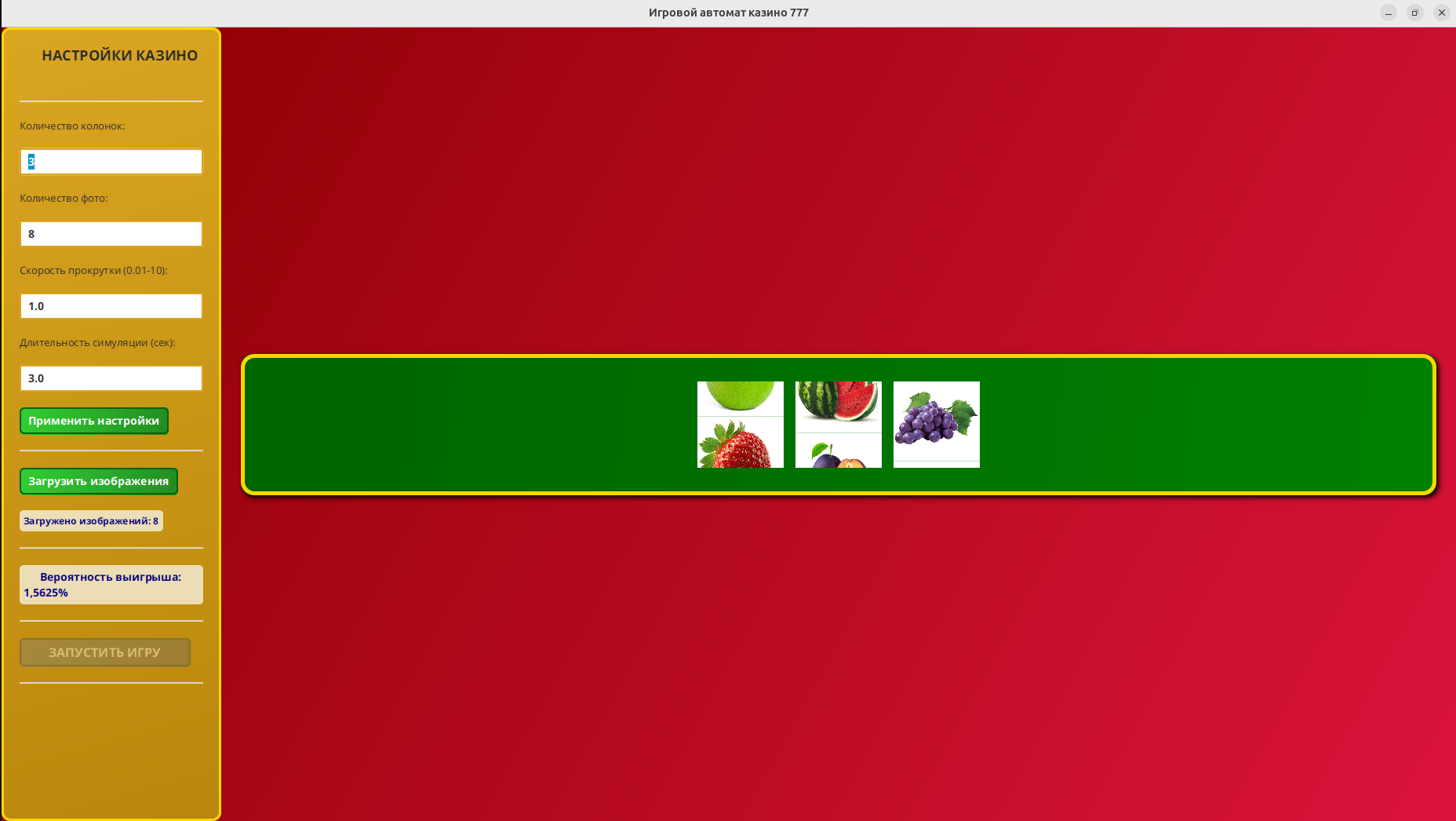
Содержимое, созданное искусственным интеллектом, может быть неверным.

**4. Тестовые примеры выполнения программы**

Проверим обработку ошибок на случай, если в директории нет картинок:

Вращение нельзя запустить:



Запустим вращение (программа была запущена, картинки в images есть):

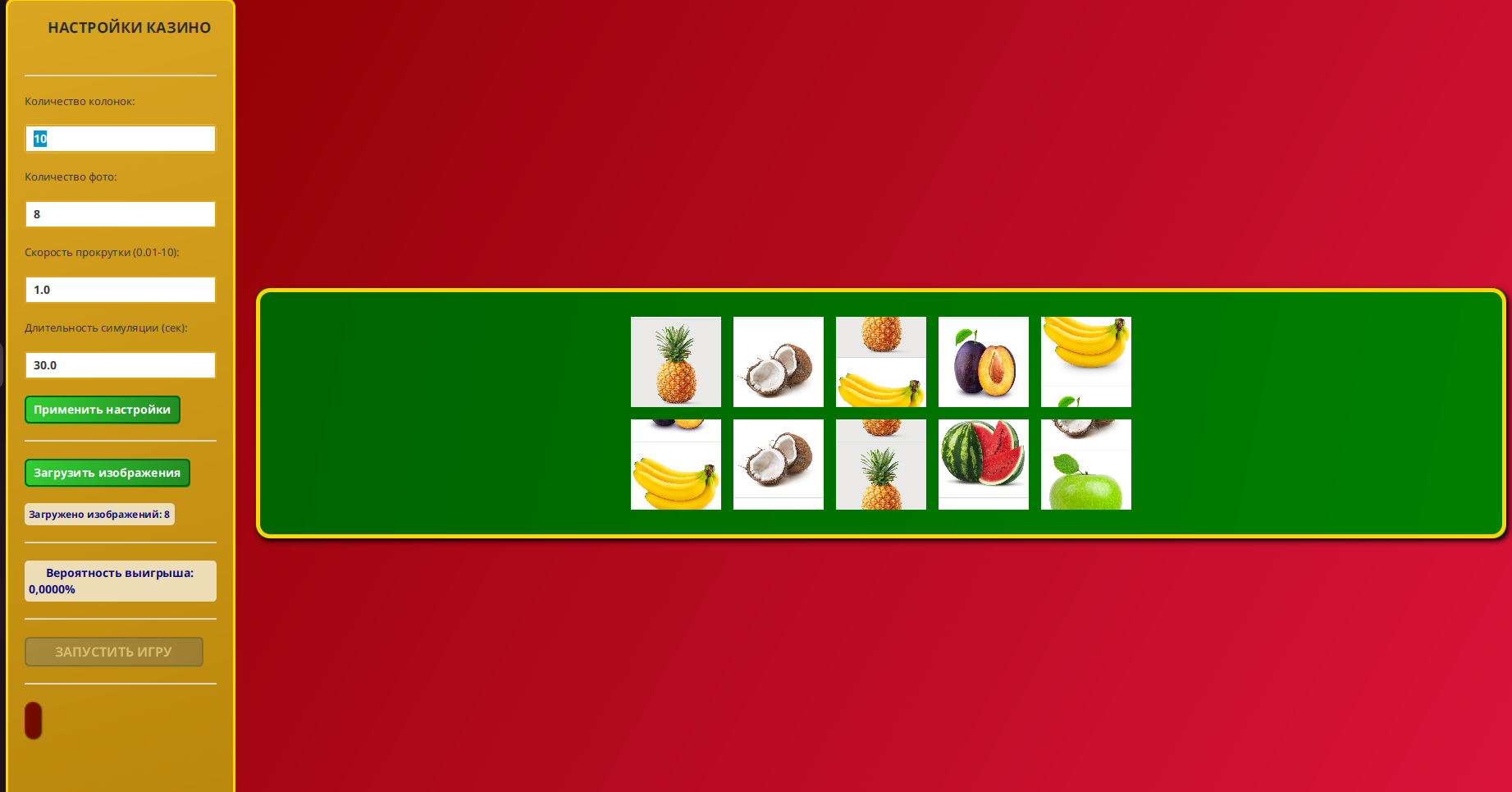
Проигрыш:

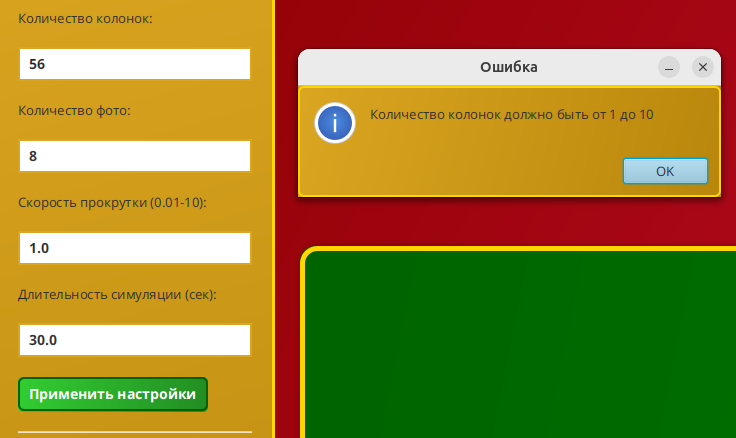


Изменим длительность симуляции и запустим:

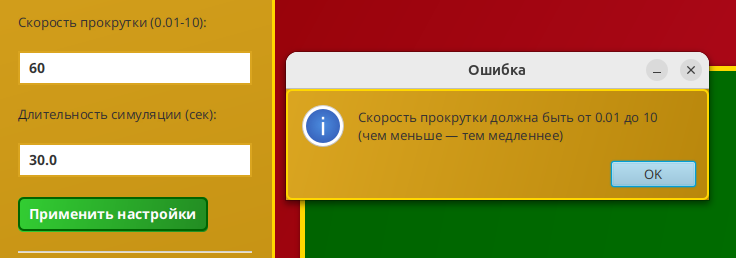


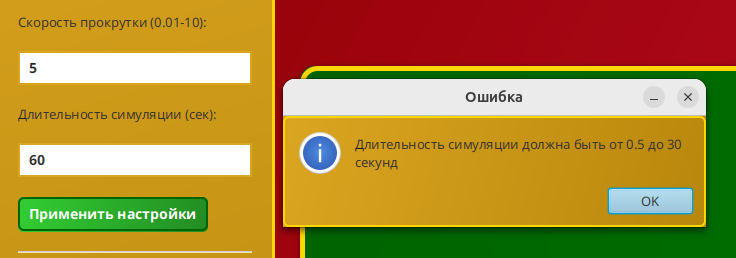
Проверим гибкость интерфейса, изменив количество колонок:



Проверим обработку ошибок в полях ввода:







**5. Вывод**

В рамках учебного проекта была разработана программа на языке Java с использованием библиотеки JavaFX, которая реализует автомат с анимацией прокрутки картинок, гибким интерфейсом, настройкой количества колонок, количества используемых изображений, скорости вращения барабанов и длительности симуляции.

**6. Листинг демонстрационной программы**

**CasinoController.java:**

**package** **dynamic.casino**;

**import** **javafx.animation.\***;

**import** **javafx.application.Platform**;

**import** **javafx.beans.property.\***;

**import** **javafx.fxml.FXML**;

**import** **javafx.geometry.Insets**;

**import** **javafx.geometry.Pos**;

**import** **javafx.scene.Node**;

**import** **javafx.scene.control.\***;

**import** **javafx.scene.image.Image**;

**import** **javafx.scene.image.ImageView**;

**import** **javafx.scene.layout.\***;

**import** **javafx.scene.shape.Rectangle**;

**import** **javafx.util.Duration**;

**import** **javafx.beans.binding.Bindings**;

**import** **javafx.beans.binding.BooleanBinding**;

**import** **java.util.concurrent.atomic.AtomicInteger**;

**import** **java.io.File**;

**import** **java.util.\***;

**public** **class** CasinoController {

@FXML

**private** VBox mainContainer;

@FXML

**private** VBox settingsPanel;

@FXML

**private** TextField columnsField;

@FXML

**private** TextField symbolsToUseField;

@FXML

**private** TextField spinSpeedField; *// Скорость прокрутки (интенсивность)*

@FXML

**private** TextField simulationSpeedField; *// Длительность симуляции в секундах*

@FXML

**private** Button startButton;

@FXML

**private** Button applySettingsButton;

@FXML

**private** Button loadImagesButton;

@FXML

**private** VBox gameArea;

@FXML

**private** Label probabilityLabel;

@FXML

**private** Label resultLabel;

@FXML

**private** Label statusLabel;

**private** **final** IntegerProperty columnsProperty = **new** SimpleIntegerProperty(3);

**private** **final** IntegerProperty rowsProperty = **new** SimpleIntegerProperty(1);

**private** **final** IntegerProperty symbolCountProperty = **new** SimpleIntegerProperty(0);

**private** **final** IntegerProperty symbolsToUseProperty = **new** SimpleIntegerProperty(0);

**private** **final** DoubleProperty spinSpeedProperty = **new** SimpleDoubleProperty(1.0); *// Интенсивность вращения*

**private** **final** DoubleProperty simulationSpeedProperty = **new** SimpleDoubleProperty(3.0); *// Длительность в секундах*

**private** **final** BooleanProperty isSpinningProperty = **new** SimpleBooleanProperty(**false**);

**private** **final** AtomicInteger activeAnimations = **new** AtomicInteger(0);

**private** List<VBox> reels;

**private** List<List<Integer>> reelSymbolIndices; *// Храним индексы, а не Image*

**private** RandomNumberGenerator rng = **new** DefaultRandomGenerator();

**private** ProbabilityCalculator probabilityCalculator = **new** DefaultProbabilityCalculator();

**private** List<Image> gameImages = **new** ArrayList<>();

**private** List<Image> usedImages = **new** ArrayList<>();

**private** String imagesDirectory = "images";

**private** GameState currentState = GameState.IDLE;

@FXML

**public** **void** initialize() {

setupSettingsPanel();

setupBindings();

createGameArea();

setupEventHandlers();

loadDefaultImages();

applyCasinoStyle();

}

**private** **void** setupSettingsPanel() {

columnsField.setText(String.valueOf(columnsProperty.get()));

**if** (symbolsToUseField != **null**) {

symbolsToUseField.setText("0");

}

**if** (spinSpeedField != **null**) {

spinSpeedField.setText(String.valueOf(spinSpeedProperty.get()));

}

**if** (simulationSpeedField != **null**) {

simulationSpeedField.setText(String.valueOf(simulationSpeedProperty.get()));

}

}

**private** **void** setupBindings() {

columnsProperty.addListener((obs, oldVal, newVal) -> {

**if** (columnsField != **null**) {

columnsField.setText(String.valueOf(newVal.intValue()));

}

updateRowsBasedOnColumns(newVal.intValue());

updateProbability();

});

symbolCountProperty.addListener((obs, oldVal, newVal) -> updateProbability());

rowsProperty.addListener((obs, oldVal, newVal) -> updateProbability());

**if** (startButton != **null**) {

BooleanBinding noImagesOrNoColumns = Bindings.createBooleanBinding(

() -> symbolCountProperty.get() <= 0 || columnsProperty.get() <= 0,

symbolCountProperty, columnsProperty

);

startButton.disableProperty().bind(isSpinningProperty.or(noImagesOrNoColumns));

}

symbolsToUseProperty.addListener((obs, oldVal, newVal) -> updateUsedImages());

spinSpeedProperty.addListener((obs, oldVal, newVal) -> {

**if** (spinSpeedField != **null**) {

spinSpeedField.setText(String.valueOf(newVal.doubleValue()));

}

});

simulationSpeedProperty.addListener((obs, oldVal, newVal) -> {

**if** (simulationSpeedField != **null**) {

simulationSpeedField.setText(String.valueOf(newVal.doubleValue()));

}

});

}

**private** **void** setupEventHandlers() {

**if** (applySettingsButton != **null**) {

applySettingsButton.setOnAction(e -> applySettings());

}

**if** (startButton != **null**) {

startButton.setOnAction(e -> executeCommand(**new** StartSpinCommand()));

}

**if** (loadImagesButton != **null**) {

loadImagesButton.setOnAction(e -> loadImagesFromDirectory());

}

}

**private** **void** applyCasinoStyle() {

**if** (mainContainer != **null**) {

mainContainer.setStyle("-fx-background-color: linear-gradient(from 0% 0% to 100% 100%, #8B0000, #dc143c);");

}

**if** (settingsPanel != **null**) {

settingsPanel.setStyle(

"-fx-background-color: linear-gradient(from 0% 0% to 100% 100%, #DAA520, #B8860B);" +

"-fx-padding: 20;" +

"-fx-border-color: #FFD700;" +

"-fx-border-width: 3;" +

"-fx-border-radius: 10;" +

"-fx-background-radius: 10;"

);

}

String buttonStyle =

"-fx-background-color: linear-gradient(from 0% 0% to 100% 100%, #32CD32, #228B22);" +

"-fx-text-fill: white;" +

"-fx-font-weight: bold;" +

"-fx-font-size: 14px;" +

"-fx-border-color: #006400;" +

"-fx-border-width: 2;" +

"-fx-border-radius: 5;" +

"-fx-background-radius: 5;";

**if** (applySettingsButton != **null**) {

applySettingsButton.setStyle(buttonStyle);

}

**if** (loadImagesButton != **null**) {

loadImagesButton.setStyle(buttonStyle);

}

**if** (startButton != **null**) {

startButton.setStyle(

"-fx-background-color: linear-gradient(from 0% 0% to 100% 100%, #FF4500, #DC143C);" +

"-fx-text-fill: white;" +

"-fx-font-weight: bold;" +

"-fx-font-size: 16px;" +

"-fx-border-color: #8B0000;" +

"-fx-border-width: 2;" +

"-fx-border-radius: 5;" +

"-fx-background-radius: 5;"

);

}

String textFieldStyle =

"-fx-background-color: white;" +

"-fx-border-color: #DAA520;" +

"-fx-border-width: 2;" +

"-fx-font-weight: bold;" +

"-fx-font-size: 14px;";

**if** (columnsField != **null**) {

columnsField.setStyle(textFieldStyle);

}

**if** (symbolsToUseField != **null**) {

symbolsToUseField.setStyle(textFieldStyle);

}

**if** (spinSpeedField != **null**) {

spinSpeedField.setStyle(textFieldStyle);

}

**if** (simulationSpeedField != **null**) {

simulationSpeedField.setStyle(textFieldStyle);

}

**if** (statusLabel != **null**) {

statusLabel.setStyle(

"-fx-text-fill: #000080;" +

"-fx-font-weight: bold;" +

"-fx-font-size: 12px;" +

"-fx-background-color: rgba(255, 255, 255, 0.7);" +

"-fx-padding: 5;" +

"-fx-background-radius: 5;"

);

}

**if** (probabilityLabel != **null**) {

probabilityLabel.setStyle(

"-fx-text-fill: #000080;" +

"-fx-font-weight: bold;" +

"-fx-font-size: 14px;" +

"-fx-background-color: rgba(255, 255, 255, 0.7);" +

"-fx-padding: 5;" +

"-fx-background-radius: 5;"

);

}

**if** (resultLabel != **null**) {

resultLabel.setStyle(

"-fx-text-fill: #FFD700;" +

"-fx-font-weight: bold;" +

"-fx-font-size: 16px;" +

"-fx-effect: dropshadow(gaussian, black, 3, 0, 1, 1);"

);

}

}

**private** **void** loadDefaultImages() {

gameImages.clear();

File imagesDir = **new** File(imagesDirectory);

**if** (!imagesDir.exists()) {

imagesDir.mkdirs();

**if** (statusLabel != **null**) {

statusLabel.setText("Создана директория 'images'. Поместите сюда изображения и нажмите 'Загрузить изображения'");

}

**return**;

}

loadImagesFromDirectory();

}

**private** **void** loadImagesFromDirectory() {

gameImages.clear();

File imagesDir = **new** File(imagesDirectory);

**if** (!imagesDir.exists() || !imagesDir.isDirectory()) {

showAlert("Ошибка", "Директория 'images' не найдена");

**return**;

}

File[] imageFiles = imagesDir.listFiles((dir, name) ->

name.toLowerCase().endsWith(".png") ||

name.toLowerCase().endsWith(".jpg") ||

name.toLowerCase().endsWith(".jpeg") ||

name.toLowerCase().endsWith(".gif"));

**if** (imageFiles == **null** || imageFiles.length == 0) {

showAlert("Предупреждение", "В директории 'images' не найдено изображений");

**if** (statusLabel != **null**) {

statusLabel.setText("Поместите изображения в папку 'images' и нажмите 'Загрузить изображения'");

}

**return**;

}

**for** (File imageFile : imageFiles) {

**try** {

Image image = **new** Image(imageFile.toURI().toString());

**if** (!image.isError()) {

gameImages.add(image);

}

} **catch** (Exception e) {

System.err.println("Ошибка загрузки изображения: " + imageFile.getName());

}

}

removeDuplicateImages();

**if** (gameImages.isEmpty()) {

showAlert("Ошибка", "Не удалось загрузить ни одного изображения");

**if** (statusLabel != **null**) {

statusLabel.setText("Ошибка загрузки изображений");

}

} **else** {

symbolCountProperty.set(gameImages.size());

updateUsedImages();

**if** (statusLabel != **null**) {

statusLabel.setText("Загружено изображений: " + gameImages.size());

}

createGameArea();

}

}

**private** **void** updateUsedImages() {

**int** symbolsToUse = symbolsToUseProperty.get();

**if** (symbolsToUse <= 0 || symbolsToUse >= gameImages.size()) {

usedImages = **new** ArrayList<>(gameImages);

} **else** {

usedImages = **new** ArrayList<>(gameImages.subList(0, Math.min(symbolsToUse, gameImages.size())));

}

symbolCountProperty.set(usedImages.size());

}

**private** **void** applySettings() {

**try** {

**int** newColumns = Integer.parseInt(columnsField.getText());

**int** symbolsToUse = 0;

**double** spinSpeed = 1.0;

**double** simulationSpeed = 3.0;

**if** (symbolsToUseField != **null** && !symbolsToUseField.getText().isEmpty()) {

symbolsToUse = Integer.parseInt(symbolsToUseField.getText());

**if** (symbolsToUse < 1 || symbolsToUse > 8) {

showAlert("Ошибка", "Количество фото должно быть от 1 до 8");

**return**;

}

}

**if** (spinSpeedField != **null** && !spinSpeedField.getText().isEmpty()) {

spinSpeed = Double.parseDouble(spinSpeedField.getText());

**if** (spinSpeed <= 0.01 || spinSpeed > 10) {

showAlert("Ошибка", "Скорость прокрутки должна быть от 0.01 до 10\n(чем меньше — тем медленнее)");

**return**;

}

}

**if** (simulationSpeedField != **null** && !simulationSpeedField.getText().isEmpty()) {

simulationSpeed = Double.parseDouble(simulationSpeedField.getText());

**if** (simulationSpeed < 0.5 || simulationSpeed > 30) {

showAlert("Ошибка", "Длительность симуляции должна быть от 0.5 до 30 секунд");

**return**;

}

}

**if** (newColumns > 0 && newColumns <= 10) {

columnsProperty.set(newColumns);

symbolsToUseProperty.set(symbolsToUse);

spinSpeedProperty.set(spinSpeed);

simulationSpeedProperty.set(simulationSpeed);

createGameArea();

} **else** {

showAlert("Ошибка", "Количество колонок должно быть от 1 до 10");

}

} **catch** (NumberFormatException e) {

showAlert("Ошибка", "Введите корректные числовые значения");

}

}

**private** **void** updateRowsBasedOnColumns(**int** columns) {

**int** rows = (columns <= 5) ? 1 : 2;

rowsProperty.set(rows);

}

**private** **void** createGameArea() {

**if** (currentState == GameState.SPINNING) {

**return**;

}

**if** (gameArea != **null**) {

gameArea.getChildren().clear();

}

reels = **new** ArrayList<>();

reelSymbolIndices = **new** ArrayList<>(); *// Инициализируем новое поле*

**if** (usedImages.isEmpty()) {

**if** (gameArea != **null**) {

Label noImagesLabel = **new** Label("Нет изображений для отображения.\nПоместите изображения в папку 'images' и нажмите 'Загрузить изображения'");

noImagesLabel.setAlignment(Pos.CENTER);

noImagesLabel.setStyle(

"-fx-text-alignment: center;" +

"-fx-text-fill: white;" +

"-fx-font-size: 16px;" +

"-fx-font-weight: bold;" +

"-fx-background-color: rgba(0, 0, 0, 0.7);" +

"-fx-padding: 20;" +

"-fx-background-radius: 10;"

);

gameArea.getChildren().add(noImagesLabel);

}

**return**;

}

**if** (gameArea != **null**) {

GridPane reelsContainer = **new** GridPane();

reelsContainer.setAlignment(Pos.CENTER);

reelsContainer.setHgap(15);

reelsContainer.setVgap(15);

reelsContainer.setPadding(**new** Insets(30));

reelsContainer.setStyle(

"-fx-background-color: linear-gradient(from 0% 0% to 100% 100%, #006400, #008000);" +

"-fx-border-color: #FFD700;" +

"-fx-border-width: 5;" +

"-fx-border-radius: 15;" +

"-fx-background-radius: 15;" +

"-fx-effect: dropshadow(gaussian, black, 10, 0, 3, 3);"

);

**int** totalColumns = columnsProperty.get();

**int** topRowColumns, bottomRowColumns;

**if** (totalColumns <= 5) {

topRowColumns = totalColumns;

bottomRowColumns = 0;

} **else** {

topRowColumns = (**int**) Math.ceil(totalColumns / 2.0);

bottomRowColumns = totalColumns - topRowColumns;

}

**int** columnIndex = 0;

**for** (**int** col = 0; col < topRowColumns; col++) {

VBox column = createColumn(columnIndex);

reelsContainer.add(column, col, 0);

columnIndex++;

}

**if** (bottomRowColumns > 0) {

**for** (**int** col = 0; col < bottomRowColumns; col++) {

VBox column = createColumn(columnIndex);

reelsContainer.add(column, col, 1);

columnIndex++;

}

}

gameArea.getChildren().add(reelsContainer);

}

updateProbability();

}

**private** VBox createColumn(**int** colIndex) {

VBox column = **new** VBox();

column.setAlignment(Pos.CENTER);

Pane slotCell = **new** Pane();

slotCell.setPrefSize(110, 110);

Rectangle clip = **new** Rectangle(110, 110);

slotCell.setClip(clip);

VBox reelBox = **new** VBox();

reelBox.setSpacing(0);

List<Integer> columnIndices = **new** ArrayList<>();

**for** (**int** i = 0; i < 3; i++) {

**int** randomIndex = rng.nextInt(usedImages.size());

Image randomImage = usedImages.get(randomIndex);

ImageView img = **new** ImageView(randomImage);

img.setFitWidth(110);

img.setFitHeight(110);

img.setPreserveRatio(**false**);

reelBox.getChildren().add(img);

columnIndices.add(randomIndex);

}

slotCell.getChildren().add(reelBox);

column.getChildren().add(slotCell);

reels.add(reelBox);

reelSymbolIndices.add(columnIndices);

**return** column;

}

**private** **void** executeCommand(Command command) {

**if** (command.canExecute(currentState)) {

command.execute();

}

}

**private** **class** StartSpinCommand **implements** Command {

@Override

**public** **void** execute() {

startSpin();

}

@Override

**public** **boolean** canExecute(GameState currentState) {

**return** currentState == GameState.IDLE && !usedImages.isEmpty();

}

}

**private** **void** startSpin() {

**if** (currentState != GameState.IDLE || usedImages.isEmpty()) **return**;

currentState = GameState.SPINNING;

isSpinningProperty.set(**true**);

**if** (resultLabel != **null**) {

resultLabel.setText("");

}

**if** (startButton != **null**) {

startButton.setStyle(

"-fx-background-color: linear-gradient(from 0% 0% to 100% 100%, #808080, #696969);" +

"-fx-text-fill: white;" +

"-fx-font-weight: bold;" +

"-fx-font-size: 16px;" +

"-fx-border-color: #2F4F4F;" +

"-fx-border-width: 2;" +

"-fx-border-radius: 5;" +

"-fx-background-radius: 5;"

);

}

**double** totalSimulationSeconds = simulationSpeedProperty.get();

**double** spinIntensity = spinSpeedProperty.get();

**long** totalSimulationMs = (**long**) (totalSimulationSeconds \* 1000);

**long** stepDurationMs = (**long**) (200 / spinIntensity);

*// Сбрасываем счётчик активных анимаций*

activeAnimations.set(0);

*// Запускаем все барабаны одновременно*

**for** (**int** col = 0; col < columnsProperty.get(); col++) {

**int** finalCol = col;

**long** delayMs = (**long**) (col \* (totalSimulationMs / (**double**) columnsProperty.get() / 4.0));

PauseTransition delay = **new** PauseTransition(Duration.millis(delayMs));

delay.setOnFinished(e -> spinColumn(finalCol, stepDurationMs, totalSimulationMs));

delay.play();

}

*// Планируем остановку и проверку результата*

Timeline stopTimeline = **new** Timeline(**new** KeyFrame(

Duration.millis(totalSimulationMs),

e -> {

*// Останавливаем логику — новые анимации не запускаются*

currentState = GameState.IDLE;

*// Начинаем асинхронное ожидание завершения активных анимаций*

waitForAnimationsToFinish();

}

));

stopTimeline.play();

}

**private** **void** waitForAnimationsToFinish() {

**if** (activeAnimations.get() == 0) {

Platform.runLater(**this**::showFinalResult);

} **else** {

PauseTransition wait = **new** PauseTransition(Duration.millis(20));

wait.setOnFinished(e -> waitForAnimationsToFinish());

wait.play();

}

}

*// Показываем результат, когда всё готово*

**private** **void** showFinalResult() {

**try** {

checkWin();

} **catch** (Exception ex) {

ex.printStackTrace();

showAlert("Ошибка", "Во время подсчёта результата произошла ошибка:\n" + ex.getMessage());

} **finally** {

isSpinningProperty.set(**false**);

**if** (startButton != **null**) {

startButton.setStyle(

"-fx-background-color: linear-gradient(from 0% 0% to 100% 100%, #FF4500, #DC143C);" +

"-fx-text-fill: white;" +

"-fx-font-weight: bold;" +

"-fx-font-size: 16px;" +

"-fx-border-color: #8B0000;" +

"-fx-border-width: 2;" +

"-fx-border-radius: 5;" +

"-fx-background-radius: 5;"

);

}

}

}

**private** **void** spinColumn(**int** col, **long** stepDurationMs, **long** totalSimulationMs) {

**if** (col >= reels.size() || col >= reelSymbolIndices.size()) **return**;

activeAnimations.incrementAndGet();

VBox reelBox = reels.get(col);

List<Integer> columnIndices = reelSymbolIndices.get(col);

TranslateTransition spin = **new** TranslateTransition(Duration.millis(stepDurationMs), reelBox);

spin.setFromY(0);

spin.setToY(-110);

spin.setInterpolator(Interpolator.LINEAR);

spin.setOnFinished(e -> {

**try** {

Node first = reelBox.getChildren().remove(0);

**if** (first **instanceof** ImageView) {

**int** newIndex = rng.nextInt(usedImages.size());

Image newImage = usedImages.get(newIndex);

((ImageView) first).setImage(newImage);

**if** (col < reelSymbolIndices.size()) {

List<Integer> indices = reelSymbolIndices.get(col);

**if** (!indices.isEmpty()) {

indices.remove(0);

indices.add(newIndex);

}

}

}

reelBox.getChildren().add(first);

reelBox.setTranslateY(0);

} **finally** {

activeAnimations.decrementAndGet();

}

*// Продолжаем только если игра ещё идёт*

**if** (currentState == GameState.SPINNING) {

spinColumn(col, stepDurationMs, totalSimulationMs);

} **else** {

}

});

spin.play();

}

**private** **void** checkWin() {

**if** (columnsProperty.get() == 0 || usedImages.isEmpty() || reelSymbolIndices == **null**) {

**return**;

}

**boolean** isWin = **false**;

*// Собираем реальные индексы верхних символов*

List<Integer> topIndices = **new** ArrayList<>();

**for** (**int** col = 0; col < columnsProperty.get() && col < reelSymbolIndices.size(); col++) {

List<Integer> colIndices = reelSymbolIndices.get(col);

**if** (colIndices != **null** && !colIndices.isEmpty()) {

**int** topIndex = colIndices.get(0); *// первый в списке = верхний символ*

topIndices.add(topIndex);

*// Выводим URL изображения для проверки*

**if** (topIndex < usedImages.size()) {

String imgUrl = usedImages.get(topIndex).getUrl();

} **else** {

}

} **else** {

}

}

*// Для 2 колонок: выигрыш, если индексы совпадают*

**if** (columnsProperty.get() == 2 && topIndices.size() == 2) {

isWin = (topIndices.get(0).equals(topIndices.get(1)));

}

*// Для 3+ колонок: все должны совпадать*

**else** **if** (topIndices.size() >= 2) {

Integer first = topIndices.get(0);

isWin = **true**;

**for** (**int** i = 1; i < topIndices.size(); i++) {

**if** (!topIndices.get(i).equals(first)) {

isWin = **false**;

**break**;

}

}

}

*// Отображаем результат*

**if** (resultLabel != **null**) {

**if** (isWin) {

resultLabel.setText("🎰 Выигрыш! 🎰");

resultLabel.setStyle(

"-fx-text-fill: #FFD700;" +

"-fx-font-size: 22px;" +

"-fx-font-weight: bold;" +

"-fx-effect: dropshadow(gaussian, black, 3, 0, 1, 1);" +

"-fx-background-color: rgba(0, 100, 0, 0.7);" +

"-fx-padding: 10;" +

"-fx-background-radius: 10;"

);

} **else** {

resultLabel.setText("Попробуйте ещё раз!");

resultLabel.setStyle(

"-fx-text-fill: #FF6347;" +

"-fx-font-size: 18px;" +

"-fx-font-weight: bold;" +

"-fx-effect: dropshadow(gaussian, black, 3, 0, 1, 1);" +

"-fx-background-color: rgba(139, 0, 0, 0.7);" +

"-fx-padding: 10;" +

"-fx-background-radius: 10;"

);

}

}

}

**private** **void** removeDuplicateImages() {

List<Image> uniqueImages = **new** ArrayList<>();

Set<String> seenUrls = **new** HashSet<>();

**for** (Image img : gameImages) {

String url = img.getUrl();

**if** (!seenUrls.contains(url)) {

seenUrls.add(url);

uniqueImages.add(img);

} **else** {

}

}

gameImages = uniqueImages;

}

**private** **void** updateProbability() {

**if** (usedImages.isEmpty()) {

**if** (probabilityLabel != **null**) {

probabilityLabel.setText("Вероятность выигрыша: 0% (нет изображений)");

}

**return**;

}

**double** probability = probabilityCalculator.calculateWinProbability(

columnsProperty.get(),

rowsProperty.get(),

symbolCountProperty.get()

);

**if** (probabilityLabel != **null**) {

probabilityLabel.setText(String.format("🎲 Вероятность выигрыша: %.4f%% 🎲", probability \* 100));

}

}

**private** **void** showAlert(String title, String message) {

Alert alert = **new** Alert(Alert.AlertType.INFORMATION);

alert.setTitle(title);

alert.setHeaderText(**null**);

alert.setContentText(message);

DialogPane dialogPane = alert.getDialogPane();

dialogPane.setStyle(

"-fx-background-color: linear-gradient(from 0% 0% to 100% 100%, #DAA520, #B8860B);" +

"-fx-border-color: #FFD700;" +

"-fx-border-width: 2;" +

"-fx-border-radius: 5;"

);

alert.showAndWait();

}

}

**Command.java:**

**package** **dynamic.casino**;

**public** **interface** Command {

**void** execute();

**boolean** canExecute(GameState currentState);

}

**DefaultProbabilityCalculator.java:**

**package** **dynamic.casino**;

**public** **class** DefaultProbabilityCalculator **implements** ProbabilityCalculator {

@Override

**public** **double** calculateWinProbability(**int** columns, **int** rows, **int** symbolCount) {

**if** (columns <= 0 || rows <= 0 || symbolCount <= 0) {

**return** 0;

}

**double** prow = Math.pow(1.0 / symbolCount, columns - 1);

**double** pwin = 1 - Math.pow(1 - prow, rows);

**return** pwin;

}

}

**DefaultRandomGenerator.java:**

**package** **dynamic.casino**;

**import** **java.util.concurrent.ThreadLocalRandom**;

**public** **class** DefaultRandomGenerator **implements** RandomNumberGenerator {

@Override

**public** **int** nextInt(**int** bound) {

**return** ThreadLocalRandom.current().nextInt(bound);

}

@Override

**public** **double** nextDouble() {

**return** ThreadLocalRandom.current().nextDouble();

}

}

**GameState.java:**

**package** **dynamic.casino**;

**public** **enum** GameState {

IDLE,

SPINNING,

STOPPING

}

**Main.java:**

**package** **dynamic.casino**;

**import** **javafx.application.Application**;

**import** **javafx.fxml.FXMLLoader**;

**import** **javafx.scene.Parent**;

**import** **javafx.scene.Scene**;

**import** **javafx.stage.Stage**;

**import** **java.io.IOException**;

**public** **class** Main **extends** Application {

@Override

**public** **void** start(Stage primaryStage) **throws** IOException {

FXMLLoader loader = **new** FXMLLoader(getClass().getResource("/casino.fxml"));

Parent root = loader.load();

primaryStage.setTitle("Игровой автомат казино 777");

primaryStage.setScene(**new** Scene(root, 1250, 750));

primaryStage.show();

}

**public** **static** **void** main(String[] args) {

launch(args);

}

}

**ProbabilityCalculator.java:**

**package** **dynamic.casino**;

**public** **interface** ProbabilityCalculator {

**double** calculateWinProbability(**int** columns, **int** rows, **int** symbolCount);

}

**RandomNumberGenerator.java:**

**package** **dynamic.casino**;

**public** **interface** RandomNumberGenerator {

**int** nextInt(**int** bound);

**double** nextDouble();

}

**SymbolFactory.java:**

**package** **dynamic.casino**;

**import** **javafx.scene.image.Image**;

**import** **javafx.scene.image.ImageView**;

**import** **java.util.List**;

**public** **class** SymbolFactory {

**public** **static** ImageView createSymbol(List<Image> gameImages, RandomNumberGenerator rng) {

ImageView imageView = **new** ImageView();

**if** (!gameImages.isEmpty()) {

Image randomImage = gameImages.get(rng.nextInt(gameImages.size()));

imageView.setImage(randomImage);

}

**return** imageView;

}

**public** **static** ImageView createSymbol(List<Image> gameImages, RandomNumberGenerator rng,

**double** fitWidth, **double** fitHeight) {

ImageView imageView = createSymbol(gameImages, rng);

imageView.setFitWidth(fitWidth);

imageView.setFitHeight(fitHeight);

imageView.setPreserveRatio(**false**);

imageView.setSmooth(**true**);

**return** imageView;

}

}

**casino.fxml:**

<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>

<?import javafx.scene.control.\*?>

<?import javafx.scene.layout.\*?>

<?import javafx.scene.text.Font?>

**<VBox** fx:id="mainContainer" prefHeight="700.0" prefWidth="1200.0" xmlns="http://javafx.com/javafx/19" xmlns:fx="http://javafx.com/fxml/1" fx:controller="dynamic.casino.CasinoController"**>**

**<children>**

**<HBox** prefHeight="600.0" prefWidth="1200.0" VBox.vgrow="ALWAYS"**>**

**<children>**

*<!-- Панель настроек -->*

**<VBox** fx:id="settingsPanel" prefWidth="280.0" spacing="20.0"**>**

**<children>**

**<Label** text="🎰 НАСТРОЙКИ КАЗИНО 🎰" textAlignment="CENTER" wrapText="true"**>**

**<font>**

**<Font** name="System Bold" size="18.0" **/>**

**</font>**

**</Label>**

**<Separator** **/>**

**<Label** text="Количество колонок:" **/>**

**<TextField** fx:id="columnsField" promptText="3" text="3" **/>**

**<Label** text="Количество фото:" **/>**

**<TextField** fx:id="symbolsToUseField" promptText="0" text="0" **/>**

**<Label** text="Скорость прокрутки (0.01-10):" **/>**

**<TextField** fx:id="spinSpeedField" promptText="1.0" text="1.0" **/>**

**<Label** text="Длительность симуляции (сек):" **/>**

**<TextField** fx:id="simulationSpeedField" promptText="3.0" text="3.0" **/>**

**<Button** fx:id="applySettingsButton" mnemonicParsing="false" text="Применить настройки" **/>**

**<Separator** **/>**

**<Button** fx:id="loadImagesButton" mnemonicParsing="false" text="Загрузить изображения" **/>**

**<Label** fx:id="statusLabel" text="Готов к загрузке изображений" wrapText="true"**>**

**<font>**

**<Font** size="12.0" **/>**

**</font>**

**</Label>**

**<Separator** **/>**

**<Label** fx:id="probabilityLabel" text="🎲 Вероятность выигрыша: " wrapText="true"**>**

**<font>**

**<Font** size="14.0" **/>**

**</font>**

**</Label>**

**<Separator** **/>**

**<Button** fx:id="startButton" mnemonicParsing="false" text="🎰 ЗАПУСТИТЬ ИГРУ 🎰" **/>**

**<Separator** **/>**

**<Label** fx:id="resultLabel" text="" wrapText="true" alignment="CENTER"**>**

**<font>**

**<Font** size="16.0" **/>**

**</font>**

**</Label>**

**</children>**

**<padding>**

**<Insets** bottom="15.0" left="15.0" right="15.0" top="15.0" **/>**

**</padding>**

**</VBox>**

*<!-- Игровая область -->*

**<VBox** fx:id="gameArea" alignment="CENTER" HBox.hgrow="ALWAYS" spacing="25.0"**>**

**<padding>**

**<Insets** bottom="25.0" left="25.0" right="25.0" top="25.0" **/>**

**</padding>**

**</VBox>**

**</children>**

**</HBox>**

**</children>**

**</VBox>**

**pom.xml:**

<?xml version="1.0" encoding="UTF-8"?>

**<project** xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd"**>**

**<modelVersion>**4.0.0**</modelVersion>**

**<groupId>**dynamic**</groupId>**

**<artifactId>**casino**</artifactId>**

**<version>**1.0-SNAPSHOT**</version>**

**<properties>**

**<maven.compiler.source>**11**</maven.compiler.source>**

**<maven.compiler.target>**11**</maven.compiler.target>**

**<project.build.sourceEncoding>**UTF-8**</project.build.sourceEncoding>**

**<javafx.version>**17.0.2**</javafx.version>**

**</properties>**

**<dependencies>**

**<dependency>**

**<groupId>**org.openjfx**</groupId>**

**<artifactId>**javafx-controls**</artifactId>**

**<version>**${javafx.version}**</version>**

**</dependency>**

**<dependency>**

**<groupId>**org.openjfx**</groupId>**

**<artifactId>**javafx-fxml**</artifactId>**

**<version>**${javafx.version}**</version>**

**</dependency>**

**</dependencies>**

**<build>**

**<resources>**

**<resource>**

**<directory>**src/main/resources**</directory>**

**<includes>**

**<include>**\*\*/\*.fxml**</include>**

**</includes>**

**</resource>**

**</resources>**

**<plugins>**

**<plugin>**

**<groupId>**org.apache.maven.plugins**</groupId>**

**<artifactId>**maven-compiler-plugin**</artifactId>**

**<version>**3.8.1**</version>**

**<configuration>**

**<source>**11**</source>**

**<target>**11**</target>**

**</configuration>**

**</plugin>**

**<plugin>**

**<groupId>**org.openjfx**</groupId>**

**<artifactId>**javafx-maven-plugin**</artifactId>**

**<version>**0.0.8**</version>**

**<configuration>**

**<mainClass>**dynamic.casino.Main**</mainClass>**

**</configuration>**

**</plugin>**

**</plugins>**

**</build>**

**</project>**