Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Волгоградский государственный технический университет»

Факультет электроники и вычислительной техники
Кафедра «Программное обеспечение автоматизированных систем»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе

по дисциплине «Объектно-ориентированный анализ и программирование»
на тему: «Проектирование программы с использованием объектно-ориентированного подхода»
(индивидуальное задание – вариант №25)

Студент: Серпинин Е.И.
Группа: ПрИн-366

Работа зачтена с оценкой _____ «__» __июня_ 2022 г.

Руководитель проекта, нормоконтроллер _____ Литовкин Д.В.

Волгоград 2022 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Волгоградский государственный технический университет»

Факультет электроники и вычислительной техники
Направление 09.03.04 «Программная инженерия»
Кафедра «Программное обеспечение автоматизированных систем»

Дисциплина «Объектно-ориентированный анализ и программирование»

Утверждаю
Зав. кафедрой _____ Орлова Ю.А.

ЗАДАНИЕ
на курсовую работу

Студент: Серпинин Е.И.
Группа: ПрИн-366

1. Тема: «Проектирование-программы с использованием объектно-ориентированного подхода» (индивидуальное задание – вариант №25)
Утверждена приказом от «01» марта 2022г. № 303-ст

2. Срок представления работы к защите « 26 » мая 2022 г.

3. Содержание пояснительной записки:
формулировка задания, требования к программе, структура программы, типовые процессы в программе, человеко-машинное взаимодействие, код программы и модульных тестов

4. Перечень графического материала:
_____

5. Дата выдачи задания «4» марта 2022 г.

Руководитель проекта: _____ Литовкин Д.В.

Задание принял к исполнению: _____ Серпинин Е.И.

«4» марта 2022 г.

Содержание

1 Формулировка задания

Правила игры «Следы»:

- Игровое поле состоит из шестиугольников, игрок располагается в одном из них.
- Игрок может перемещаться в любой из соседних шестиугольников, если тот проходим.
- При перемещении он оставляет за собой след (шестиугольник закрашивается определенным цветом).
- Игрок не может повторно наступать на шестиугольники, на которых оставлены следы того же цвета, что и у игрока.
- Цель - игрок должен достичь целевого шестиугольника.
- На поле могут быть разбросаны ключи, которые игрок должен собрать. Посещение целевого шестиугольника до сбора всех ключей не приводит к завершению игры.
- Игрок не оставляет следов на целевом шестиугольнике и может посещать его несколько раз.

Подвариант 1: необходимо предусмотреть в программе точки расширения, используя которые можно реализовать вариативную часть программы (в дополнение к базовой функциональности).

Вариативность: предусмотреть различные критерии окончания игры и их комбинации (через отношения И, ИЛИ, НЕ). Игроку должно сообщаться условие окончания игры.
НЕ изменяя ранее созданные классы, а используя точки расширения, реализовать: следующие критерии окончания игры: - собраны все ключи; - достигнут выход; - длина пройденного пути равна, больше или меньше заданной.

2 Нефункциональные требования

1. Программа должна быть реализована на языке Java SE 12 с использованием стандартных библиотек, в том числе, библиотеки Swing.
2. Форматирование исходного кода программы должно соответствовать Java Code Conventions, September 12, 1997.

3 Первая итерация разработки

3.1 Формулировка упрощенного варианта задания

Правила игры «Следы»:

- Игровое поле состоит из шестиугольников, игрок располагается в одном из них.
- Игрок может перемещаться в любой из соседних шестиугольников, если тот проходим.
- При перемещении он оставляет за собой след (шестиугольник закрашивается определенным цветом).
- Игрок не может повторно наступать на шестиугольники, на которых оставлены следы того же цвета, что и у игрока.
- Цель - игрок должен достичь целевого шестиугольника.
- На поле могут быть разбросаны ключи, которые игрок должен собрать. Посещение целевого шестиугольника до сбора всех ключей не приводит к завершению игры.
- Игрок не оставляет следов на целевом шестиугольнике и может посещать его несколько раз.

3.2 Функциональные требования (сценарии)

Сценарий «Игра»

1. Пользователь инициирует начало игры
2. Игра создает карту из шестиугольных Ячеек и размещает на них ключи, Робота и Выход
3. Игра обрабатывает игровой цикл пока игра не завершена
   3.1. По желанию пользователя Робот делает шаг в одну из сторон шестиугольника
4. Игра завершается

Сценарий «Игра не завершена»

1. Игра проверяет доходимость Робота до ключей и выхода. Если пути нет, то она считает, что игрок проиграл
2. Игра проверяет находиться ли Робот на выходе, и остались ли на карте ключи. Если Робот на выходе и на карте нет ключей, то Игра считает, что игрок победил

Сценарий «Робот делает шаг в Ячейку без своего следа»

1. По указанию робота ячейка возвращает след
2. Если след не совпадает со следом робота
    a. Робот меняет свою координату на Поле, соответствующей пустой Ячейке
    b. Робот оставляет свой след на Ячейке, на которую наступил

Сценарий «Робот делает шаг в Ячейку с ключем»

1. Робот подбирает Ключ и убирает его с текущей Ячейки

Сценарий «Робот делает шаг в ячейку Выхода»

1. Робот меняет свою координату на Поле, соответствующей ячейке Выхода

3.3 Словарь предметной области

Игра – способна создавать карту из шестиугольников, размещать на ней ключи, Робота и Выход, так же проверяет условия победы игрока

Поле - прямоугольная область, состоящая из шестиугольных ячеек. Знает о роботе и ключах

Ячейка - шестиугольная область поля. Одновременно может содержать в себе только робота, или ключ, или быть пустой. Ячейки могут содержать на себе след

Выход - разновидность ячейки. На поле может быть только одна ячейка, являющаяся выходом. На ней не может быть оставлен след

Робот – умеет менять свою координату на поле, подбирать ключи, оставлять след на ячейке. Определяет по следу ячейки может ли он на неё наступить. Им управляет игрок.

Ключ – объект в ячейке. Нужен для победы игрока

След – цвет ячейки

3.4 Структура программы на уровне классов

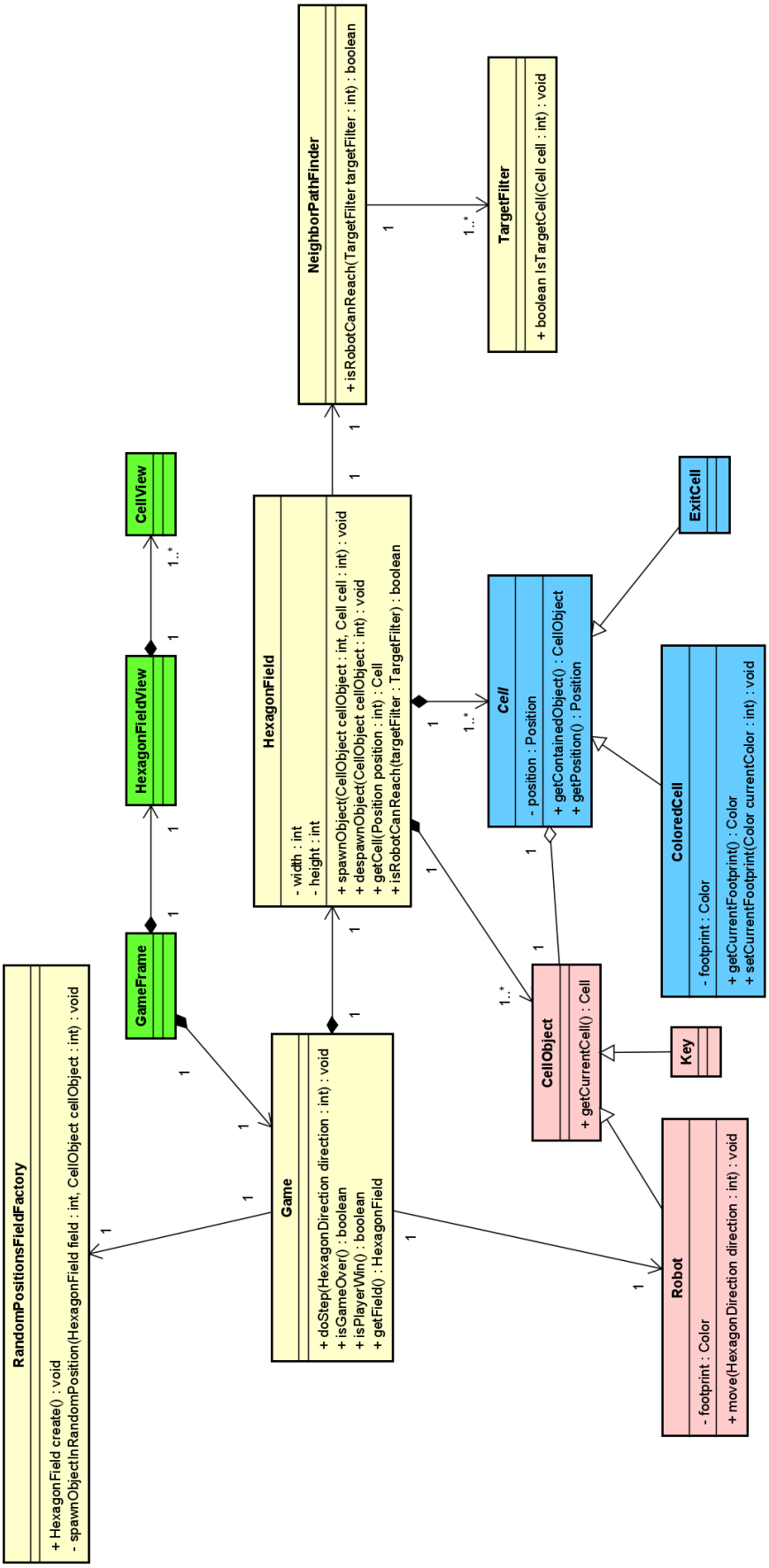


Диаграмма классов вычислительной модели

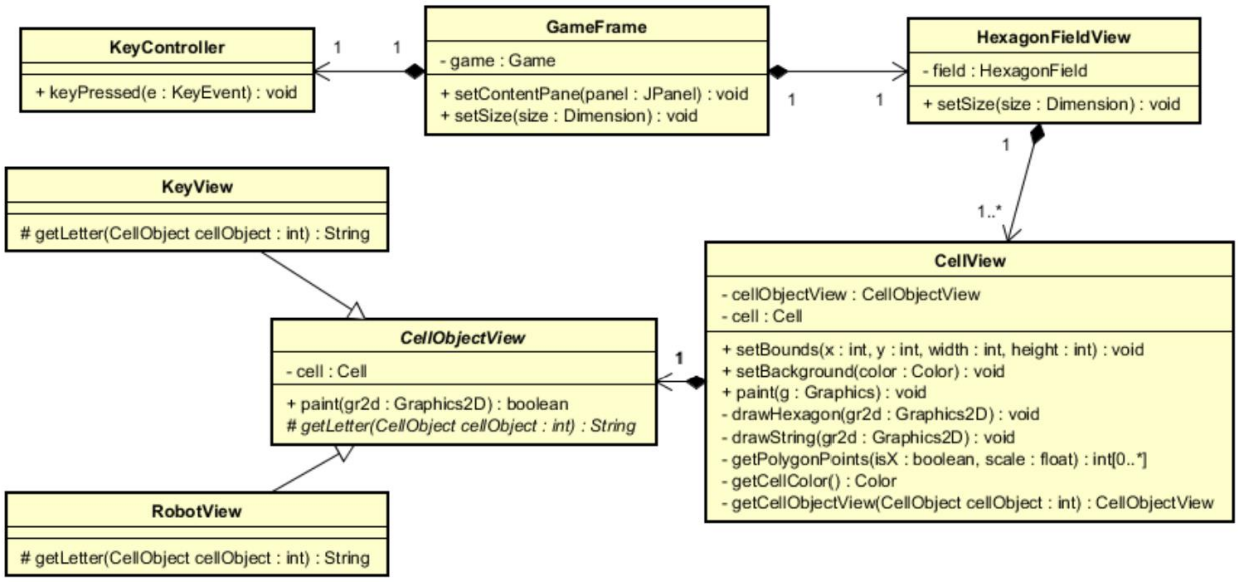
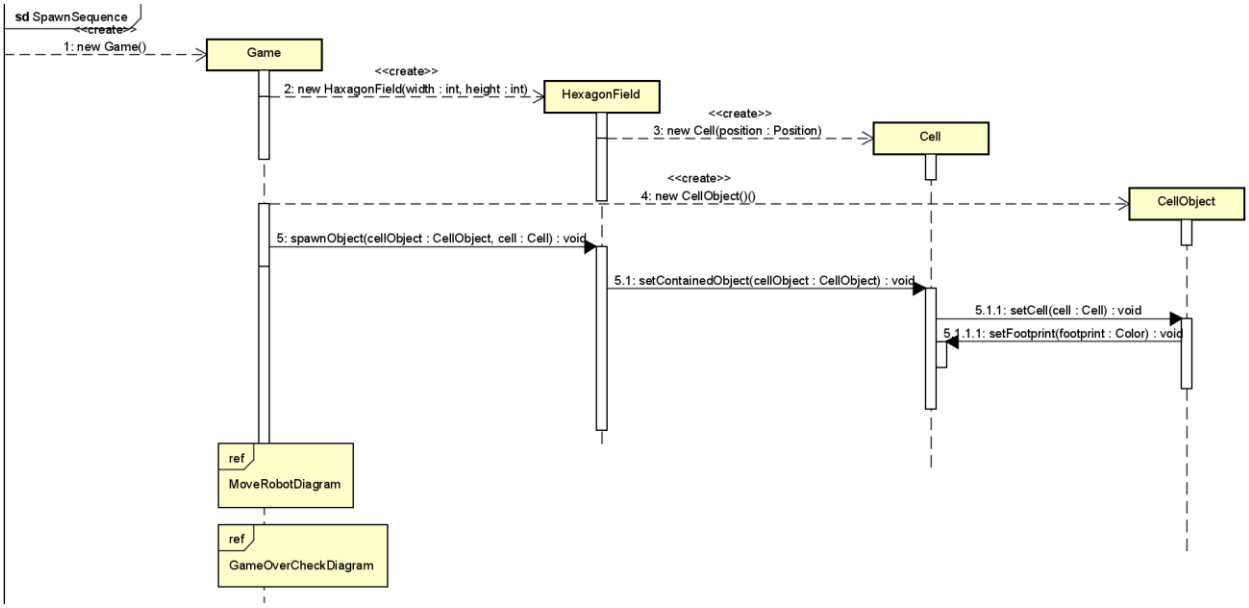
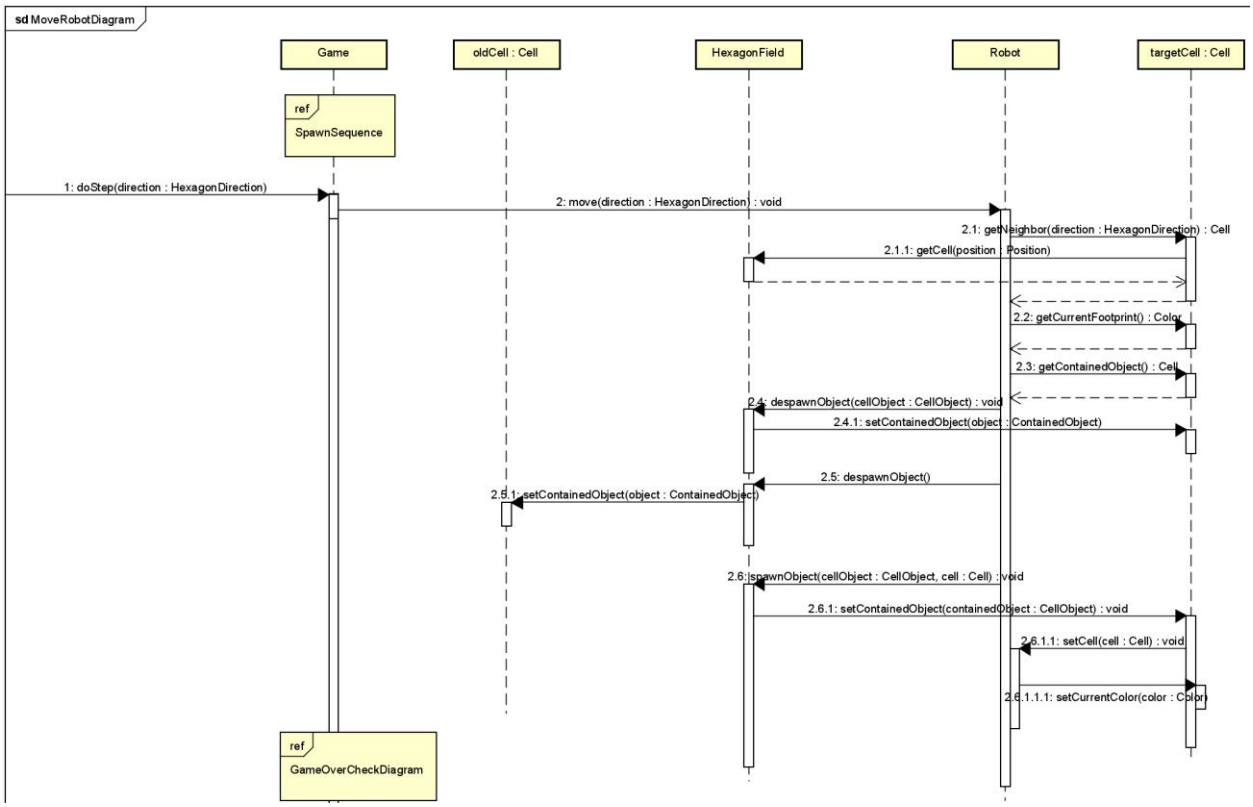


Диаграмма классов представления

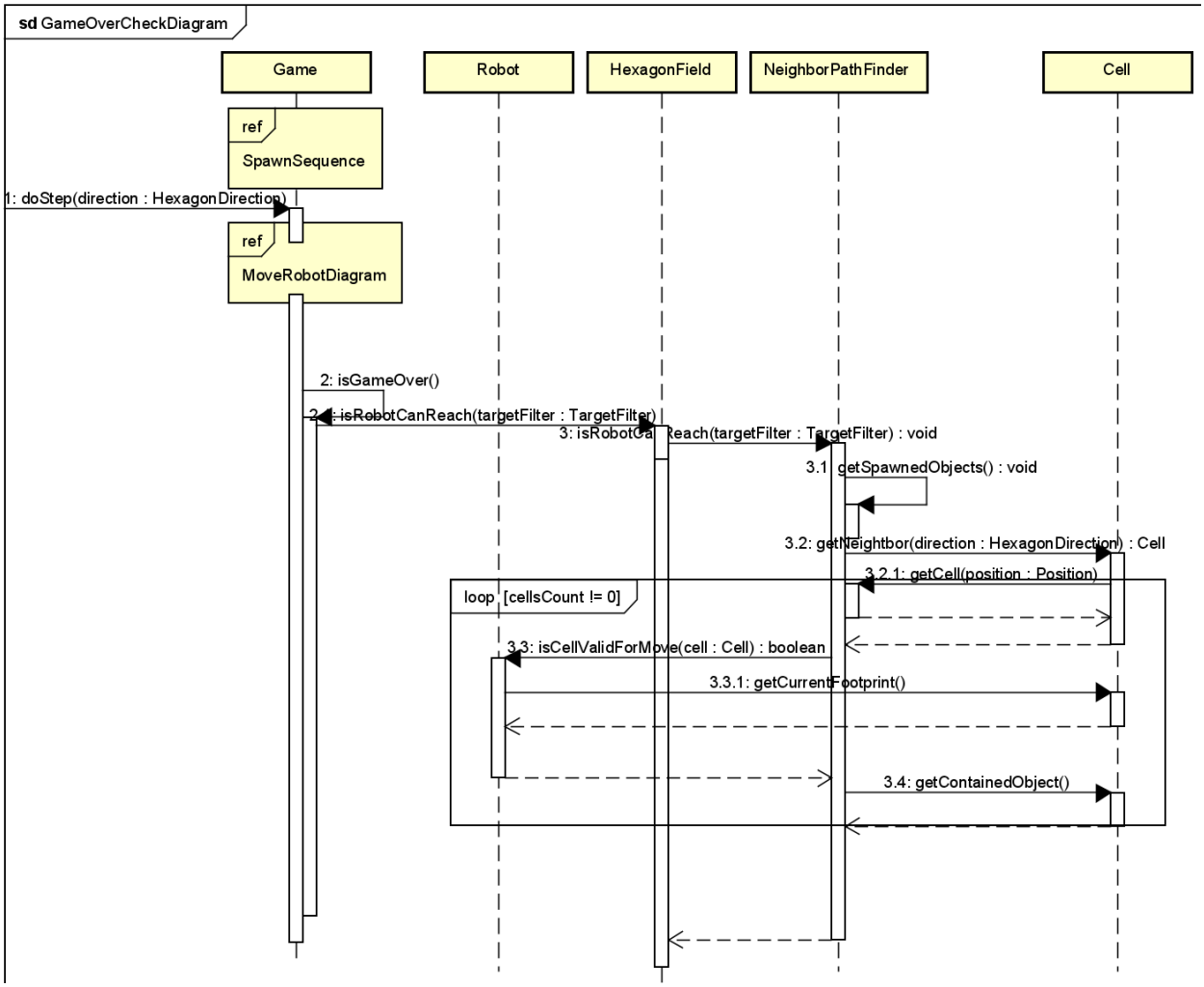
3.5 Типовые процессы в программе



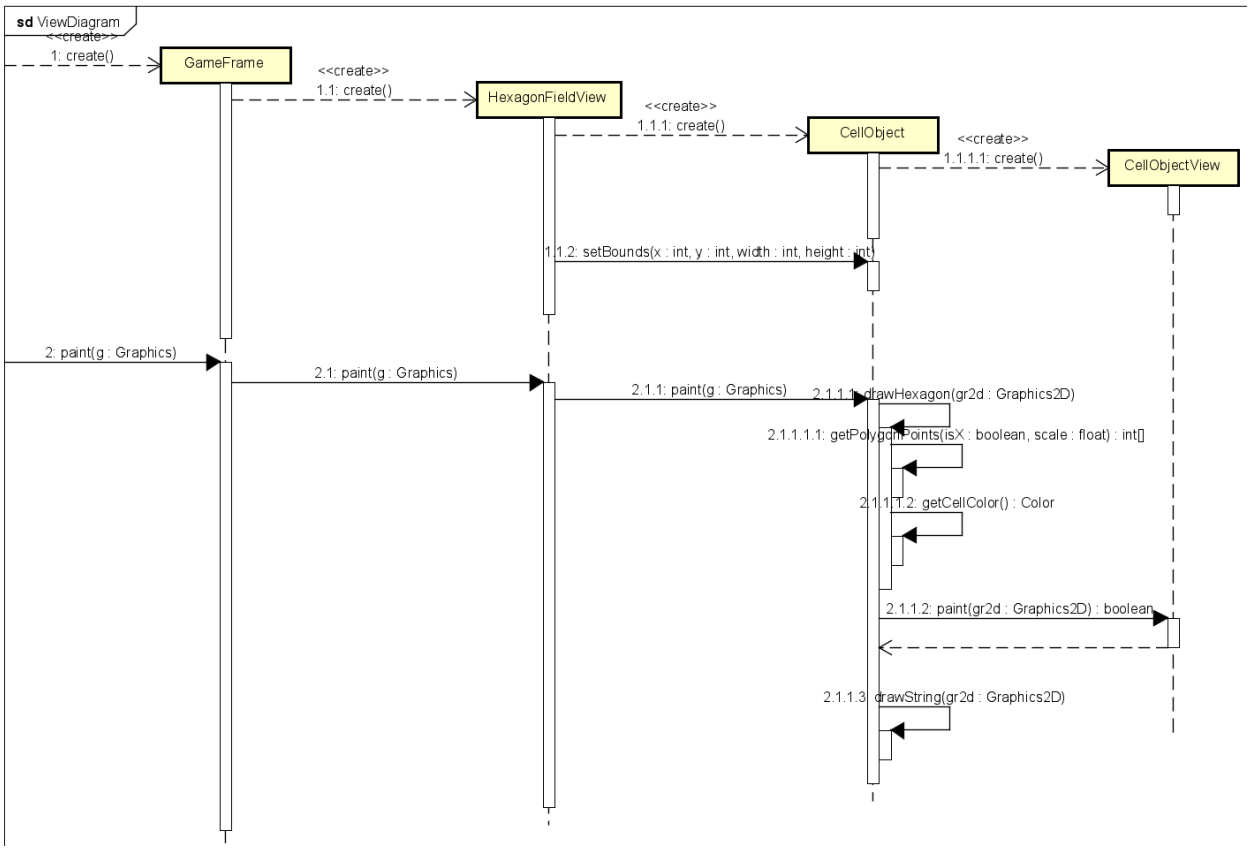
Создание игровой карты, расстановка объектов и игрока



Ход робота в клетку без следа с ключом



Проверка условий окончания игры



Создание и отрисовка представления

3.6 Реализация ключевых классов

```
public abstract class Cell {

    private HexagonField field;
    private Position position;
    private CellObject containedObject;

    public Cell(HexagonField field, Position position){
        this.field = field;
        this.position = position;
    }

    public Cell getNeighbor(HexagonDirection direction){
        return field.getCell(direction.toPosition(position.y % 2 ==
1).add(position));
    }

    public void setContainedObject(CellObject mutualCellObject) {
        if (this.containedObject != null && mutualCellObject == null) {
            CellObject old = this.containedObject;
            this.containedObject = null;
            old.setCell(null);
        }
        else if (this.containedObject == null && mutualCellObject != null) {
            this.containedObject = mutualCellObject;
            this.containedObject.setCell(this);
        }
    }

    public CellObject getContainedObject() { return containedObject; }
}


public class ColoredCell extends Cell {

    public static final Color defaultFootprint = Color.white;
    private Color currentFootprint = defaultFootprint;

    public ColoredCell(HexagonField field, Position currentPosition) {
        super(field, currentPosition);
    }

    public Color getCurrentFootprint() {
        return currentFootprint;
    }

    public void setCurrentFootprint(Color currentFootprint) {
        this.currentFootprint = currentFootprint;
    }
}


public class ExitCell extends Cell {

    public ExitCell(HexagonField field, Position currentPosition) {
        super(field, currentPosition);
    }
}
```

```
public class CellObject {

    private Cell currentCell;

    public void setCell(Cell mutualCell) {
        if (this.currentCell != null && mutualCell == null) {
            Cell old = this.currentCell;
            this.currentCell = null;
            old.setContainedObject(null);
        }
        else if (this.currentCell == null && mutualCell != null) {
            this.currentCell = mutualCell;
            this.currentCell.setContainedObject(this);
        }
    }

    public Cell getCell() { return currentCell; }

}


public class Robot extends CellObject {

    private final HexagonField field;
    private final Color footprintColor;

    public Robot(HexagonField field, Color footprintColor){
        this.field = field;
        this.footprintColor = footprintColor;
    }

    public void move(HexagonDirection direction){
        Cell targetCell = getCell().getNeighbor(direction);

        if (isCellValidForMove(targetCell)){

            CellObject objectInCell = targetCell.getContainedObject();
            if (objectInCell != null){
                field.despawnObject(objectInCell);
            }

            field.despawnObject(this);
            field.spawnObject(this, targetCell);
        }
    }

    public boolean isCellValidForMove(Cell cell){
        return cell != null
                && (!(cell instanceof ColoredCell)
                    ||
!footprintColor.equals(((ColoredCell)cell).getCurrentFootprint()));
    }

    @Override
    public void setCell(Cell cell) {
        super.setCell(cell);

        if (cell instanceof ColoredCell){
            ((ColoredCell)cell).setCurrentFootprint(footprintColor);
        }
    }

    public Color getFootprintColor() {
        return footprintColor;
```

```
    }
}


public class RandomPositionsFieldFactory implements FieldFactory {

    private final int fieldWidth;
    private final int fieldHeight;
    private final int maxKeysCount;

    public RandomPositionsFieldFactory(int fieldWidth, int fieldHeight, int
maxKeysCount){
        this.fieldWidth = fieldWidth;
        this.fieldHeight = fieldHeight;
        this.maxKeysCount = maxKeysCount;
    }

    @Override
    public HexagonField create() {
        HexagonField field = new HexagonField(fieldWidth, fieldHeight,
Utils.getRandomPoint(fieldWidth, fieldHeight));
        int keysCount = Utils.rnd.nextInt(maxKeysCount) + 1;

        for (int i = 0; i < keysCount; i++){
            spawnObjectInRandomPosition(field, new Key());
        }

        spawnObjectInRandomPosition(field, new Robot(field, Color.ORANGE));
        return field;
    }

    private void spawnObjectInRandomPosition(HexagonField field, CellObject
cellObject){
        do {
            try {
                Position position = Utils.getRandomPoint(field.getWidth(),
field.getHeight());
                field.spawnObject(cellObject, field.getCell(position));
                return;
            }
            catch (InvalidParameterException ignored){}
        }
        while (true);
    }
}


public class NeighborPathFinder implements PathFinder {

    private final HexagonField field;

    public NeighborPathFinder(HexagonField field) {
        this.field = field;
    }

    @Override
    public boolean isRobotCanReach(TargetFilter targetFilter) {
        Robot robot = (Robot) field.getSpawnedObjects().stream().filter(obj -
> obj instanceof Robot).findFirst().orElse(null);

        List<Cell> checkedCells = new ArrayList<>();
        List<Cell> toCheckCells = new ArrayList<>();
        toCheckCells.add(robot.getCell());
```

```
        while (!toCheckCells.isEmpty()){
            Cell currentCell = toCheckCells.get(0);

            for (HexagonDirection direction : HexagonDirection.values()){
                Cell nextCell = currentCell.getNeighbor(direction);

                if (nextCell != null && robot.isCellValidForMove(nextCell) &&
!checkedCells.contains(nextCell)){
                    toCheckCells.add(nextCell);
                    checkedCells.add(nextCell);
                }
            }

            if (targetFilter.IsTargetCell(currentCell))
            {
                return true;
            }

            toCheckCells.remove(0);
        }

        return false;
    }
}


public class KeyTargetFilter implements TargetFilter {

    @Override
    public boolean IsTargetCell(Cell cell) {
        return cell.getContainedObject() instanceof Key;
    }
}


public class ExitTargetFilter implements TargetFilter {

    @Override
    public boolean IsTargetCell(Cell cell) {
        return cell instanceof ExitCell;
    }
}


public class Game {

    private final HexagonField field;
    private final FinishGameRulesHandler finishGameRulesHandler;
    private final Robot robot;

    public Game(FinishGameRulesHandlerFactory
finishGameRulesHandlerFactoryFactory, FieldFactory fieldFactory){
        field = fieldFactory.create();
        this.finishGameRulesHandler =
finishGameRulesHandlerFactoryFactory.create(field);
        robot = (Robot) field.getSpawnedObjects().stream().filter(obj -> obj
instanceof Robot).findFirst().get();
    }

    public void doStep(HexagonDirection direction){
        if (finishGameRulesHandler.isGameOver()){
            return;
```

```
        }

        robot.move(direction);
        finishGameRulesHandler.updateGameState();
    }

    public boolean isGameOver(){
        return finishGameRulesHandler.isGameOver();
    }

    public boolean isPlayerWin(){
        return finishGameRulesHandler.isPlayerWin();
    }

    public HexagonField getField() {
        return field;
    }
}


public enum HexagonDirection {
    RIGHT(1, 0),
    RIGHT_UP(1, 1),
    LEFT_UP(0, 1),
    LEFT(-1, 0),
    LEFT_DOWN(0, -1),
    RIGHT_DOWN(1, -1);

    private final Position position;

    HexagonDirection(int x, int y){
        position = new Position(x, y);
    }

    public Position toPosition(boolean isOddLine){
        return isOddLine && this != RIGHT && this != LEFT ? position.add(-1,
0) : new Position(position);
    }
}


public class HexagonField {

    private final Cell[][] cells;
    private final List<CellObject> spawnedObjects = new ArrayList<>();
    private final int width;
    private final int height;

    public HexagonField(int width, int height, Position exitPosition){
        if (width < 1 || height < 1){
            throw new InvalidParameterException();
        }

        this.width = width;
        this.height = height;

        cells = new Cell[width][height];
        for (int x = 0; x < width; x++){
            for (int y = 0; y < height; y++){
                cells[x][y] = new ColoredCell(this, new Position(x, y));
            }
        }
```

```
        cells[exitPosition.x][exitPosition.y] = new ExitCell(this,
exitPosition);
    }

    public void spawnObject(CellObject cellObject, Cell cell){
        if (spawnedObjects.contains(cellObject)
                || cell.getContainedObject() != null
                || (!(cellObject instanceof Robot) && cell instanceof
ExitCell)){
            throw new InvalidParameterException();
        }

        spawnedObjects.add(cellObject);
        cell.setContainedObject(cellObject);
    }

    public void despawnObject(CellObject cellObject){
        if (!spawnedObjects.contains(cellObject)){
            return;
        }

        for (int x = 0; x < cells.length; x++){
            for (int y = 0; y < cells[0].length; y++){
                if (cellObject == cells[x][y].getContainedObject()){
                    cells[x][y].setContainedObject(null);
                }
            }
        }

        spawnedObjects.remove(cellObject);
    }

    public Cell getCell(Position position){
        try {
            return cells[position.x][position.y];
        }
        catch (ArrayIndexOutOfBoundsException e){
            return null;
        }
    }

    public int getWidth() {
        return width;
    }

    public int getHeight() {
        return height;
    }

    public List<CellObject> getSpawnedObjects() {
        return spawnedObjects;
    }
}


public class Position {

    public int x;
    public int y;

    public Position(){}

    public Position(int x, int y) {
        this.x = x;
```

```
        this.y = y;
    }

    public Position(Position position){
        x = position.x;
        y = position.y;
    }

    public Position add(int x, int y){
        return add(new Position(x, y));
    }

    public Position add(Position position) {
        return new Position(x + position.x, y + position.y);
    }

    @Override
    public String toString(){
        return "x: " + x + " y: " + y;
    }
}


public class GameFrame extends JFrame {

    private final Game game;
    private final PathFinderFactory pathFinder = new
NeighborPathFinderFactory();
    private final List<FinishGameRulesHandlerFactory> rulesList = new
ArrayList<>(){{
        add(new FinishGameRulesHandlerFactory(new ArrayList<>() {{
            add(new FinishGameRulesHandler.RuleParameters(new
KeysFinishGameRuleFactory(pathFinder), false, RuleLinkType.AND));
            add(new FinishGameRulesHandler.RuleParameters(new
ExitFinishGameRuleFactory(pathFinder), false, RuleLinkType.AND));
        }}));
        add(new FinishGameRulesHandlerFactory(new ArrayList<>() {{
            add(new FinishGameRulesHandler.RuleParameters(new
KeysFinishGameRuleFactory(pathFinder), true, RuleLinkType.AND));
            add(new FinishGameRulesHandler.RuleParameters(new
ExitFinishGameRuleFactory(pathFinder), false, RuleLinkType.AND));
        }}));
        add(new FinishGameRulesHandlerFactory(new ArrayList<>() {{
            add(new FinishGameRulesHandler.RuleParameters(new
KeysFinishGameRuleFactory(pathFinder), false, RuleLinkType.OR));
            add(new FinishGameRulesHandler.RuleParameters(new
ExitFinishGameRuleFactory(pathFinder), false, RuleLinkType.OR));
        }}));
        add(new FinishGameRulesHandlerFactory(new ArrayList<>() {{
            add(new FinishGameRulesHandler.RuleParameters(new
RobotStepsFinishGameRuleFactory(RobotStepsFinishGameRule.RuleMode.MORE, 10),
false, RuleLinkType.AND));
            add(new FinishGameRulesHandler.RuleParameters(new
ExitFinishGameRuleFactory(pathFinder), false, RuleLinkType.AND));
        }}));
    }};

    public GameFrame() {
        FinishGameRulesHandlerFactory selectedHandlerFactory =
rulesList.get(Utils.rnd.nextInt(rulesList.size()));
        JOptionPane.showMessageDialog(null,
selectedHandlerFactory.toString());
        game = new Game(selectedHandlerFactory, new
RandomPositionsFieldFactory(8, 8, 3));
```

```
        HexagonFieldView hexagonFieldView = new
HexagonFieldView(game.getField());

        setContentPane(hexagonFieldView);
        setSize(hexagonFieldView.getSize());
        setResizable(false);
        setFocusable(true);

        addKeyListener(new KeyController());
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    private class KeyController implements KeyListener {

        @Override
        public void keyTyped(KeyEvent e) {
        }

        @Override
        public void keyPressed(KeyEvent e) {
            int code = e.getKeyCode();

            if(code == KeyEvent.VK_Q) {
                game.doStep(HexagonDirection.LEFT_UP);
            }
            else if(code == KeyEvent.VK_W) {
                game.doStep(HexagonDirection.RIGHT_UP);
            }
            else if(code == KeyEvent.VK_A) {
                game.doStep(HexagonDirection.LEFT);
            }
            else if(code == KeyEvent.VK_S) {
                game.doStep(HexagonDirection.RIGHT);
            }
            else if(code == KeyEvent.VK_Z) {
                game.doStep(HexagonDirection.LEFT_DOWN);
            }
            else if(code == KeyEvent.VK_X) {
                game.doStep(HexagonDirection.RIGHT_DOWN);
            }

            repaint();

            if (game.isGameOver()){
                JOptionPane.showMessageDialog(null, game.isPlayerWin() ?
"Player is WIN" : "Player is LOSE");
                System.exit(0);
            }
        }

        @Override
        public void keyReleased(KeyEvent e) {
        }
    }
}


public class HexagonFieldView extends JPanel {

    private final Position margins = new Position(10, 10);

    public HexagonFieldView(HexagonField field) {
        setLayout(null);
        setSize(
```

```
                (int)(margins.x * 2 + CellView.CELL_SIZE * (field.getWidth()
+ 1f) * CellView.CELL_WIDTH_MOD),
                (int)(margins.y * 2 + CellView.CELL_SIZE * (field.getHeight()
+ 1) * 0.75f)
        );

        for (int x = 0; x < field.getWidth(); x++){
            for (int y = 0; y < field.getHeight(); y++){
                int modelY = field.getHeight() - y - 1;
                boolean withOffset = modelY % 2 == 0;

                CellView cellView = new CellView(field.getCell(new
Position(x, modelY)));
                cellView.setBounds(
                        (int)(margins.x + CellView.CELL_SIZE *
CellView.CELL_WIDTH_MOD * (x + (withOffset ? 0.5f : 0))),
                        (int)(margins.y + CellView.CELL_SIZE * y * 0.75f),
                        CellView.CELL_SIZE, CellView.CELL_SIZE
                );
                add(cellView);
            }
        }

        setFocusable(true);
    }
}


public class CellView extends JPanel {

    public static final int CELL_SIZE = 65;
    public static final float CELL_WIDTH_MOD = 0.866025f;

    private static final String EXIT_LETTER = "ET";

    private static final float[] xHexagonPoints = new float[]{
            0f, CELL_WIDTH_MOD, CELL_WIDTH_MOD, 0f, -CELL_WIDTH_MOD, -
CELL_WIDTH_MOD, 0f
    };
    private static final float[] yHexagonPoints = new float[]{
            1f, 0.5f, -0.5f, -1f, -0.5f, 0.5f, 1f
    };

    private final Cell cell;
    private CellObjectView cellObjectView;

    public CellView(Cell cell) {
        this.cell = cell;

        setPreferredSize(new Dimension(CELL_SIZE, CELL_SIZE));
        setBackground(new Color(0, 0, 0, 0));
    }

    @Override
    public void paint(Graphics g) {
        super.paint(g);

        CellObject cellObject = cell.getContainedObject();
        if (cellObjectView == null && cellObject != null
                || cellObjectView != null && cellObjectView.cellObject !=
cellObject){
            cellObjectView = getCellObjectView(cellObject);
        }
```

```
        Graphics2D gr2d = (Graphics2D) g;
        drawHexagon(gr2d);

        if ((cellObjectView == null || !cellObjectView.paint(gr2d)) && cell
instanceof ExitCell){
            drawString(gr2d);
        }
    }

    private void drawHexagon(Graphics2D gr2d){
        gr2d.setPaint(getCellColor());
        gr2d.fillPolygon(
                getPolygonPoints(true, CELL_SIZE/2f),
                getPolygonPoints(false, CELL_SIZE/2f),
                xHexagonPoints.length
        );

        gr2d.setStroke(new BasicStroke(2));
        gr2d.setPaint(Color.black);
        gr2d.drawPolygon(
                getPolygonPoints(true, CELL_SIZE/2f),
                getPolygonPoints(false, CELL_SIZE/2f),
                xHexagonPoints.length
        );
    }

    private void drawString(Graphics2D gr2d){
        gr2d.setColor(Color.black);
        gr2d.setFont(new Font("Microsoft JhengHei Light", Font.BOLD, 20));

        FontMetrics fm = gr2d.getFontMetrics();
        int msgWidth = fm.stringWidth(EXIT_LETTER);
        int msgHeight = fm.getHeight();

        gr2d.drawString(EXIT_LETTER, (CELL_SIZE - msgWidth)/2, CELL_SIZE / 2
+ msgHeight/4);
    }

    private int[] getPolygonPoints(boolean isX, float scale){
        float[] points = isX ? xHexagonPoints : yHexagonPoints;
        int[] result = new int[points.length];

        for (int i = 0; i < points.length; i++){
            result[i] = Math.round((points[i] + 1f) * scale);
        }

        return result;
    }

    private Color getCellColor(){
        return cell instanceof ColoredCell ?
((ColoredCell)cell).getCurrentFootprint() : Color.white;
    }

    private CellObjectView getCellObjectView(CellObject cellObject){
        if (cellObject instanceof Key){
            return new KeyView(cellObject);
        }
        else if (cellObject instanceof Robot){
            return new RobotView(cellObject);
        }

        return null;
```

```
    }
}


public class KeyView extends CellObjectView {

    public KeyView(CellObject cellObject) {
        super(cellObject);
    }

    @Override
    protected String getLetter() {
        return "K";
    }
}
```

3.7 Реализация ключевых тестовых случаев

```
public class CommonTests {
    private final int fieldWidth = 7;
    private final int fieldHeight = 11;
    private final PathFinderFactory pathFinder = new
NeighborPathFinderFactory();
    private final FinishGameRulesHandlerFactory finishGameRulesHandlerFactory
= new FinishGameRulesHandlerFactory(
            new ArrayList<>() {{
                add(new FinishGameRulesHandler.RuleParameters(new
ExitFinishGameRuleFactory(pathFinder), false, RuleLinkType.AND));
                add(new FinishGameRulesHandler.RuleParameters(new
KeysFinishGameRuleFactory(pathFinder), false, RuleLinkType.AND));
            }}
    );

    private Game game;
    private HexagonField field;
    private Robot robot;

    @BeforeEach
    public void Setup(){
        game = new Game(finishGameRulesHandlerFactory, new
TestsFieldFactory(fieldWidth, fieldHeight));
        field = game.getField();
        robot = (Robot) field.getSpawnedObjects().stream().filter(obj -> obj
instanceof Robot).findFirst().get();
    }

    @Test
    // заспавнить поле размером width и height > проверить, что получившиеся
размеры совпадают
    public void FieldSizeTest(){
        assertEquals(fieldWidth, field.getWidth());
        assertEquals(fieldHeight, field.getHeight());

        for (int x = 0; x < fieldWidth; x++){
            for (int y = 0; y < fieldHeight; y++){
                assertNotNull(field.getCell(new Position(x, y)), "x: " + x +
" y: " + y + " is null");
            }
        }

        assertNull(field.getCell(new Position(fieldWidth + 1, fieldHeight +
1)));
    }

    @Test
    // пройтись кругом по соседям ячейки и вернутся обратно > должны прийти к
той же ячейке
    public void CircleMoveTest(){
        ColoredCell initialCell = (ColoredCell) robot.getCell();

        robot.move(HexagonDirection.LEFT_DOWN);
        robot.move(HexagonDirection.RIGHT);
        initialCell.setCurrentFootprint(ColoredCell.defaultFootprint);
        robot.move(HexagonDirection.LEFT_UP);

        assertEquals(initialCell, robot.getCell());
    }
```

```
    @Test
    // заспавнить объект в ячейке > объект в ячейке
    public void SpawnObjectTest(){
        CellObject spawnedObject = new Key();
        Cell cell = field.getCell(new Position(0, 0));

        field.spawnObject(spawnedObject, cell);

        assertNotNull(field.getSpawnedObjects().stream().filter(obj -> obj ==
spawnedObject).findFirst().orElse(null));
        assertEquals(spawnedObject, cell.getContainedObject());
        assertEquals(cell, spawnedObject.getCell());
    }

    @Test
    // задеспавнить объект в ячейке > объекта нет в ячейке
    public void DespawnObjectTest(){
        CellObject despawnedObject = robot;
        Cell cell = robot.getCell();

        field.despawnObject(despawnedObject);

        assertNull(field.getSpawnedObjects().stream().filter(obj -> obj ==
despawnedObject).findFirst().orElse(null));
        assertNull(cell.getContainedObject());
        assertNull(despawnedObject.getCell());
    }

    @Test
    // движения робота в ячейку без следа > перемещение совершилось, ячейка
со следом
    public void MoveToWithoutFootprintTest(){
        Cell initialCell = robot.getCell();

        robot.move(HexagonDirection.RIGHT);

        assertNull(initialCell.getContainedObject());
        assertEquals(initialCell.getNeighbor(HexagonDirection.RIGHT),
robot.getCell());
        assertEquals(robot.getFootprintColor(), ((ColoredCell)
robot.getCell()).getCurrentFootprint());
    }

    @Test
    // движение робота в ячейку со следом > движения не произошло
    public void MoveToWithFootprintTest(){
        Cell initialCell = robot.getCell();
        ColoredCell targetCell = (ColoredCell)
initialCell.getNeighbor(HexagonDirection.RIGHT);

        targetCell.setCurrentFootprint(robot.getFootprintColor());
        robot.move(HexagonDirection.RIGHT);

        assertEquals(robot, initialCell.getContainedObject());
        assertNull(targetCell.getContainedObject());
    }

    @Test
    // движение робота к выходу, ключей нет > игра выиграна
    public void MoveToExitWithoutKeysTest(){
        List<CellObject> keys = field.getSpawnedObjects().stream().filter(obj
-> obj instanceof Key).toList();
```

```
        for (CellObject key : keys){
            field.despawnObject(key);
        }

        game.doStep(HexagonDirection.RIGHT);
        game.doStep(HexagonDirection.RIGHT);

        assertTrue(game.isGameOver());
        assertTrue(game.isPlayerWin());
    }

    @Test
    // подбор ключа роботом > ключа на карте нет
    public void KeyCollectTest(){
        int initKeysCount = field.getSpawnedObjects().stream().filter(obj ->
obj instanceof Key).toList().size();
        robot.move(HexagonDirection.LEFT);
        robot.move(HexagonDirection.LEFT);

        List<CellObject> keys = field.getSpawnedObjects().stream().filter(obj
-> obj instanceof Key).toList();
        assertEquals(initKeysCount - 1, keys.size());
        assertEquals(robot, robot.getCell().getContainedObject());
    }
}
```

4 Вторя итерация разработки

4.1 Функциональные требования (сценарии)

Сценарий «Игра не завершена» при правиле «Собрать все ключи»

1. Игра спрашивает у Обработчика правил игры, завершена ли игра
2. Обработчик правил игры спрашивает у каждого Правила завершена ли игра
3. Одно из Правил игры спрашивает у Поля доходимость Робота до Ключей. Если пути нет, то Игра считает, что игрок проиграл
4. Одно из Правил игры спрашивает у Поля остались ли на карте ключи. Если на карте нет ключей, то Игра считает, что игрок победил

Сценарий «Игра не завершена» при правиле «Шагов меньше заданного и выход достигнут»

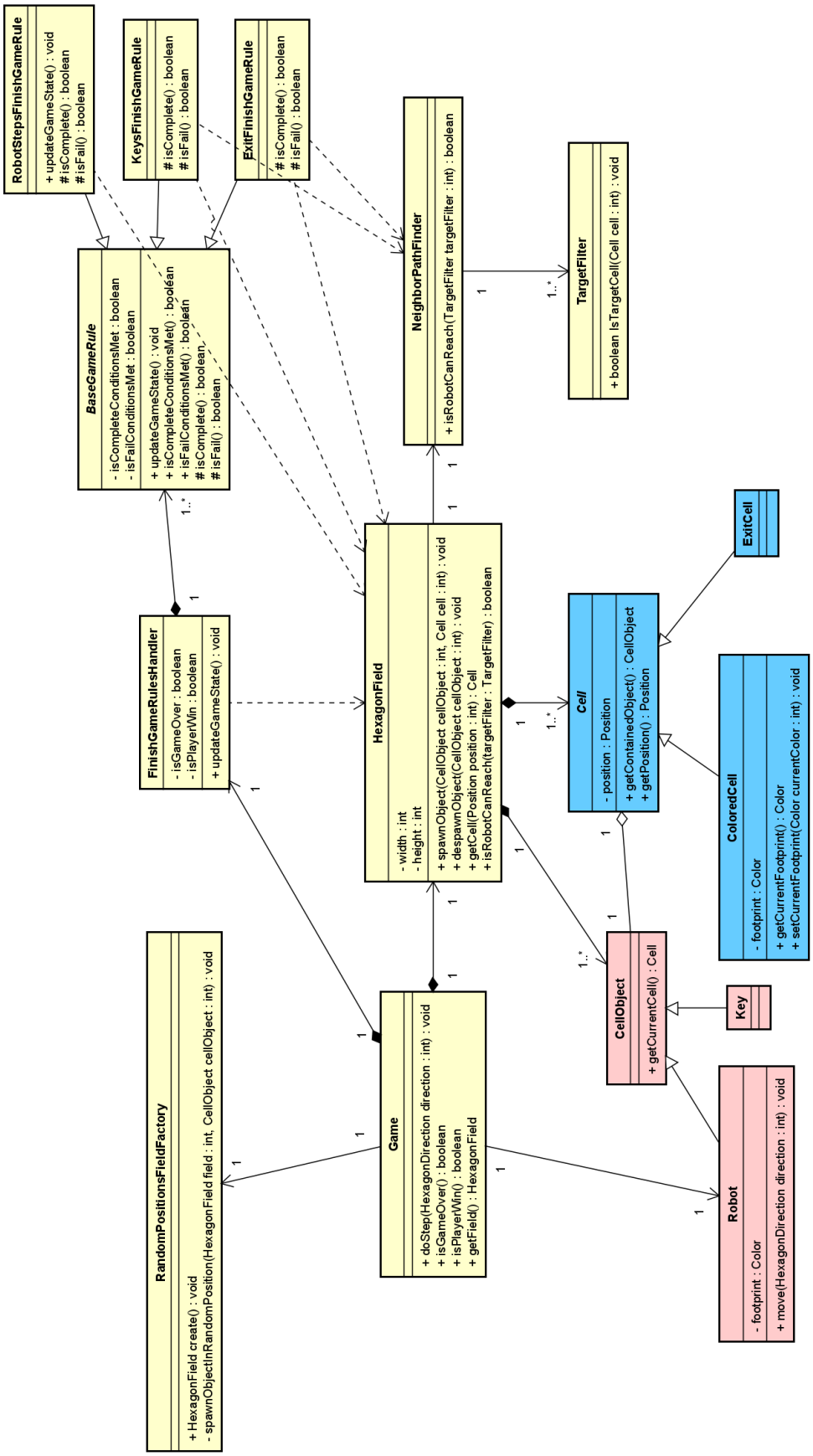
1. Игра спрашивает у Обработчика правил игры, завершена ли игра
2. Обработчик правил игры спрашивает у каждого Правила завершена ли игра
3. Одно из Правил игры спрашивает у Поля доходимость Робота до Выхода. Если пути нет, то Игра считает, что игрок проиграл
4. Одно из Правил игры спрашивает у Поля количество закрашенных клеток роботом. Если их кол-во превышает или равняется заданному значению, то Игра считает, что игрок проиграл

4.2 Словарь предметной области

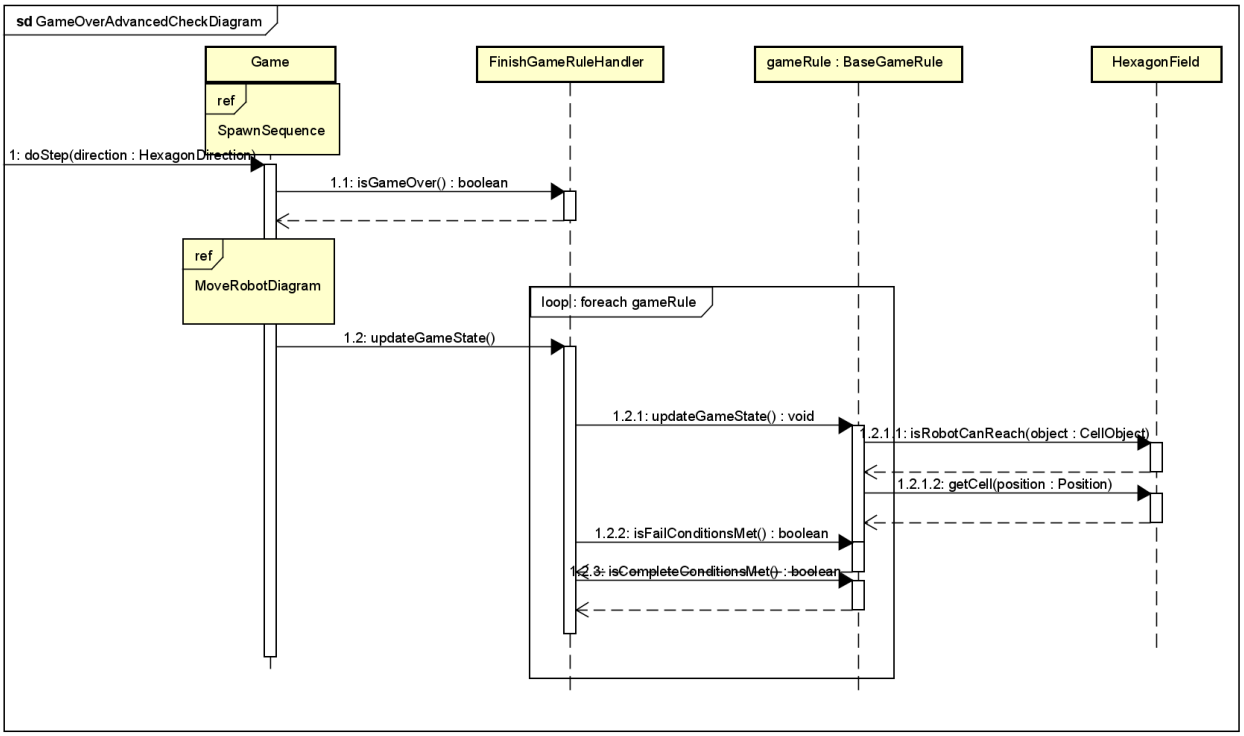
Правило – атомарное условие, которое должно выполняться или не выполняться для победы игрока

Обработчик правил игры – совокупность Правил, связанных логическими операторами И, ИЛИ и НЕ (без вложенности), которые формируют окончательное правило победы игрока в игровой сессии

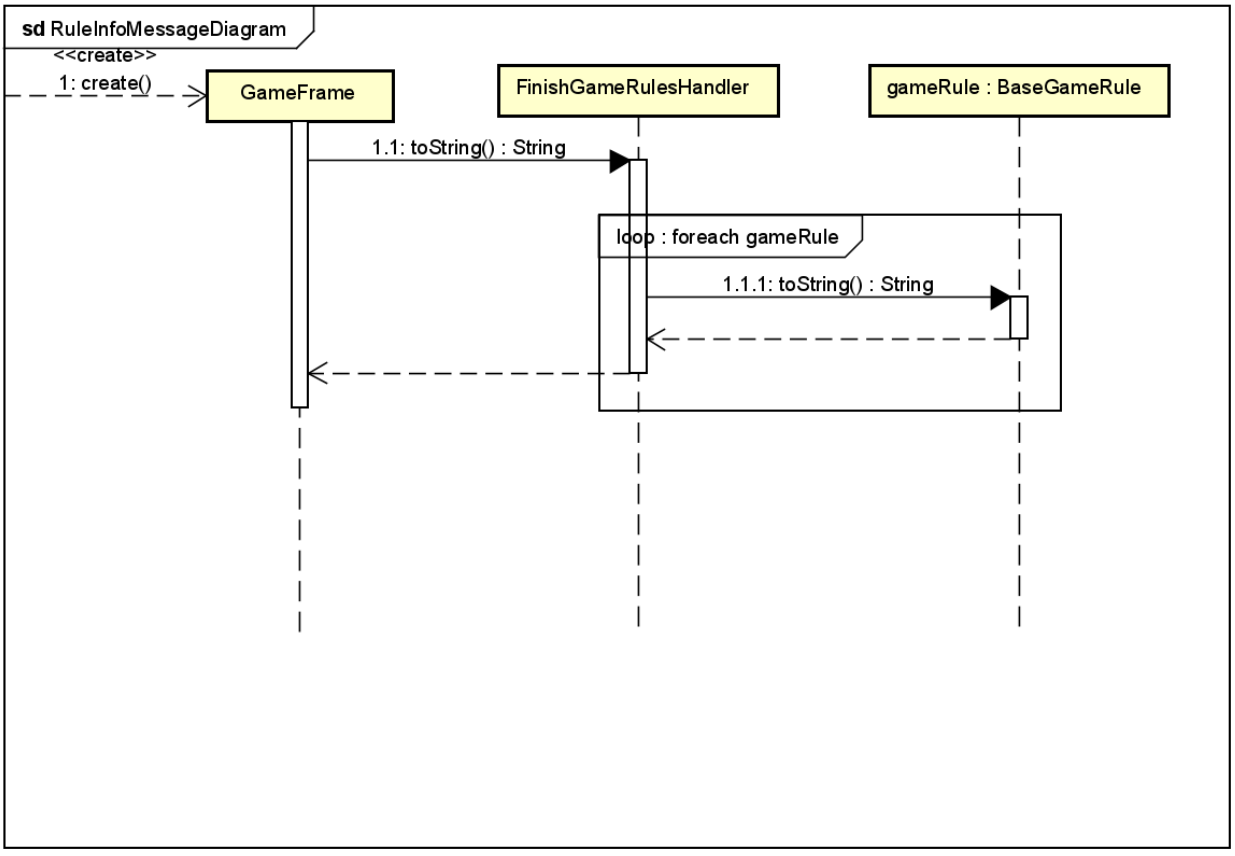
4.3 Структура программы на уровне классов



4.4 Типовые процессы в программе



Проверка победы/проигрыша игрока по заданным правилам



Вывод правил в начале игры

4.5 Реализация ключевых классов

```
public abstract class BaseGameRule implements FinishGameRule {

        protected HexagonField field;
        private boolean isCompleteConditionsMet;
        private boolean isFailConditionsMet;

        public BaseGameRule(HexagonField field) {
            this.field = field;
        }

        @Override
        public void updateGameState() {
            isCompleteConditionsMet = isComplete();
            isFailConditionsMet = isFail();
        }

        @Override
        public boolean isCompleteConditionsMet() {
            return isCompleteConditionsMet;
        }

        @Override
        public boolean isFailConditionsMet() {
            return isFailConditionsMet;
        }

        protected abstract boolean isComplete();
        protected abstract boolean isFail();
    }
public class FinishGameRulesHandler {

        private final List<RuleParameters> gameRulesParameters;
        private boolean isGameOver;
        private boolean isPlayerWin;

        public FinishGameRulesHandler(HexagonField field, List<RuleParameters>
    gameRulesParameters){
            this.gameRulesParameters = gameRulesParameters;

            for (RuleParameters gameRulesParameter : gameRulesParameters){
                gameRulesParameter.initialize(field);
            }
        }

        public void updateGameState(){
            if (isGameOver){
                return;
            }

            RulesInfo rulesInfo = new RulesInfo(gameRulesParameters);
            handleGameStateResult(rulesInfo.updateRulesAndCountIndependent());
            handleGameStateResult(rulesInfo.handleUpdatedRules());
        }

        public boolean isGameOver(){
            return isGameOver;
        }

        public boolean isPlayerWin(){
            return isPlayerWin;
        }
```

```
    private void handleGameStateResult(GameStateResult gameStateResult){
        if (gameStateResult == GameStateResult.None){
            return;
        }

        isGameOver = true;
        this.isPlayerWin = gameStateResult == GameStateResult.PlayerWin;
    }

    @Override
    public String toString() {
        StringBuilder result = new StringBuilder();
        for (RuleParameters parameters : gameRulesParameters) {
            if (parameters.ruleLinkType == RuleLinkType.OR){
                continue;
            }

            if (result.length() != 0) {
                result.append(" И ");
            }

            if (parameters.completeConditionsIsNegative){
                result.append("НЕ ");
            }

            result.append(parameters.rule.toString());
        }

        for (RuleParameters parameters : gameRulesParameters) {
            if (parameters.ruleLinkType != RuleLinkType.OR) {
                continue;
            }

            if (result.length() != 0) {
                result.append(" ИЛИ ");
            }

            if (parameters.completeConditionsIsNegative){
                result.append("НЕ ");
            }

            result.append(parameters.rule.toString());
        }

        return result.toString();
    }

    public static class RuleParameters {

        private FinishGameRuleFactory ruleFactory;
        private FinishGameRule rule;

        public final boolean completeConditionsIsNegative;
        public final RuleLinkType ruleLinkType;

        public RuleParameters(FinishGameRuleFactory ruleFactory, boolean
completeConditionsIsNegative, RuleLinkType ruleLinkType) {
            this.ruleFactory = ruleFactory;
            this.completeConditionsIsNegative =
completeConditionsIsNegative;
            this.ruleLinkType = ruleLinkType;
        }

        public void initialize(HexagonField field){
```

```
            rule = ruleFactory.create(field);
            ruleFactory = null;
        }

        public String toStringRule(){
            return rule == null ? ruleFactory.create(null).toString() :
rule.toString();
        }
    }

    private class RulesInfo {

        private final List<RuleParameters> gameRulesParameters;

        public boolean haveNotIndependent;
        public boolean notIndependentComplete;
        public boolean notIndependentFail;

        public int independentCount;
        public int failIndependentCount;

        public RulesInfo(List<RuleParameters> gameRulesParameters) {
            this.gameRulesParameters = gameRulesParameters;
        }

        public GameStateResult updateRulesAndCountIndependent(){
            for (RuleParameters parameters : gameRulesParameters) {
                parameters.rule.updateGameState();

                boolean isCompleteConditionsMet =
parameters.rule.isCompleteConditionsMet();
                boolean isFailConditionsMet =
parameters.rule.isFailConditionsMet();

                if (parameters.completeConditionsIsNegative){
                    isCompleteConditionsMet = !isCompleteConditionsMet;
                }

                if (parameters.ruleLinkType == RuleLinkType.AND){
                    handleAndRule(parameters, isCompleteConditionsMet,
isFailConditionsMet);
                }
                else {
                    if (isCompleteConditionsMet){
                        return GameStateResult.PlayerWin;
                    }

                    handleOrRule(parameters, isFailConditionsMet);
                }
            }

            return GameStateResult.None;
        }

        private void handleAndRule(RuleParameters parameters, boolean
isCompleteConditionsMet, boolean isFailConditionsMet){
            haveNotIndependent = true;
            notIndependentComplete &= isCompleteConditionsMet;

            if (!parameters.completeConditionsIsNegative) {
                notIndependentFail |= isFailConditionsMet;
            }
        }
```

```
        private void handleOrRule(RuleParameters parameters, boolean
    isFailConditionsMet){
            independentCount++;

            if (!parameters.completeConditionsIsNegative &&
    isFailConditionsMet){
                failIndependentCount++;
            }
        }

        public GameStateResult handleUpdatedRules(){
            if (haveNotIndependent){
                if (notIndependentComplete){
                    return GameStateResult.PlayerWin;
                }

                if (notIndependentFail){
                    if (independentCount == 0){
                        return GameStateResult.PlayerLose;
                    }

                    failIndependentCount++;
                }

                independentCount++;
            }

            if (independentCount == failIndependentCount){
                return GameStateResult.PlayerLose;
            }

            return GameStateResult.None;
        }
    }

    private enum GameStateResult {
        None, PlayerWin, PlayerLose
    }
}
public class FinishGameRulesHandlerFactory {

    private final List<FinishGameRulesHandler.RuleParameters> gameRules;

    public
FinishGameRulesHandlerFactory(List<FinishGameRulesHandler.RuleParameters>
gameRules){
        this.gameRules = gameRules;
    }

    public FinishGameRulesHandler create(HexagonField field){
        return new FinishGameRulesHandler(field, gameRules);
    }

    @Override
    public String toString() {
        StringBuilder result = new StringBuilder();
        for (FinishGameRulesHandler.RuleParameters parameters : gameRules)
{
            if (parameters.ruleLinkType == RuleLinkType.OR){
                continue;
            }

            if (result.length() != 0) {
                result.append(" И ");
```

```
                }

                if (parameters.completeConditionsIsNegative){
                    result.append("НЕ ");
                }

                result.append(parameters.toStringRule());
            }

            for (FinishGameRulesHandler.RuleParameters parameters : gameRules)
    {
                if (parameters.ruleLinkType != RuleLinkType.OR) {
                    continue;
                }

                if (result.length() != 0) {
                    result.append(" ИЛИ ");
                }

                if (parameters.completeConditionsIsNegative){
                    result.append("НЕ ");
                }

                result.append(parameters.toStringRule());
            }

            return result.toString();
        }
    }

public class RobotStepsFinishGameRule extends BaseGameRule {

    private final RuleMode ruleMode;
    private final int value;
    private Cell lastRobotCell = null;
    private int stepsCount;

    public RobotStepsFinishGameRule(HexagonField field, RuleMode ruleMode,
int value) {
        super(field);

        this.ruleMode = ruleMode;
        this.value = value;
    }

    @Override
    public void updateGameState() {
        Robot robot = (Robot) field.getSpawnedObjects().stream().filter(obj -
> obj instanceof Robot).findFirst().orElse(null);

        if (!robot.getCell().equals(lastRobotCell)){
            lastRobotCell = robot.getCell();
            stepsCount++;
        }

        super.updateGameState();
    }

    @Override
    protected boolean isComplete() {
        switch (ruleMode){
            case LESS:
                return stepsCount < value;
            case EQUALS:
```

```
                return stepsCount == value;
            case MORE:
                return stepsCount > value;
            default:
                return false;
        }
    }

    @Override
    protected boolean isFail() {
        switch (ruleMode){
            case LESS:
                return !isComplete();
            case EQUALS:
                return stepsCount > value;
            case MORE:
            default:
                return false;
        }
    }

    @Override
    public String toString() {
        return "количество шагов должно быть "
                + (ruleMode == RuleMode.LESS ? "меньше" : ruleMode ==
RuleMode.EQUALS ? "равно" : "больше")
                + " " + value;
    }

    public enum RuleMode {
        LESS, EQUALS, MORE
    }
}

public class KeysFinishGameRule extends BaseGameRule {

    private final PathFinder pathFinder;

    public KeysFinishGameRule(HexagonField field, PathFinder pathFinder) {
        super(field);
        this.pathFinder = pathFinder;
    }

    @Override
    protected boolean isComplete() {
        List<CellObject> spawnedObjects = field.getSpawnedObjects();
        int keysCount = 0;

        for (CellObject cellObject : spawnedObjects){
            if (cellObject instanceof Key){
                keysCount++;
            }
        }
        return keysCount == 0;
    }

    @Override
    protected boolean isFail() {
        return !isComplete() &&
!pathFinder.isRobotCanReach(Utils.KEY_TARGET_FILTER);
    }

    @Override
    public String toString() {
```

```
        return "собрать все ключи (метка K)";
    }
}

public class ExitFinishGameRule extends BaseGameRule {

    private final PathFinder pathFinder;

    public ExitFinishGameRule(HexagonField field, PathFinder pathFinder) {
        super(field);
        this.pathFinder = pathFinder;
    }

    @Override
    protected boolean isComplete() {
        Robot robot = (Robot) field.getSpawnedObjects().stream().filter(obj -
> obj instanceof Robot).findFirst().orElse(null);
        return robot.getCell() instanceof ExitCell;
    }

    @Override
    protected boolean isFail() {
        return !pathFinder.isRobotCanReach(Utils.EXIT_TARGET_FILTER);
    }

    @Override
    public String toString() {
        return "достигнуть выхода (метка ET)";
    }
}
```

4.6 Реализация ключевых тестовых случаев

```
// робот всегда по центру, левее на 2 клетки ключ, правее на 2 клетки выход
public class TestsFieldFactory implements FieldFactory {

    private final int fieldWidth;
    private final int fieldHeight;

    public TestsFieldFactory(int fieldWidth, int fieldHeight){
        this.fieldWidth = fieldWidth;
        this.fieldHeight = fieldHeight;
    }

    @Override
    public HexagonField create() {
        Position center = new Position(fieldWidth / 2, fieldHeight / 2);
        HexagonField field = new HexagonField(fieldWidth, fieldHeight,
center.add(2, 0));

        field.spawnObject(new Robot(field, Color.ORANGE),
field.getCell(center));
        field.spawnObject(new Key(), field.getCell(center.add(-2, 0)));

        return field;
    }
}
```

```
@Test
// движение робота к выходу, ключей нет > игра выиграна
public void MoveToExitWithoutKeysTest(){
    List<CellObject> keys = field.getSpawnedObjects().stream().filter(obj ->
obj instanceof Key).toList();

    for (CellObject key : keys){
        field.despawnObject(key);
    }

    game.doStep(HexagonDirection.RIGHT);
    game.doStep(HexagonDirection.RIGHT);

    assertTrue(game.isGameOver());
    assertTrue(game.isPlayerWin());
}
```

```
@Test
// движение робота к выходу, ключи есть > игра продолжается
public void MoveToExitWithKeysTest(){
    game.doStep(HexagonDirection.RIGHT);
    game.doStep(HexagonDirection.RIGHT);

    assertFalse(game.isGameOver());
}
```

```
@Test
// все ячейки вокруг робота со следом > игра проиграна
public void AllAroundWithFootprintTest(){
    game.doStep(HexagonDirection.LEFT_DOWN);
    game.doStep(HexagonDirection.RIGHT_DOWN);
```

```
        game.doStep(HexagonDirection.RIGHT);
        game.doStep(HexagonDirection.RIGHT_UP);
        game.doStep(HexagonDirection.LEFT_UP);
        game.doStep(HexagonDirection.LEFT_DOWN);

        assertTrue(game.isGameOver());
        assertFalse(game.isPlayerWin());
    }

    @Test
    // нет прохода к выходу > игра проиграна
    public void CantReachExit(){
        game.doStep(HexagonDirection.RIGHT);
        game.doStep(HexagonDirection.RIGHT_DOWN);
        game.doStep(HexagonDirection.RIGHT);
        game.doStep(HexagonDirection.RIGHT_UP);
        game.doStep(HexagonDirection.LEFT_UP);
        game.doStep(HexagonDirection.LEFT);
        game.doStep(HexagonDirection.LEFT);

        assertTrue(game.isGameOver());
        assertFalse(game.isPlayerWin());
    }
```

5 Человеко-машинное взаимодействие

Общий вид главного экрана программы представлен ниже. На нём располагается игровое поле, на котором изображен робот, 3 ключа и ячейка выхода, а так же 2е закрашенные ячейки.

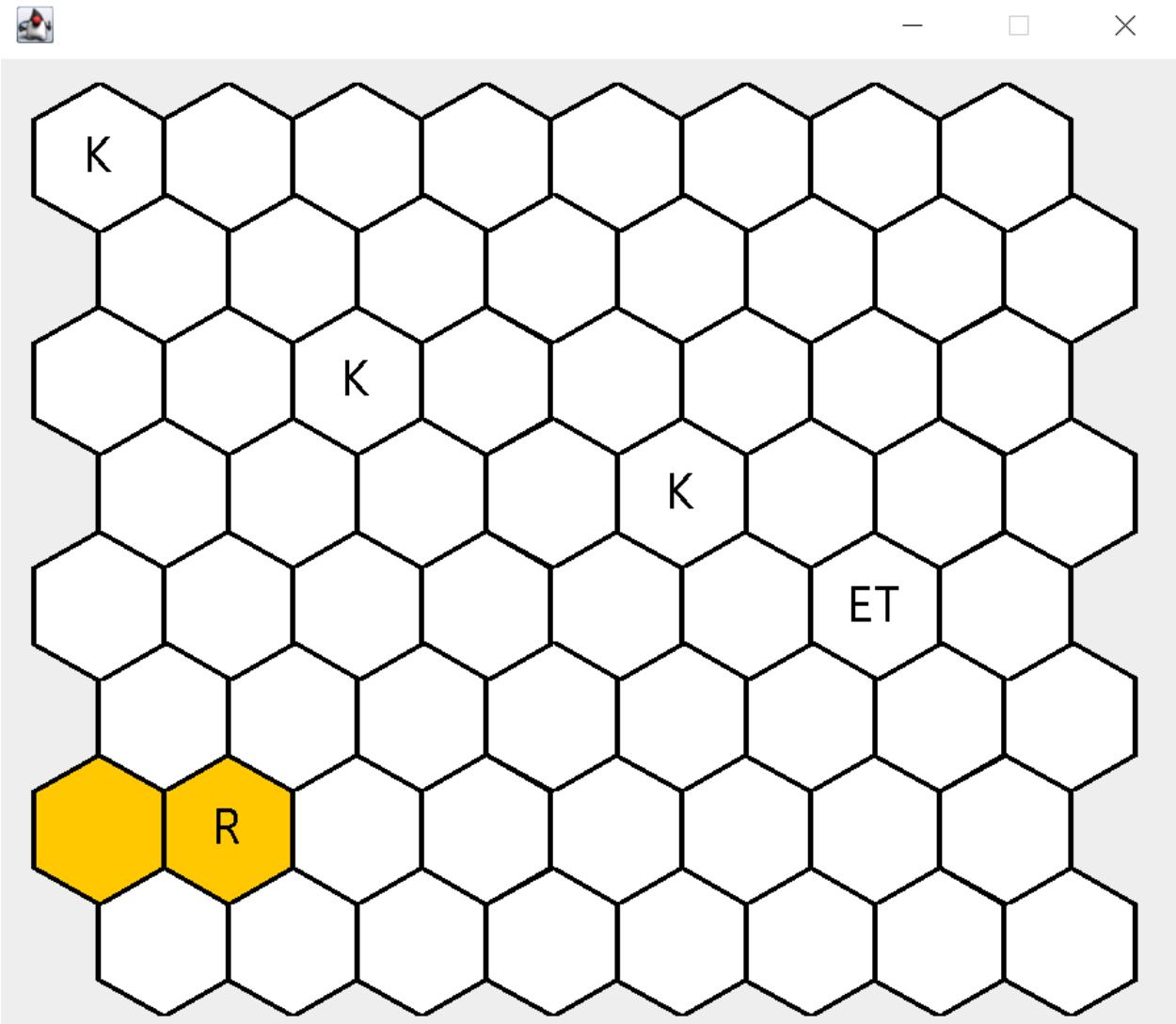


Рис. 1. Общий вид главного экрана программы

Управление активным роботом пользователь осуществляет с помощью клавиатуры.

Q – движение влево-вверх

W – движение вправо-вверх
A – движение влево
S – движение вправо
Z – движение влево-вниз
X – движение вправо-вниз

Изображение робота представлено на рисунке 2. Под ним ячейка закрашена цветом его следа.

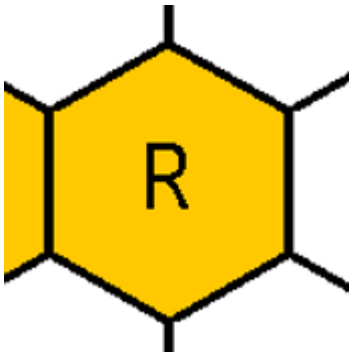


Рис. 2. Изображение робота

Ключ представлен аналогичным образом, но буквой “K”. Ячейка выхода буквами “ET”.

При запуске игры (Рис. 3) появляется условия победы. Каждый запуск выдаются случайные из определенного набора.

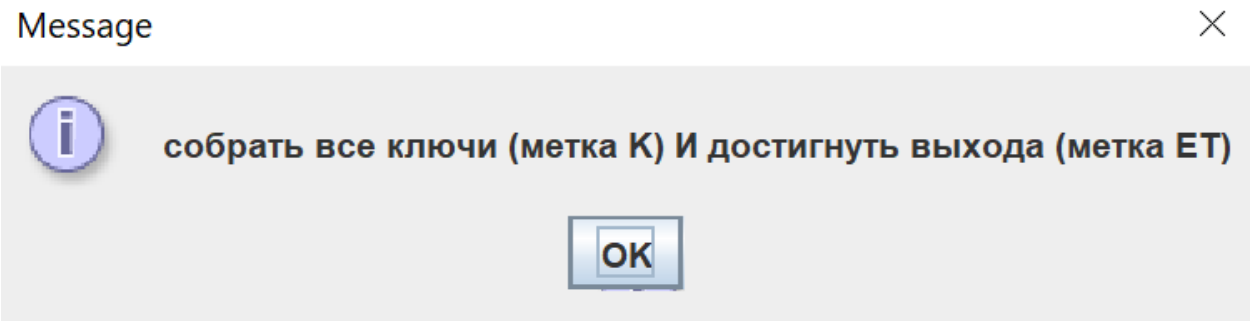


Рис. 3. Сообщение с условиями победы

Когда игрок проигрывает или выигрывает, то появляется подобное окно (Рис. 4) с сообщением о победе или проигрыше.

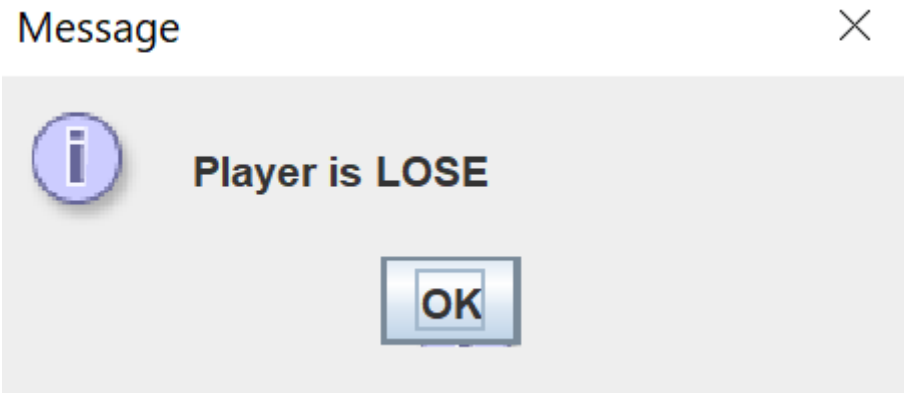


Рис. 4. Сообщение о проигрыше