ФГБОУ ВО «Уральский государственный горный университет»

Инженерно-экономический факультет

Кафедра информатики

Курсовой проект

По дисциплине «Программная инженерия»

На тему «Bubble Bobble»

Выполнил:

Студент гр. АУБП-23

Монахов И.А.

Проверил:

Доцент каф. информатики

Волкова Е.А

Екатеринбург, 2025г.

Оглавление

[**1.1.** **Формулировка задания** 3](#_Toc199096243)

[**1.2.** **Алгоритмическое решения задачи** 4](#_Toc199096244)

[**1.2.1.** **Основной игровой цикл** 5](#_Toc199096245)

[**1.2.2.** **Реализация игрового движка с основными игровыми механиками** 7](#_Toc199096246)

[1.2.2.1. Физика движения и столконвения 7](#_Toc199096247)

[2.2.1.1. Логика сущностей 9](#_Toc199096248)

[**3.** **Решение задачи** 12](#_Toc199096249)

[**2.1.** **Основной игровой цикл** 12](#_Toc199096250)

[**2.2** **Описание основных классов** 13](#_Toc199096251)

[**2.3** **Интерфейс приложения** 19](#_Toc199096252)

[**3.** **Контрольны тесты** 21](#_Toc199096253)

[**3.1** **Контрольный робот** 21](#_Toc199096254)

[**3.2** **Работа приложения на контрольных тестах** 25](#_Toc199096255)

[**3.3** **Результат работы** 26](#_Toc199096256)

# **Формулировка задания**

Реализовать аркадную игру «Bubble Bobble». Игрок управляет одним из драконов. Игрок может двигаться по платформам на уровне и перепрыгивать с одной платформы на другую. В качестве оружия против врагов игрок может использовать пузырьки, которые дракон выпускает из пасти. Враг попадает в ловушку внутри пузыря и может быть обезврежен, если затем прыгнуть на пузырь сбоку или сверху. При этом враг, находящийся в пузыре, превратится во что-нибудь съедобное (чаще всего бананы), за съедение даются очки.

Пример игрового поля:

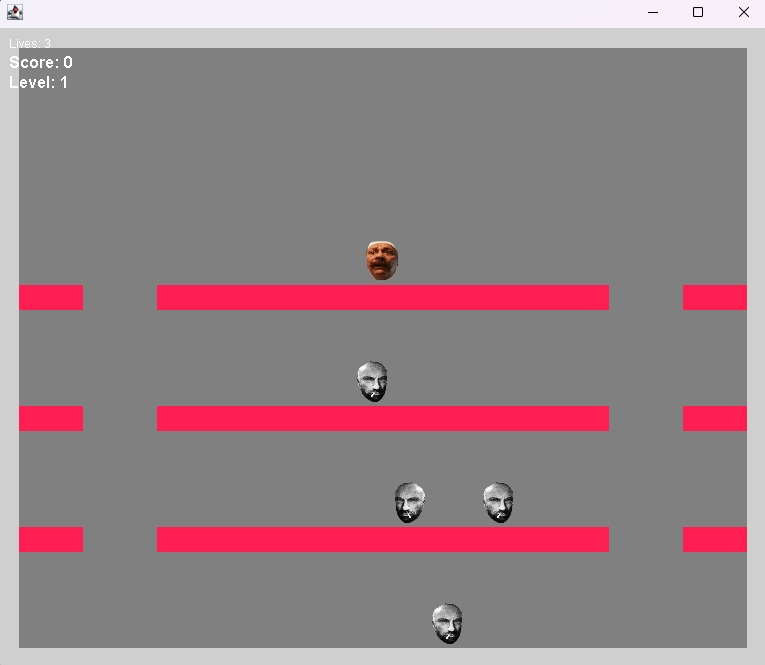


Рисунок 1. Пример поля

**Постановка задачи**

Написать программу реализующую игру «Bubble Bobble».

# **Алгоритмическое решения задачи**

Для реализации игры были использованы следующие библиотеки и зависимости:

**Java Standard Library (встроенные пакеты):**

* java.awt.\* - для графики и работы с окнами
* java.awt.event.\* - для обработки событий
* javax.swing.\* - для GUI компонентов
* java.imageio.\* - для работы с изображениями

**Lambok – для сокращения шаблонного кода:**

* Аннотации @Data, @AllArgsConstructor,

@ NoArgsConstructor.

**Maven – система сборки проекта.**

## **Основной игровой цикл**

Основной игровой цикл реализован в классе GamePanel, который наследуется от JPanel

1. Инициализация игрового цикла

Задается целевое время одного кадра:

Rt = 1 секунда / 75 FPS ≈ 0.0133 сек (13.3 мс).

Запоминается время следующего кадра: Nrt =

текущее время + Rt.

1. Главный цикл while

Обновление состояния игры update:

Проверяет, не закончилась ли игра.

Обновляет позицию игрока, врагов, пузырей и фруктов.

Обрабатывает столкновения и логику (например, переход уровня).

Отрисовка repaint:

Автоматически вызывает paintComponent, который рисует все объекты.

Синхронизация FPS:

Вычисляет оставшееся время до следующего кадра: remTime = Nrt - текущее время.

Если цикл успел раньше (remTime > 0), поток «засыпает» на это время.

Если цикл отстает (remTime < 0), пропускает паузу. Планирует следующий кадр: Nrt += Rt.

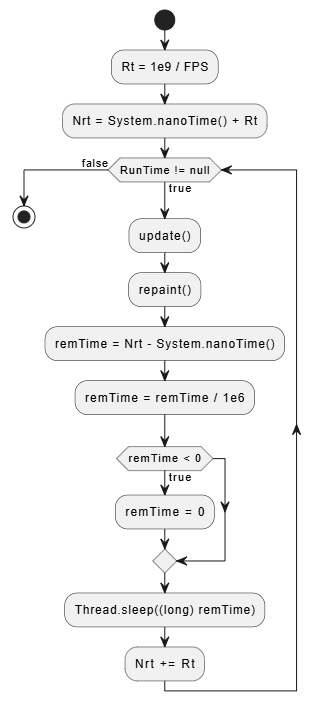


Рисунок 2. Блок-схема метода run

## **Реализация игрового движка с основными игровыми механиками**

## Физика движения и столконвения

1. Вертикальная коллизия.

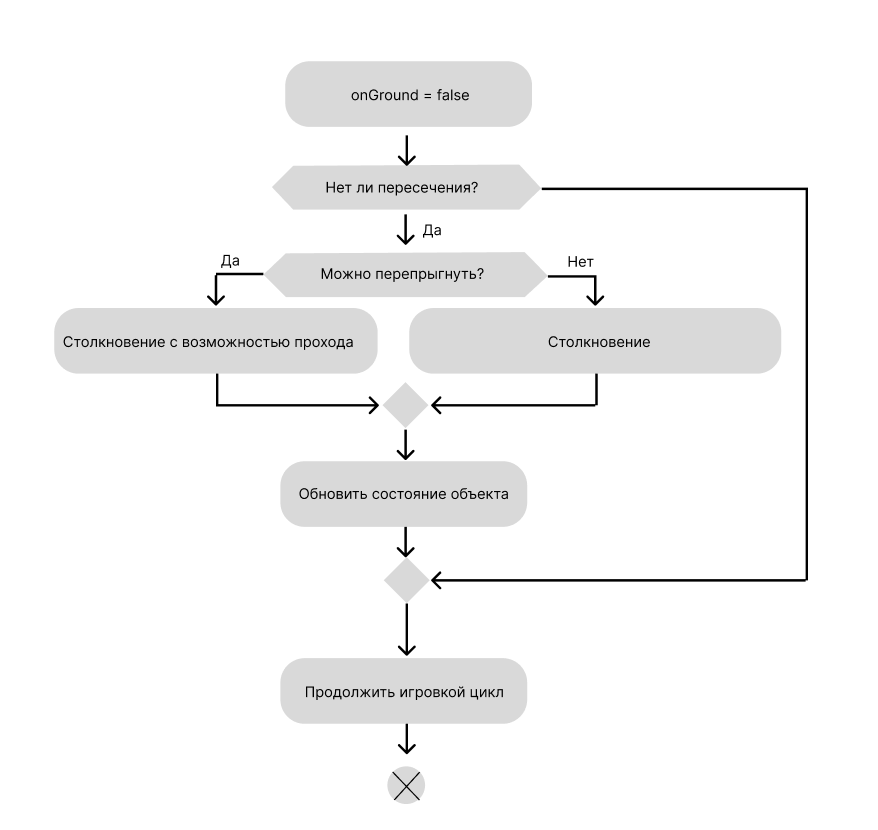


Рисунок 3. Блок-схема метода verticalCollision

1. Горизонтальная коллизия.

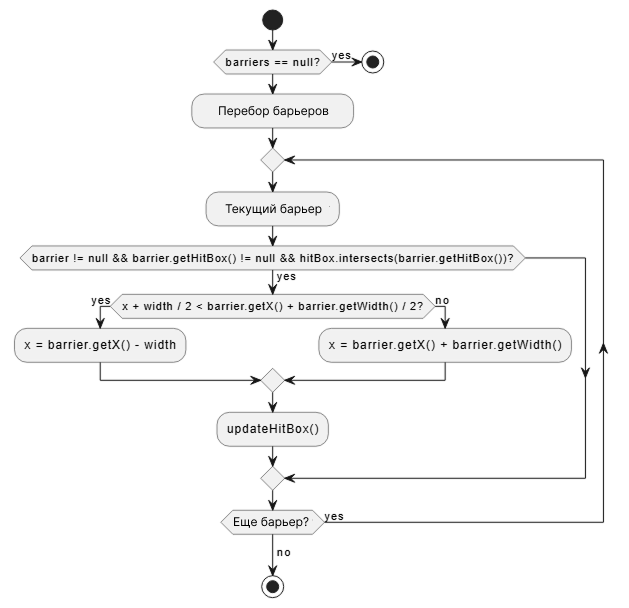


Рисунок 4. Блок-схема метода horizontalCollision

## Логика сущностей

1. Главный герой (управление персонажем)

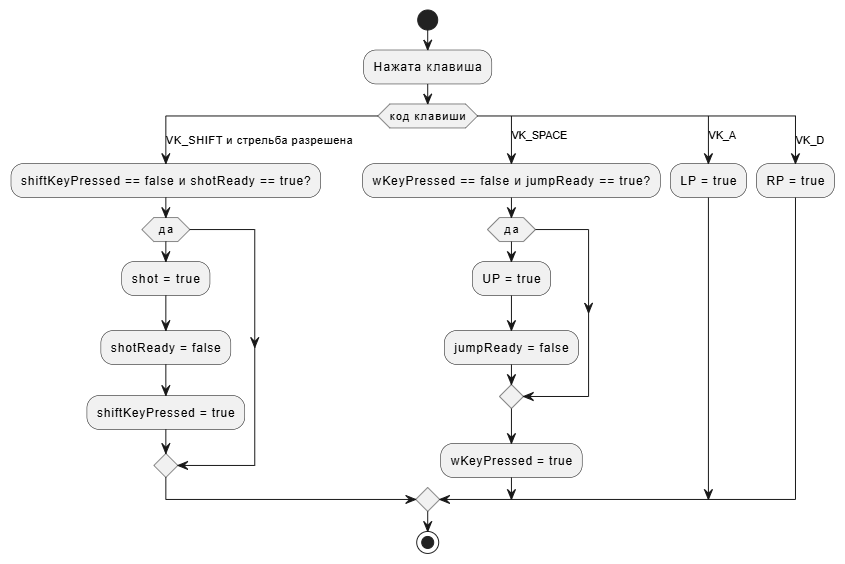


Рисунок 5. Блок-схема класса Control

1. Враги (монстры)

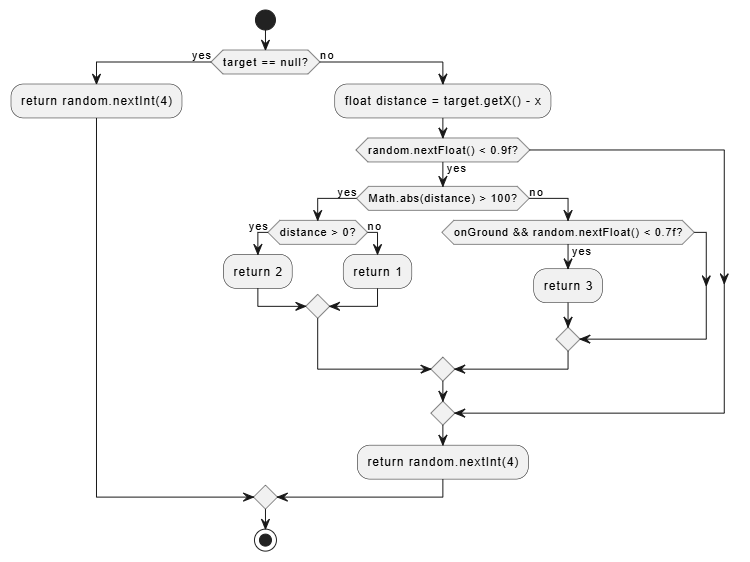


Рисунок 6. Блок-схема метода makeDecision

1. Пузыри (оружие игрока)

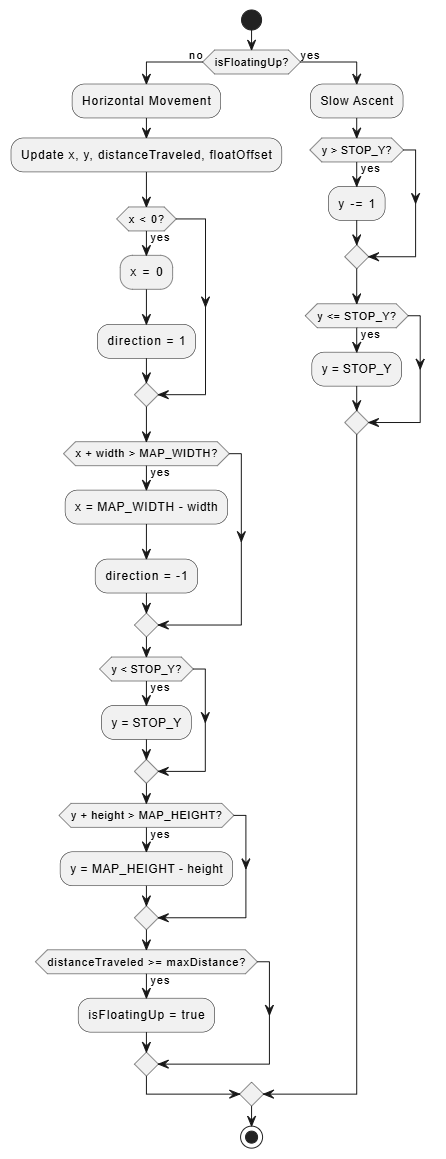


Рисунок 7. Блок-схема метода updateEmptyBubble

1. **Решение задачи**

## **Основной игровой цикл**

Выбор Java для реализации игры Bubble Bobble обусловлен рядом ключевых факторов, которые делают этот язык программирования оптимальным решением для данного проекта. Во-первых, Java является кроссплатформенным языком, что позволяет запускать игру на различных операционных системах без необходимости внесения изменений в исходный код. Это особенно важно для indie-игр, где целевая аудитория может использовать разные платформы. Во-вторых, Java предоставляет богатый набор стандартных библиотек, включая Swing, которые идеально подходят для создания 2D-игр с относительно простой графикой. Эти библиотеки позволяют реализовать отрисовку спрайтов, обработку пользовательского ввода и базовую физику без необходимости подключения дополнительных движков.

## **Описание основных классов**

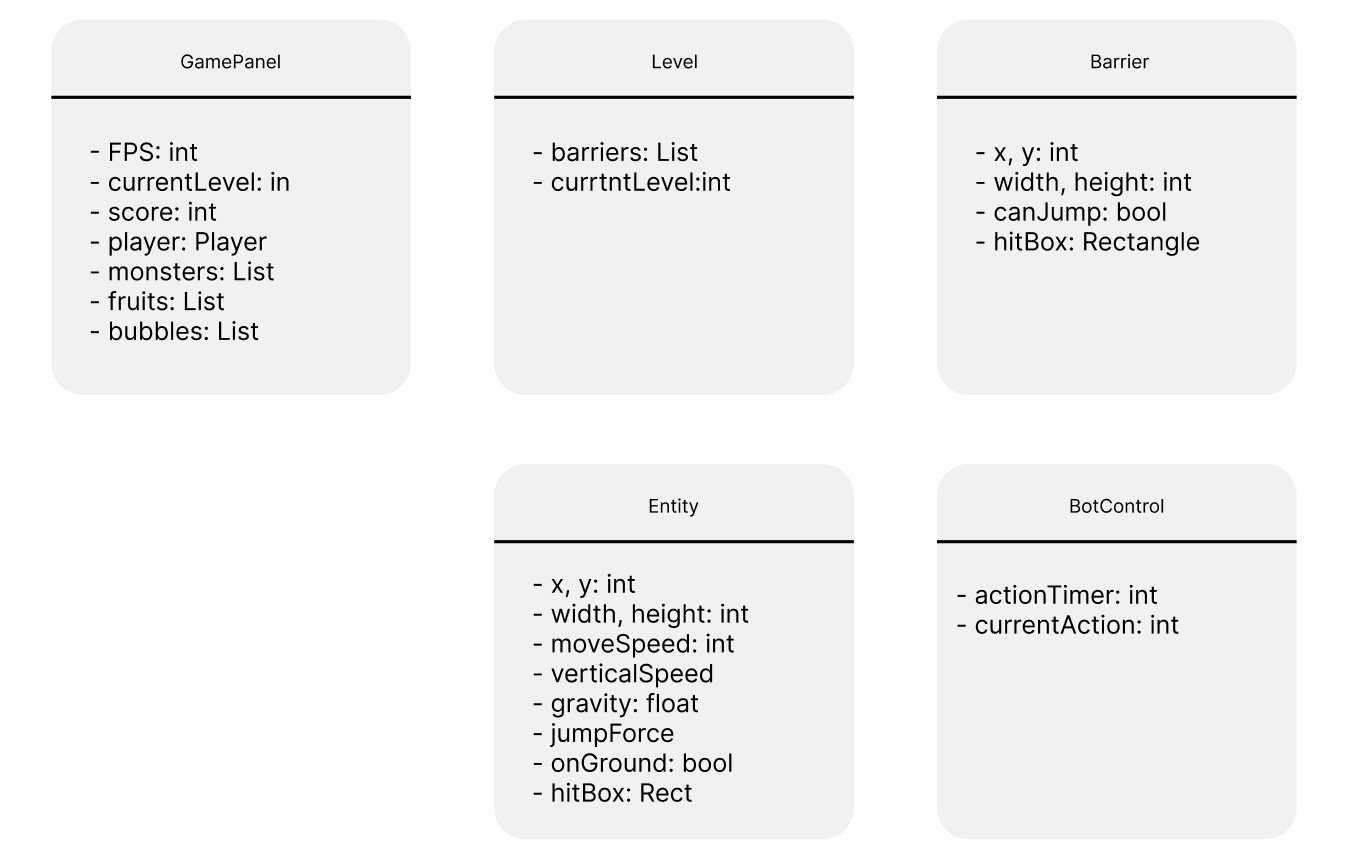


Рисунок 8. Диаграмма классов

Класс GamePanel

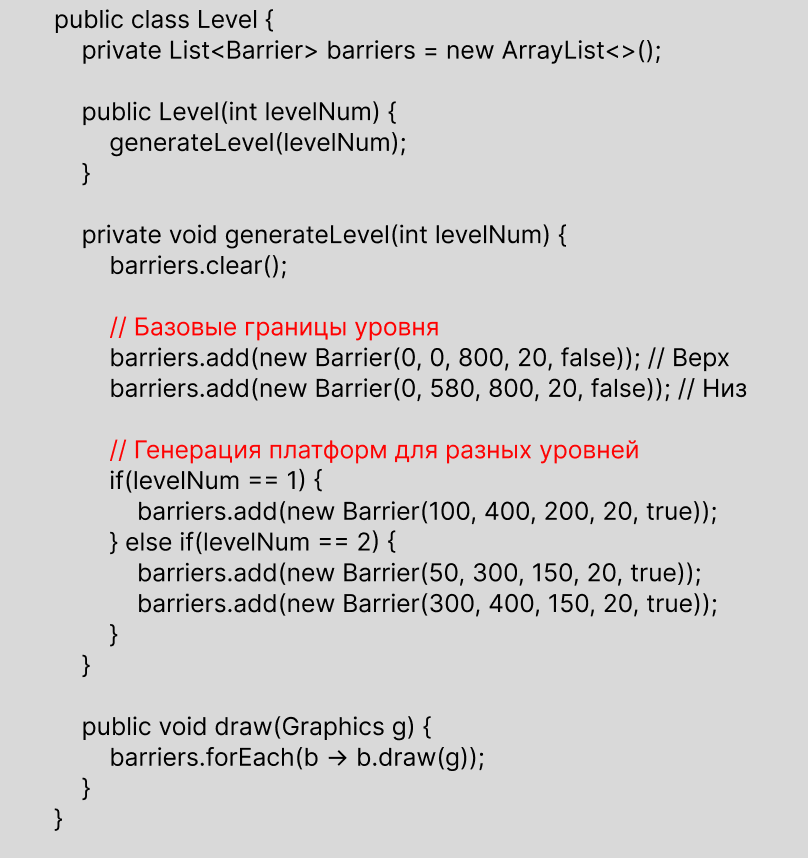
GamePanel является центральным классом игры, наследующимся от JPanel. Он выполняет функции игрового движка, включая управление основным игровым циклом с фиксированной частотой кадров (75 FPS), обработку ввода пользователя через KeyboardListener, обновление состояния всех игровых объектов и их отрисовку. В его обязанности входит создание и управление экземплярами Player, Monster, Bubble и других игровых объектов, обработка коллизий между ними, а также контроль перехода между уровнями. Класс хранит текущее состояние игры (счет, уровень, жизни игрока) и отображает соответствующую информацию на экране.



Листинг 1. Класс GamePanel

Класс Level

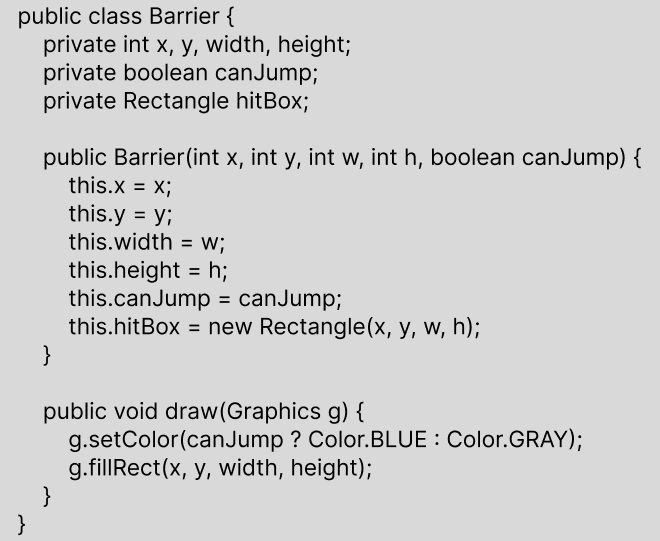
Level отвечает за генерацию и управление игровыми уровнями. Для каждого уровня (всего их три) класс создает уникальную конфигурацию платформ (Barrier) с разным расположением и свойствами. Уровни отличаются сложностью - на первом уровне платформы расположены в виде простых горизонтальных линий, на втором появляются разрывы, а на третьем - более сложные конструкции. Класс также определяет границы игрового поля и проверяет завершение уровня (когда все враги побеждены).



Листинг 2. Класс Level.

Класс Barrier

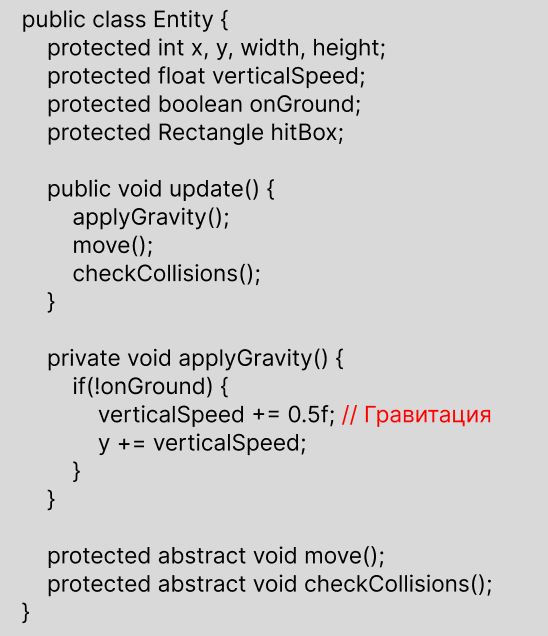
Barrier представляет собой игровые платформы и препятствия. Каждый барьер имеет координаты (x, y), размеры (width, height) и флаг canJump, определяющий его проходимость. Непроходимые барьеры (canJump=false) полностью блокируют движение, в то время как проходимые позволяют игроку и врагам прыгать через них снизу вверх. Класс содержит прямоугольник hitBox для обработки коллизий и метод draw() для отрисовки платформ разными цветами в зависимости от их типа.



Листинг 3. Класс Barrier

Класс Entity

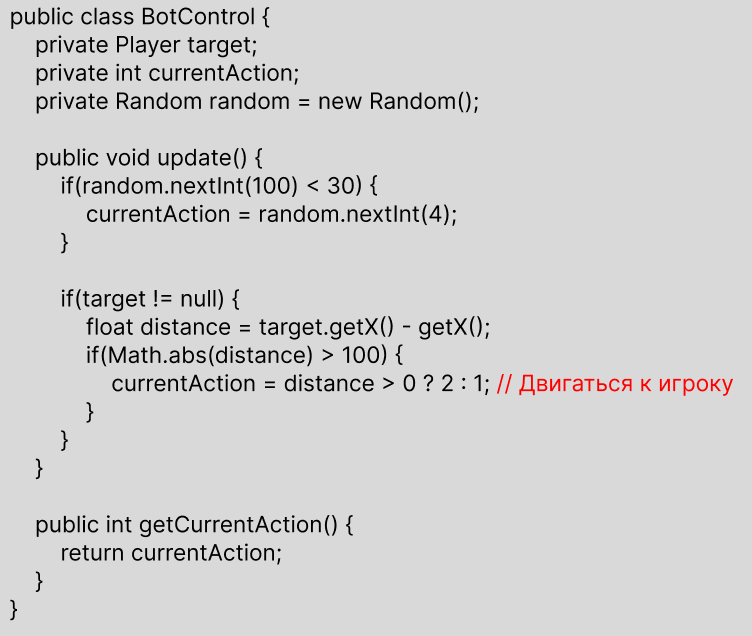
Entity является базовым классом для всех движущихся объектов игры (Player, Monster, Bubble, Fruit). Он реализует базовую физику: гравитацию (verticalSpeed += gravity), прыжки (применение силы jumpForce), обработку столкновений с платформами (методы horizontalCollision и verticalCollision). Класс содержит общие для всех объектов свойства: координаты, размеры, скорость, состояние (onGround), а также hitBox для обнаружения столкновений. Методы horizontalMovement и verticalMovement обеспечивают передвижение объектов в ответ на управляющие сигналы.



Листинг 4. Класс Entity.

Класс BotControl

BotControl реализует искусственный интеллект для управления врагами (Monster). Используя простую систему принятия решений (makeDecision), ИИ выбирает одно из четырех действий: стоять на месте, двигаться влево, вправо или прыгать. Решения принимаются на основе расстояния до игрока (target) и случайного фактора. Каждое действие выполняется в течение случайного промежутка времени (actionTimer), что создает эффект непредсказуемого поведения. Метод applyAction преобразует выбранное действие в управляющие сигналы (LP, RP, UP), аналогичные тем, что генерируются клавиатурой для игрока.



Листинг 5. Класс BotControl.

## **Интерфейс приложения**

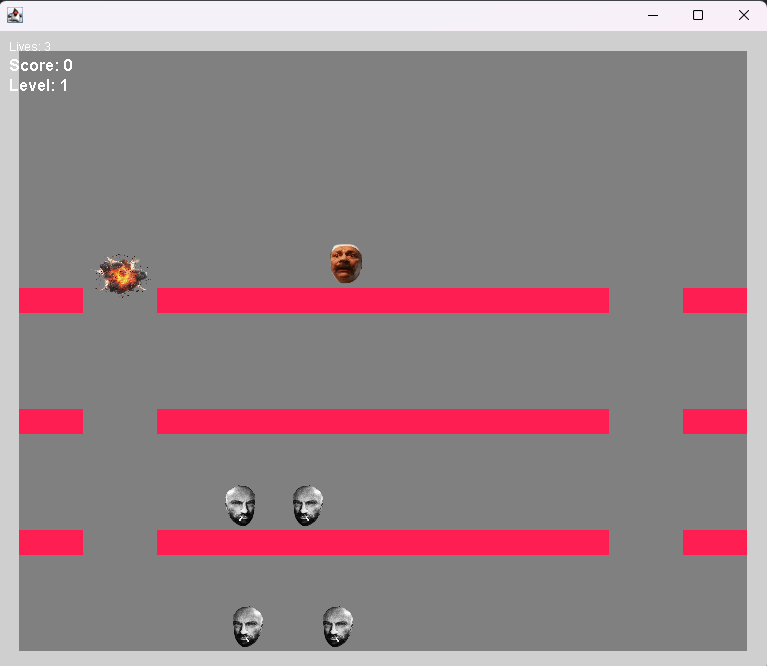
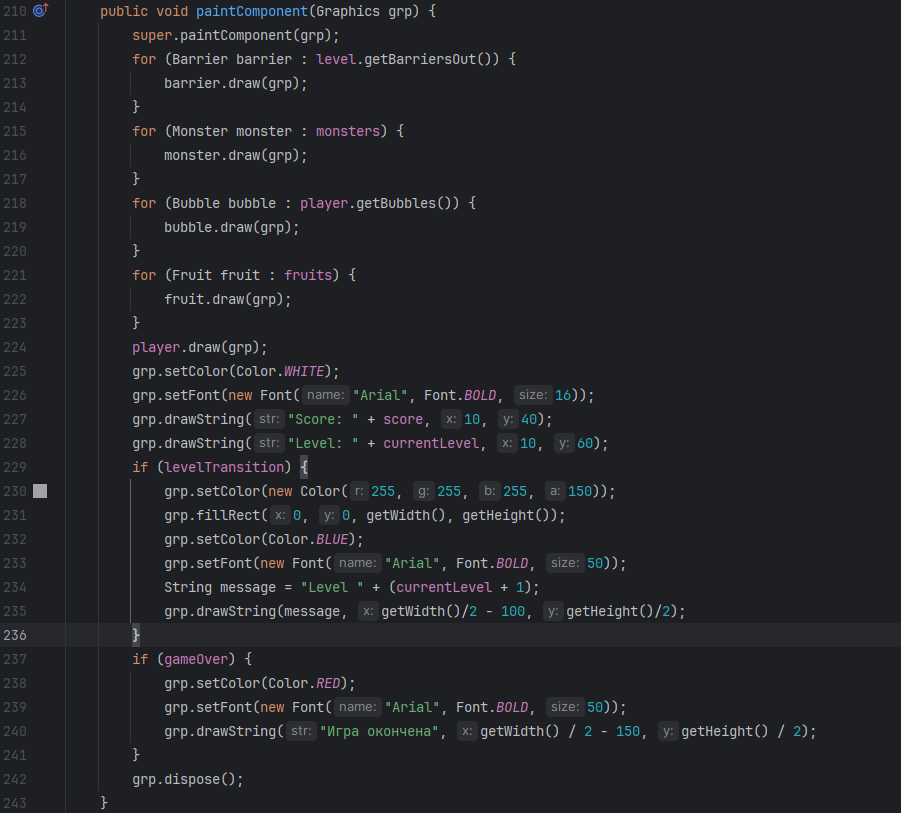
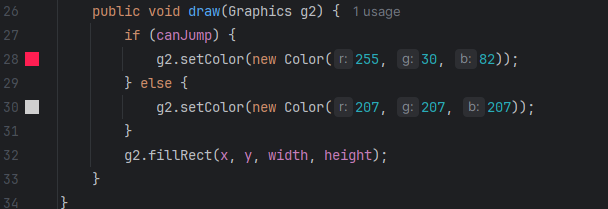


Рисунок 9. Интерфейс приложения

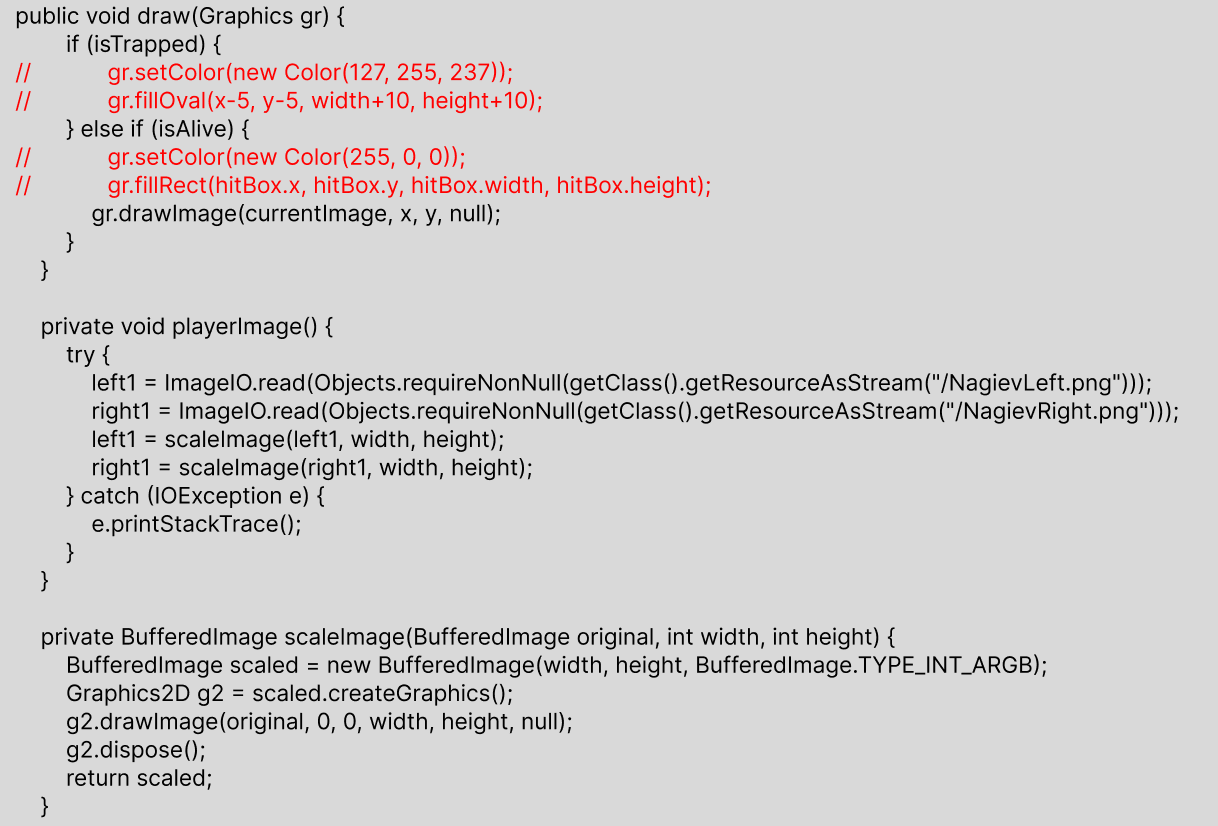
Отрисовка в игре реализована через стандартный Swing-подход с использованием: JPanel и Метода paintComponent для кастомного рендеринга.



Листинг 6. Метод paintComponent.



Листинг 7. Отрисовка платформ.



Листинг 8. Отрисовка игрока.

1. **Контрольны тесты**

## **Контрольный робот**

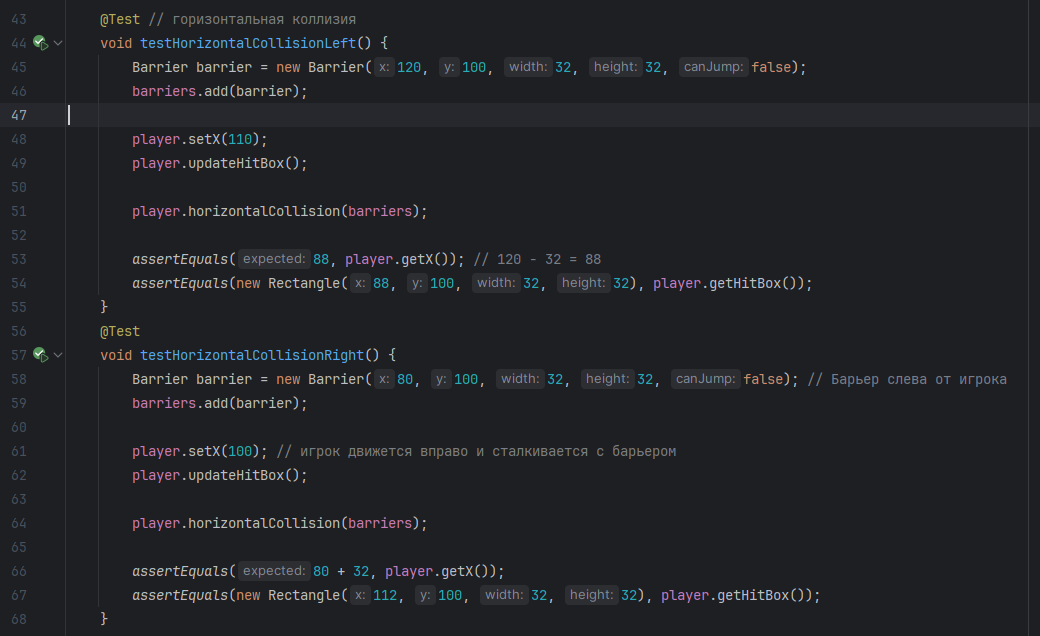
Для тестирования мы используем фреймворк JUnit 5, который является современным инструментом для написания и выполнения юнит-тестов в Java, обеспечивая аннотации, такие как @Test и @BeforeEach, для структурирования тестовых методов и подготовки данных.

Тестирлвал класс BotControl, который отвечает за принятие решений ботом на основе положения цели (target), состояния onGround и текущего положения x, а также класс Entity (через Player), который реализует базовую логику движения, коллизий и обновления направления.



Листинг 9. Тест BotControl.

Тесты testHorizontalCollisionLeft и testHorizontalCollisionRight проверяют, что игрок правильно останавливается при столкновении с барьером, в зависимости от того, с какой стороны он находится (слева или справа). Проверяется как изменение координаты x, так и обновление хитбокса.



Листинг 10. Тесты горизонтальной коллизии.

Тест testVerticalCollisionJumpPlatform проверяет случай, когда столкновения нет, чтобы убедиться, что состояние игрока (позиция y, скорость, onGround) остается неизменным. Это важно для корректной работы гравитации и движения, когда игрок находится в воздухе вдали от препятствий.



Листинг 11. Тесты вертикальной коллизии.

## **Работа приложения на контрольных тестах**

Все тесты, написанные для классов «Entity», «BotControl», проходят успешно и в указанные временные рамки и можно сделать выводы, что всё работает корректно.

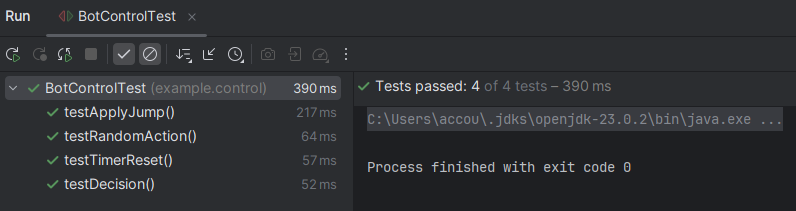


Рисунок 10. Тест класса BotControl.

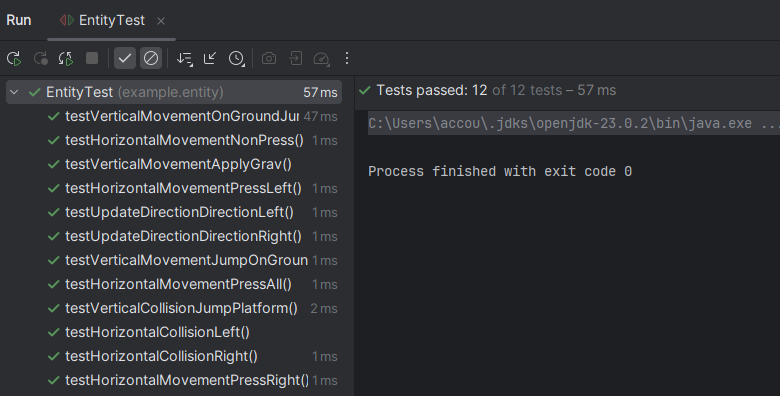


Рисунок 11. Тест класса Entity.

## **Результат работы**

В результате выполнения курсовой работы было разработано приложение, которое реализует игру «Bubble Bobble». В игре реализовано множество функций, которые способствуют созданию полноценного игрового процесса. Все основные игровые механики реализованы в полном объеме и соответствуют техническому заданию.