ГОСУДАРСВЕННОЕ БЮДЖЕТНОЕ
ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
РОСТОВСКОЙ ОБЛАСТИ
«РОСТОВСКИЙ-НА-ДОНУ КОЛЛЕДЖ СВЯЗИ И ИНФОРМАТИКИ»

ПРАКТИЧЕСКАЯ РАБОТА №21

по дисциплине «Разработка программных модулей»
на тему: «Рефакторинг кода в Java-проектах с GUI и событийной моделью»

Выполнил: студент
группы ИС-33 Саруев П.А.
Руководитель: преподаватель,
Трифанкова А.А.

Ростов-на-Дону
2025 г.

Отчёт:

1. **GameModel.java - Разделение большого метода updateGame()**

```
public void updateGame() {  1 usage
    if (currentState != GameState.RUNNING) return;

    bullets.forEach(Bullet::update);
    explosions.forEach(Explosion::update);

    if(playerTank.isAlive() && playerTank.isMovingForward()) {
        processTankMovement(playerTank,  step: 5);
    }

    for (Tank enemy : enemies) {
        enemy.update();
        if (enemy.isAlive() && enemy.isMovingForward()) {
            processTankMovement(enemy,  step: 3);
        }
        if (enemy.isAlive() && enemy.canShoot() && Math.random() < 0.05) {
            if (isPlayerInLineOfSight(enemy, playerTank,  range: 500)) {
                Bullet newBullet = enemy.shoot();
                if (newBullet != null) bullets.add(newBullet);
            }
        }
    }

    checkCollisions();

    enemies.removeIf( Tank tank -> !tank.isAlive());
    bullets.removeIf( Bullet bullet -> !bullet.isAlive());
    explosions.removeIf( Explosion exp -> !exp.isAlive());

    if (score >= initialEnemyCount && initialEnemyCount > 0) {
        currentState = GameState.VICTORY;
    }
    if (playerTank != null && !playerTank.isAlive()) {
        currentState = GameState.DEFEAT;
    }
```

Рис. 1 (GameModel.java – до)

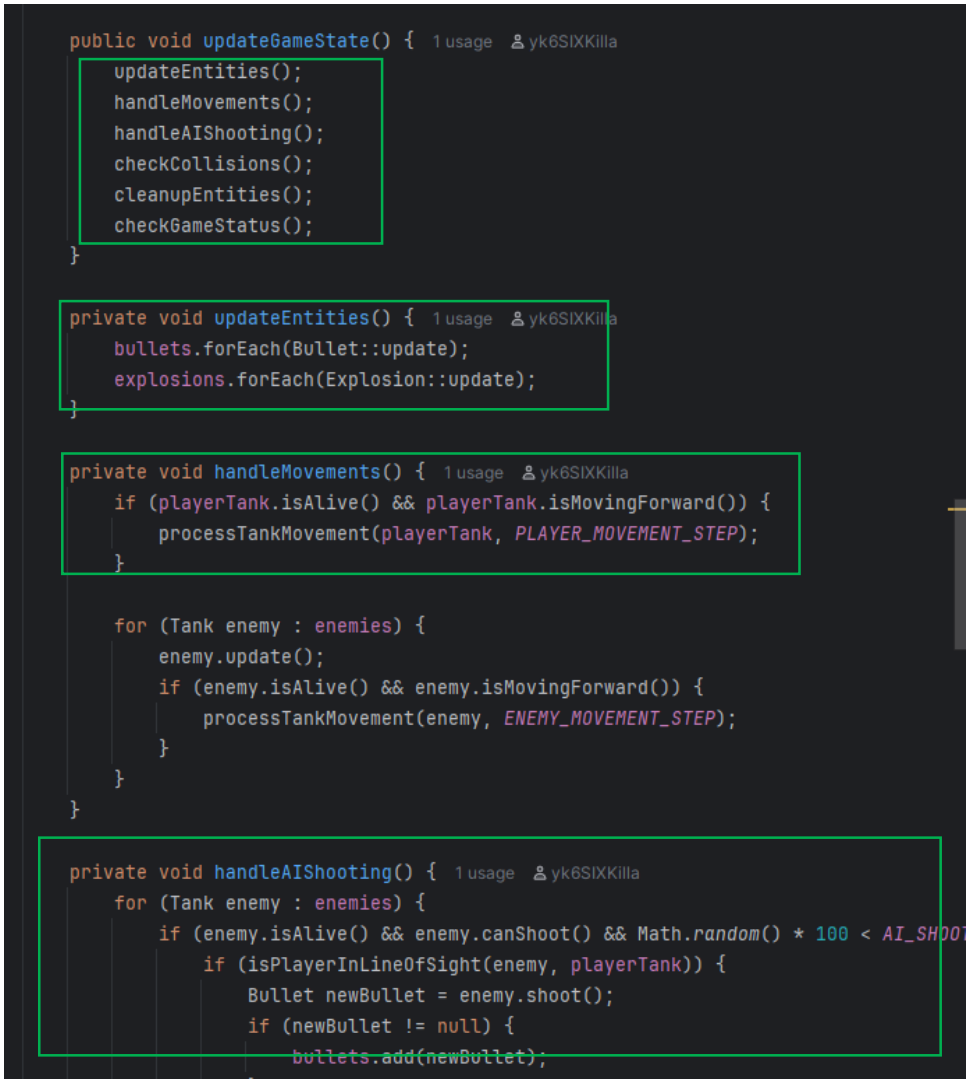

Рис. 2 (GameModel.java – после)

Сделал: Разделил один большой метод updateGame() из 40+ строк на несколько логически связанных методов: updateEntities(), handleMovements(), handleAIShooting(), checkCollisions(), cleanupEntities(), checkGameStatus().

2. **Выделение бизнес-логики из обработчиков событий - Создание класса GameLogicProcessor в GameModel.java**

```java
public void updateGame() {  1 usage
    if (currentState != GameState.RUNNING) return;

    bullets.forEach(Bullet::update);
    explosions.forEach(Explosion::update);

    if(playerTank.isAlive() && playerTank.isMovingForward()) {
        processTankMovement(playerTank,  step: 5);
    }

    for (Tank enemy : enemies) {
        enemy.update();
        if (enemy.isAlive() && enemy.isMovingForward()) {
            processTankMovement(enemy,  step: 3);
        }
        if (enemy.isAlive() && enemy.canShoot() && Math.random() < 0.05) {
            if (isPlayerInLineOfSight(enemy, playerTank,  range: 500)) {
                Bullet newBullet = enemy.shoot();
                if (newBullet != null) bullets.add(newBullet);
            }
        }
    }
```

Рис. 3 (GameModel.java – До. Старая Логика updateGame)

```java
    private void checkCollisions() {  1 usage
        List<Bullet> currentBullets = new ArrayList<>(bullets);
        for (Bullet bullet : currentBullets) {
            if (!bullet.isAlive()) continue;

            Rectangle bRect = bullet.getBounds();

            for (Rectangle wall : obstacles) {
                if (bRect.intersects(wall)) {
                    bullet.setAlive(false);
                    explosions.add(new Explosion( x: bullet.getX() - 15,  y: bullet.getY() - 15));
                    break;
                }
            }
            if (!bullet.isAlive()) continue;

            if (bullet.isPlayerBullet()) {
                for (Tank enemy : enemies) {
                    if (enemy.isAlive() && enemy.getBounds().intersects(bRect)) {
                        enemy.takeDamage();
                        bullet.setAlive(false);
                        if (!enemy.isAlive()) {
                            score++;
                            explosions.add(new Explosion(enemy.getX(), enemy.getY()));
                        }
                        break;
                    }
                }
            } else {
                if (playerTank != null && playerTank.isAlive() && playerTank.getBounds().intersects(bRect)) {
                    playerTank.takeDamage();
                    bullet.setAlive(false);
                    explosions.add(new Explosion(playerTank.getX(), playerTank.getY()));
                }
            }
```

Рис. 4 (GameModel.java – До. Старая Логика checkCollisions)

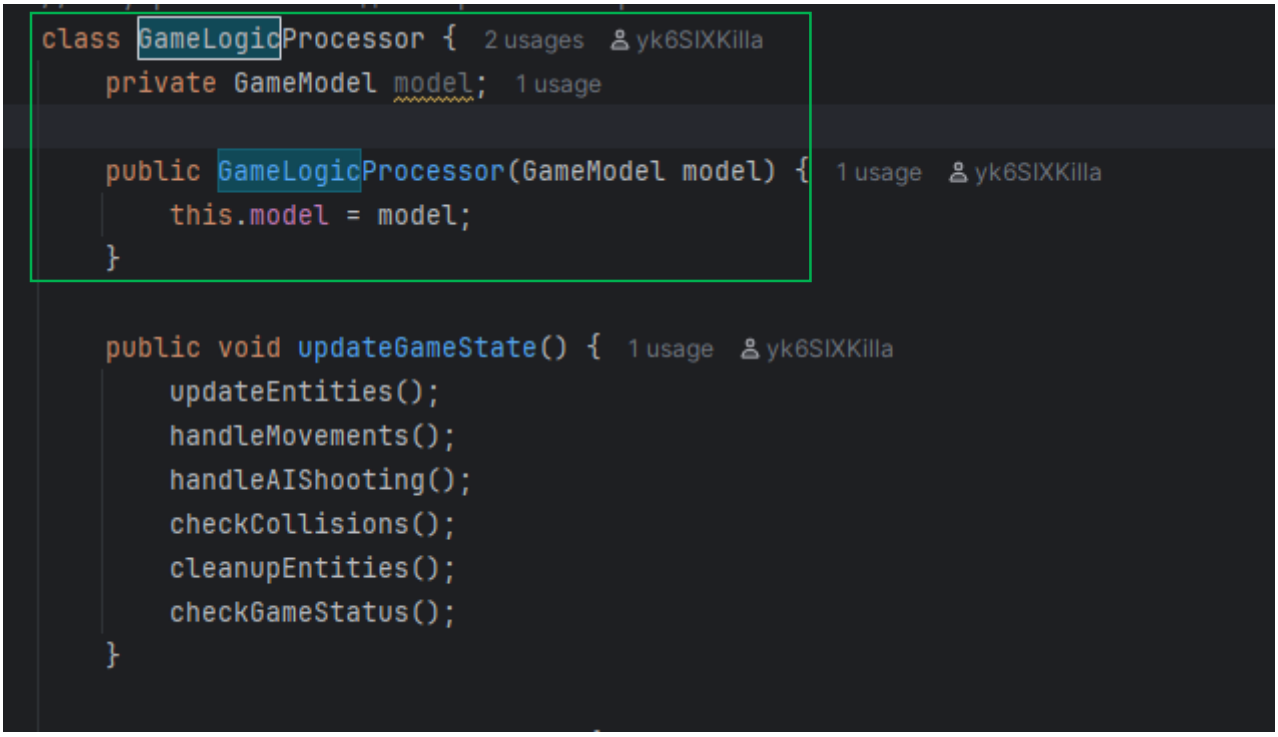

Рис. 5 (GameModel.java – После. Создание Внутреннего класа GameLogicProcessor)

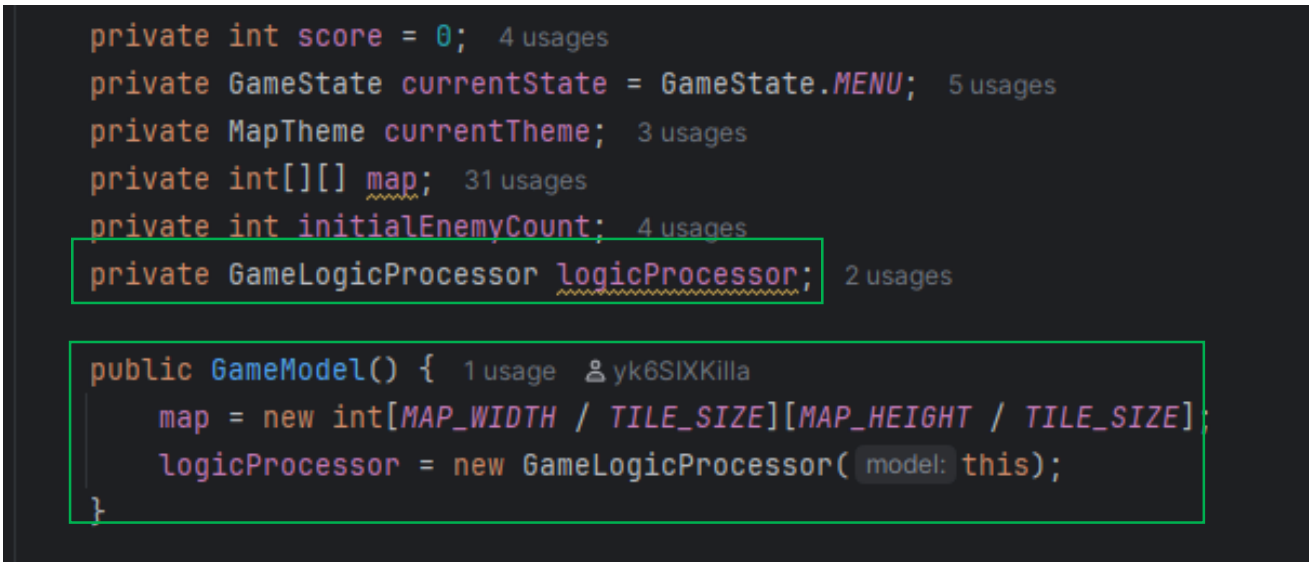

Рис. 6 (GameModel.java – После.)

Сделал: Создал внутренний класс GameLogicProcessor, который теперь содержит всю бизнес-логику игры. Класс GameModel теперь отвечает только за хранение состояния, а логика обработки вынесена в отдельный компонент.

3. Переименование переменных и методов по Java Code Conventions

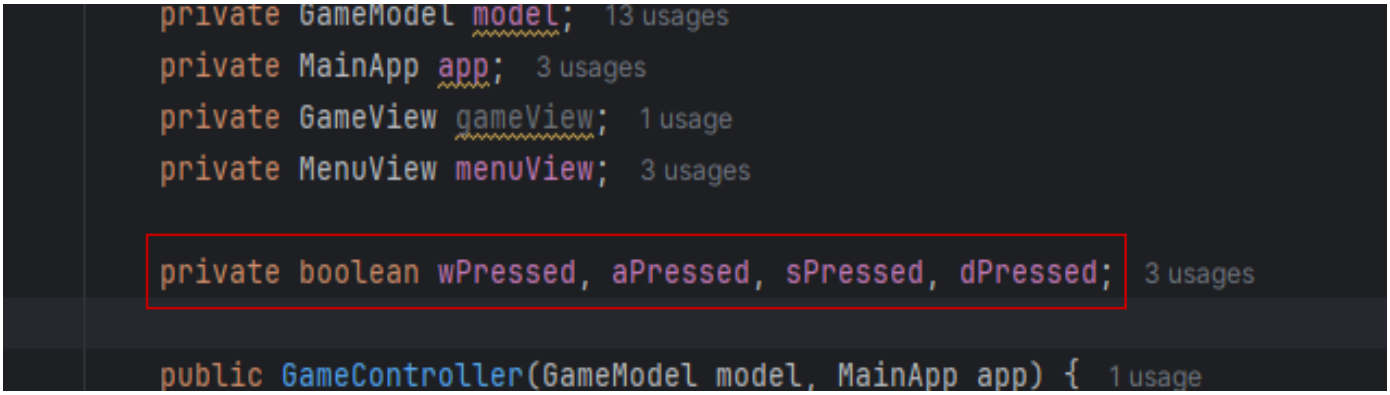

Рис. 7 (GameController.java – До.)

```
public Bullet shoot() {  2 usages
    if (!canShoot()) return null;
    lastShotTime = System.currentTimeMillis();

    int bX = x + SIZE / 2 - Bullet.SIZE / 2;
    int bY = y + SIZE / 2 - Bullet.SIZE / 2;
    int offset = SIZE / 2 + 15;

    if (direction == 0) bY -= offset;
    else if (direction == 1) bX += offset;
    else if (direction == 2) bY += offset;
    else if (direction == 3) bX -= offset;

    return new Bullet(bX, bY, direction, isPlayer);
}
```

Рис. 8 (Tank.java – До.)

```
private GameModel model;  7 usages
private MainApp application;  3 usages
private GameView gameView;  1 usage
private MenuView menuView;  3 usages

private boolean isWPressed, isAPressed, isSPressed, isDPressed;  3 usages
```

Рис. 8

Рис. 9 (GameController.java – После.)

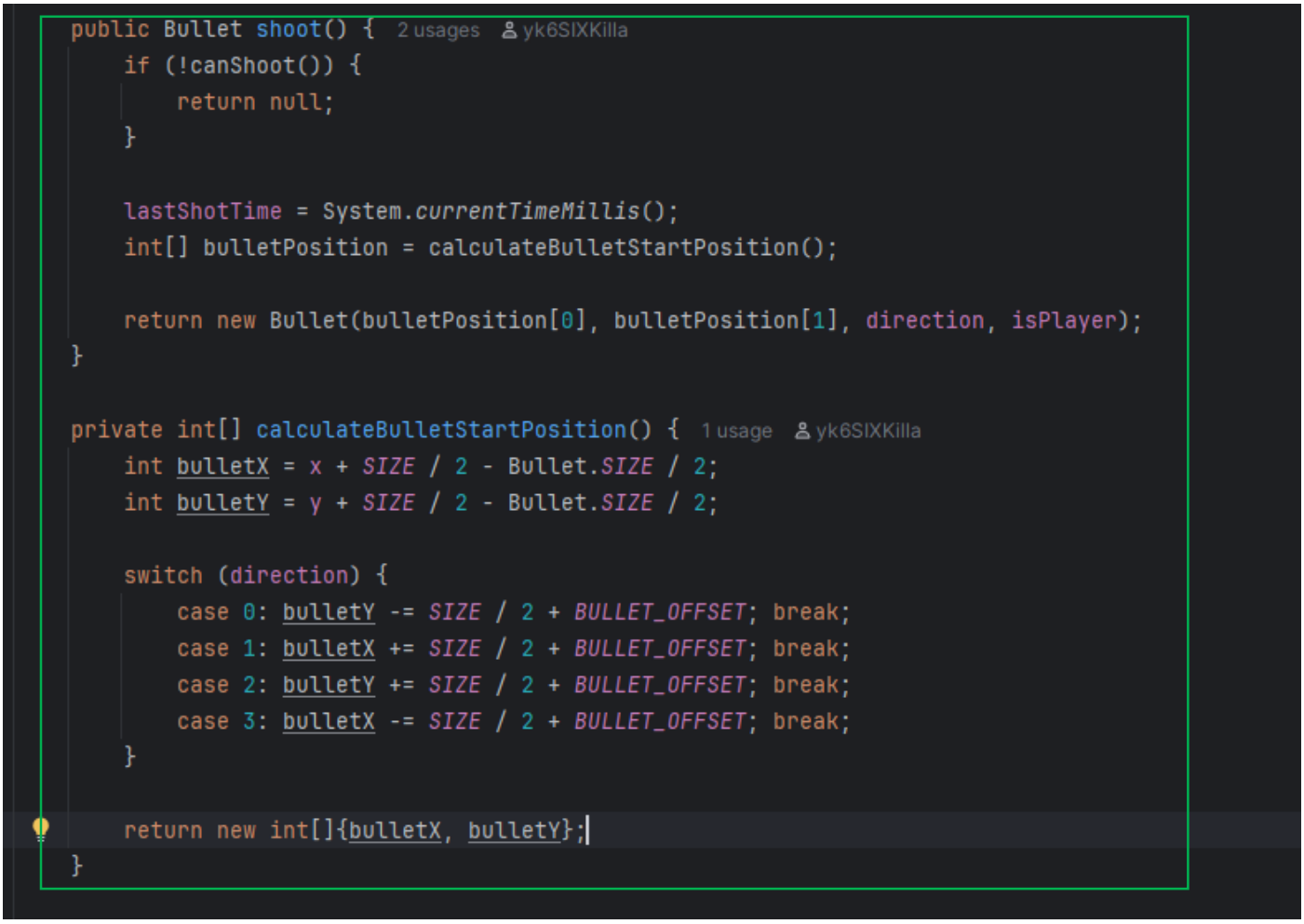

Рис. 10 (Tank.java – После.)

Сделал: Переименовал сущности по принципам Java Code Conventions:

1. wPressed → isWPressed (логические переменные с префиксом is)
2. Бессмысленный код расчёта позиции → calculateBulletStartPosition()
3. Методы с короткими именами → полные описательные имена

4. Устранение дублирования кода

```
private void processTankMovement(Tank tank, int step) {  2 usages
    Rectangle future = tank.getFutureBounds(step);

    if (future.x < 0 || future.x + future.width > MAP_WIDTH ||
            future.y < 0 || future.y + future.height > MAP_HEIGHT) {
        if (!tank.isPlayer()) tank.setDirection((int)(Math.random()*4));
        return;
    }

    boolean collision = false;
    for (Rectangle wall : obstacles) {
        if (future.intersects(wall)) {
            collision = true;
            break;
        }
    }

    if (!collision) {
        tank.setPosition(future.x, future.y);
    } else {
        if (!tank.isPlayer()) {
            tank.setDirection((int) (Math.random() * 4));
        }
    }
}
```

Рис. 11 (GameModel.java – до.)

```java
    private void processTankMovement(Tank tank, int step) {  2 usages  yk6SIXKilla
        Rectangle futureBounds = tank.getFutureBounds(step);

        if (isOutOfBounds(futureBounds)) {
            if (!tank.isPlayer()) {
                tank.setDirection((int) (Math.random() * 4));
            }
            return;
        }

        if (hasWallCollision(futureBounds)) {
            if (!tank.isPlayer()) {
                tank.setDirection((int) (Math.random() * 4));
            }
            return;
        }

        tank.setPosition(futureBounds.x, futureBounds.y);
    }

    private boolean isOutOfBounds(Rectangle bounds) {  1 usage  yk6SIXKilla
        return bounds.x < 0 || bounds.x + bounds.width > MAP_WIDTH ||
                bounds.y < 0 || bounds.y + bounds.height > MAP_HEIGHT;
    }

    private boolean hasWallCollision(Rectangle bounds) {  1 usage  yk6SIXKilla
        for (Rectangle wall : obstacles) {
            if (bounds.intersects(wall)) {
                return true;
            }
        }
        return false;
    }

    private boolean isPlayerInLineOfSight(Tank shooter, Tank target) {  1 usage  yk6SIXKilla
        int dx = Math.abs(shooter.getX() - target.getX());
        int dy = Math.abs(shooter.getY() - target.getY());
        return (dx < 50 && dy < LINE_OF_SIGHT_RANGE) ||
                (dy < 50 && dx < LINE_OF_SIGHT_RANGE);
    }
}
```

Рис. 12 (GameModel.java – После.)

```java
private void checkCollisions() {  1 usage  yk6SIXKilla
    List<Bullet> currentBullets = new ArrayList<>(bullets);

    for (Bullet bullet : currentBullets) {
        if (!bullet.isAlive()) {
            continue;
        }

        checkWallCollision(bullet);

        if (!bullet.isAlive()) {
            continue;
        }

        if (bullet.isPlayerBullet()) {
            checkPlayerBulletCollision(bullet);
        } else {
            checkEnemyBulletCollision(bullet);
        }
    }
}

private void checkWallCollision(Bullet bullet) {  1 usage  yk6SIXKilla
    Rectangle bulletRect = bullet.getBounds();

    for (Rectangle wall : obstacles) {
        if (bulletRect.intersects(wall)) {
            bullet.setAlive(false);
            explosions.add(new Explosion( x: bullet.getX() - 15,  y: bullet.getY() - 15));
            return;
        }
    }
}
```

Рис. 13 (GameModel.java – После.)

Сделал: Устранил дублирование кода:

1. Вынес общую логику проверки границ в методы isOutOfBounds() и hasWallCollision()
2. Создал общий метод checkWallCollision() для проверки столкновения пуль со стенами
3. Разделил специфичную логику для пуль игрока и врагов в отдельные методы

5. Введение констант вместо «магических чисел»

```
private int health;  4 usages
private Color color;  2 usages
private boolean isPlayer;  6 usages
private long lastShotTime = 0;  2 usages
private final long SHOOT_COOLDOWN = 600;  1 usage
```

```
public boolean canShoot() { return System.currentTimeMillis() - lastShotTime > SHOOT_COOLDOWN; }
```

Рис. 14-15 (Tank.java – До. Дублирование)

```
}
if (enemy.isAlive() && enemy.canShoot() && Math.random() < 0.05) {
    if (isPlayerInLineOfSight(enemy, playerTank, range: 500)) {
        Bullet newBullet = enemy.shoot();
        if (newBullet != null) bullets.add(newBullet);
    }
```

```
private boolean isPlayerInLineOfSight(Tank shooter, Tank target, int range) {  1 usa
    int dx = Math.abs(shooter.getX() - target.getX());
    int dy = Math.abs(shooter.getY() - target.getY());
    return (dx < 50 && dy < range) || (dy < 50 && dx < range);
}

private void checkCollisions() {  1 usage
```

```
private int[] findSafePosition() {  2 usages
    int maxAttempts = 200;
    for (int i = 0; i < maxAttempts; i++) {
        int x = 50 + (int) (Math.random() * (MAP_WIDTH - 100));
        int y = 50 + (int) (Math.random() * (MAP_HEIGHT - 100));
```

Рис. 16-17-18 (GameModel.java – До. Вероятность, проверка видимости, макс кол-во попыток.)

```
private static final long SHOOT_COOLDOWN_MS = 600;  1 usage
private static final int AI_DIRECTION_CHANGE_PROBABILITY = 2;  1 usage
private static final int BULLET_OFFSET = 15;  4 usages
```

```
private static final int AI_SHOOT_PROBABILITY = 5;  1 usage
private static final int LINE_OF_SIGHT_RANGE = 500;  2 usages
private static final int PLAYER_MOVEMENT_STEP = 5;  1 usage
private static final int ENEMY_MOVEMENT_STEP = 3;  1 usage
private static final int MAX_POSITION_ATTEMPTS = 200;  1 usage
private static final int MIN_SAFE_DISTANCE = 50;  4 usages
```

Рис. 19-20 (GameModel.java – После. Вероятность, проверка видимости, макс кол-во попыток.)

Сделал: Заменил «магические числа» на именованные константы с поясняющими названиями. Это улучшило читаемость кода и упростило его изменение.

6. Дополнительно улучшил визуализацию танка

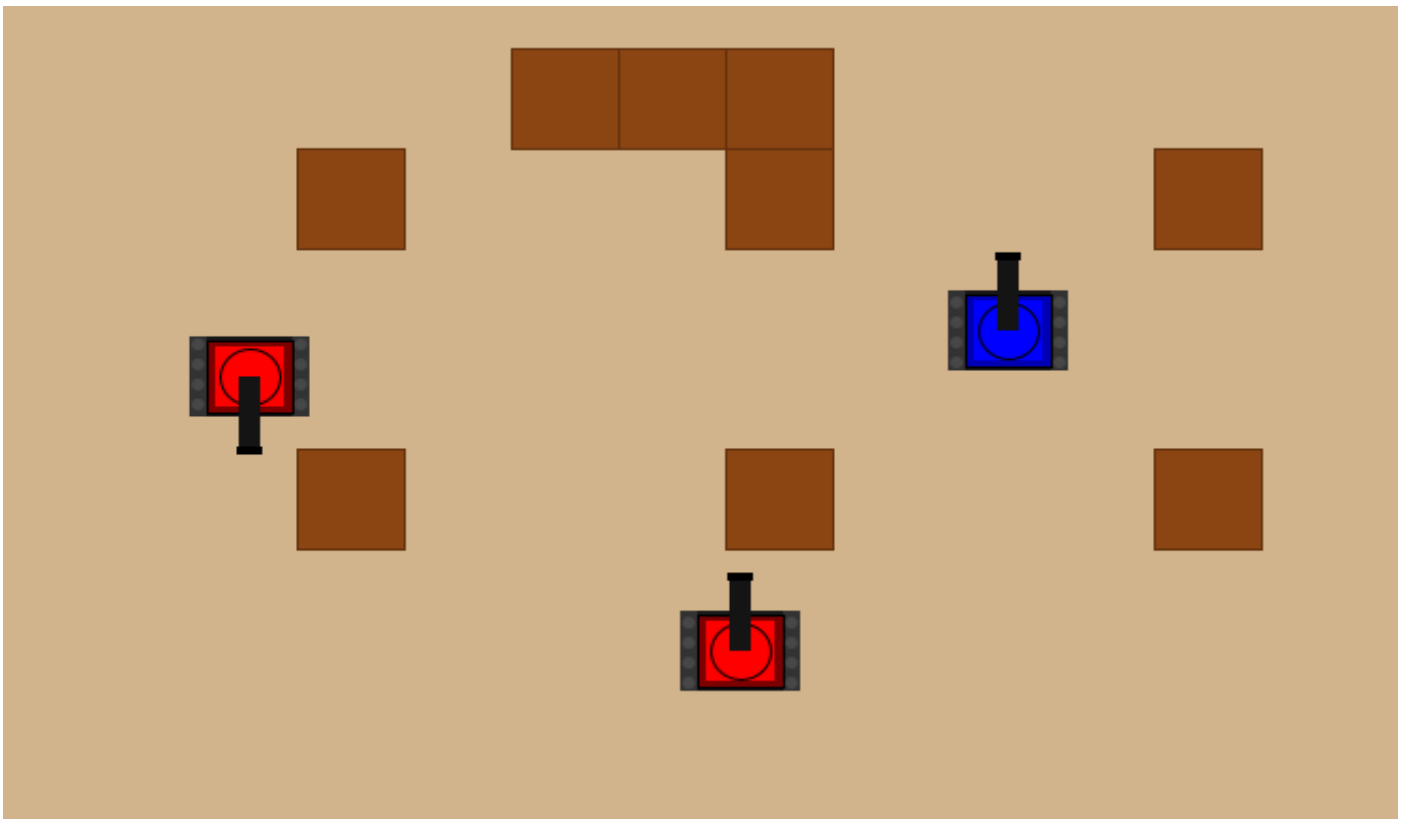

Рис. 21 (Танки - До)

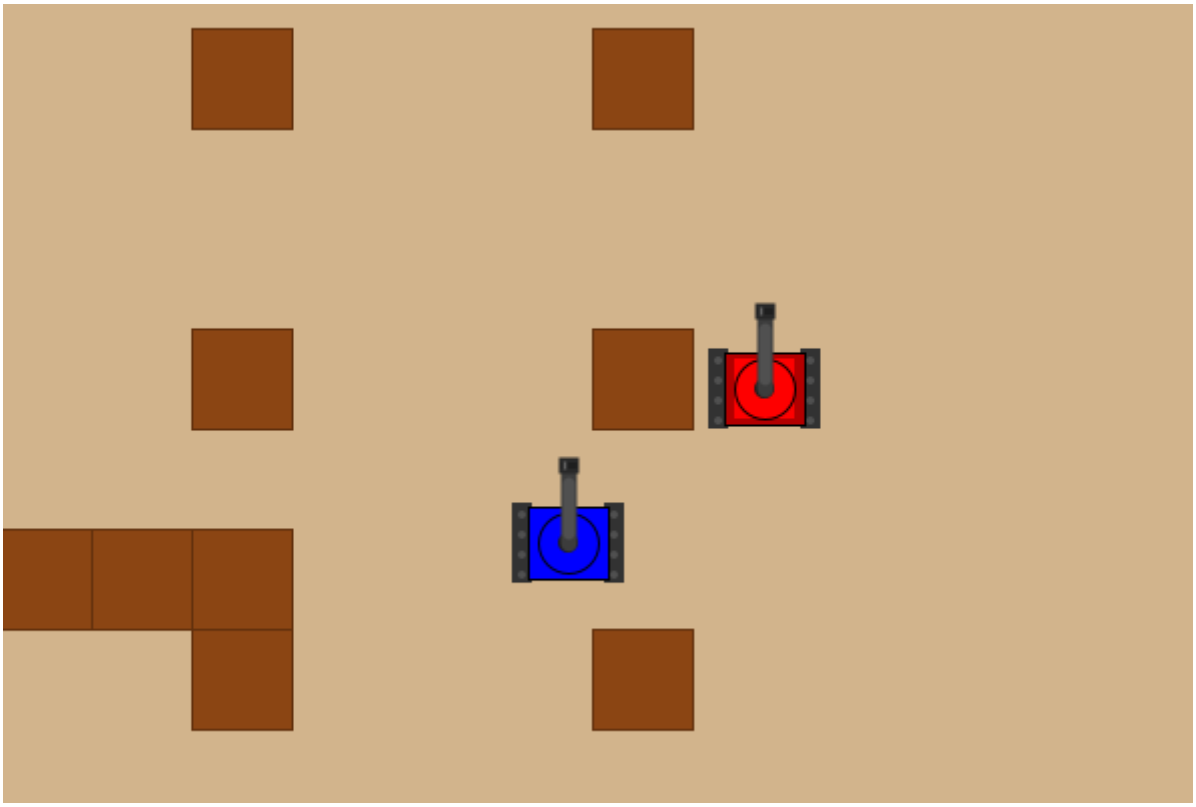

Рис. 22 (Танки - После)

Итоговое сравнение:

1. Читаемость кода значительно улучшилась за счёт разделения больших методов и введения осмысленных названий
2. Поддерживаемость повысилась благодаря устранению дублирования и введению констант
3. Архитектура стала чище с выделением бизнес-логики в отдельный класс GameLogicProcessor
4. Визуальное качество (на мой взгляд) улучшилось с переработкой отрисовки танка
5. Соответствие стандартам достигнуто за счёт применения Java Code Conventions
6. Код стал более структурированным, понятным и легким для дальнейшего развития и отладки.