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
и.о. зав. кафедрой _____ Сычев О.А.

ЗАДАНИЕ
на курсовую работу

Студент: Шаталов И.П.
Группа: ПрИн-367

1. Тема: «Проектирование и реализация программы с использованием объектно-ориентированного подхода» (индивидуальное задание – вариант №25_03)
Утверждена приказом от «05» февраля 2025г. № 183-ст

2. Срок представления работы к защите « 06 » июня 2025 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 «14» февраля 2025 г.

Руководитель проекта: _____ Литовкин Д.В.

Задание принял к исполнению: _____ Шаталов И.П.
«14» февраля 2025 г.

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 и реализация программы с использованием объектно-ориентированного подхода»
(индивидуальное задание – вариант №25_03)

Студент: Шаталов И.П.
Группа: ПрИн-367

Работа зачтена с оценкой _____ «___» _____ 2025 г.

Руководитель проекта, нормоконтроллер _____ Литовкин Д.В.

Волгоград 2025 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.

Дополнительные требования:

- Предусмотреть в программе точки расширения, используя которые можно реализовать вариативную часть программы (в дополнение к базовой функциональности).

Вариативность:

- Кроме статических версий шестиугольников, на поле могут существовать аналогичные, но подвижные шестиугольники с произвольным поведением перемещения в соседнюю позицию и с произвольным способом активации этого перемещения.

НЕ изменяя ранее созданные классы, а используя точки расширения, реализовать:

- шестиугольник, на котором можно оставлять след, перемещается в соседнюю не занятую позицию (если она имеется) при попытке встать на него.
- шестиугольник с выходом, меняется местами со случайным соседним шестиугольником, если игрок встал на него.

Уровень сложности: Средний-Высокий

2. Нефункциональные требования

1. Программа должна быть реализована на языке Java SE 22 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)

1) **Сценарий** “Игра завершается победой пользователя”
    1. **По указанию пользователя**, Игра стартует.

    2. **По указанию Игры**, Уровень создает Поле и заполняет его стенами, ключом, позицией игрока и выходом.
    3. **В ответ на запрос Игры**, Поле сообщает о позиции Игрока.
    4. **Делать** {
        4.1. **По указанию пользователя**
            4.1.1. **Игрок** перемещается в соседнюю активную клетку.
            4.1.2. Если клетка **содержит Выход**
                4.1.2.1. Точка_Выхода предоставляет значения для проверки выполнения условия игры (см. Сценарий “Выход с уровня”).
                4.1.2.2. Если проверка выполнении условия игры прошла успешно
                    4.1.2.2.1. **Игра** считает Игрока победителем.
        } **Пока хоть одна соседняя клетка активна**
    5. **Сценарий завершается.**

1.1) Альтернативный сценарий «Досрочное завершение игры **пользователем**». Сценарий **выполняется в любой точке** главного сценария.

1. **По указанию пользователя,** программа завершается без определения победителя.
2. **Сценарий завершается.**

2) Дочерний сценарий “Уровень заполняет Поле Стенами, Ключами и Выходом”.

1. **По указанию** Игры, Уровень заселяет Поле:

1.1. **Уровень создает** и **расставляет** последовательности Стен внутри Поля.

1.2. **Уровень создаёт** Позицию Игрока и **помещает** ее на Поле.

1.3. **Уровень создает** Ключ(и) и **помещает** их на Поле.

1.4. **Уровень создает** Точку_Выхода и **помещает** ее на поле.

2. **Сценарий завершается**.

3) Дочерний сценарий “Игрок подбирает Ключ”

1. **В ответ на запрос Игрока** Ячейка сообщает о своем содержимом.
2. **Игрок** обнаруживает в содержимом Ключ.
3. **Игрок добавляет** Ключ себе.
4. **Игрок сообщает** Ячейке об удалении Ключа из нее.
5. **Ячейка удаляет** Ключ из своего содержимого.
6. **Сценарий завершается.**

4) Дочерний сценарий “Ключа нет в Ячейке, на которой расположен Игрок”.

1. **В ответ на запрос Игрока** Ячейка сообщает о своем содержимом.
2. Игрок не обнаруживает в содержимом Ключа.
3. **Сценарий завершается.**

5) Дочерний сценарий “Игрок перемещается в Ячейку”

1. **По указанию пользователя Игрок** инициирует перемещение на выбранную Ячейку.

2. **В ответ на запрос Игрока** Ячейка сообщает о своем содержимом.
3. На основе полученной информации Игрок определяет возможность перемещения в данную клетку.
4. Позиция **Игрока перемещается** на выбранную пользователем ячейку.
5. **Ячейка**, на которой находится игрок, **становится** неактивной для повторного перемещения на нее.
6. **Сценарий завершается.**

6) Дочерний сценарий “Попадание на ячейку с Выходом”

1. **По указанию пользователя** Игрок попадает на Ячейку с Выходом.
2. Игрок предоставляет информацию о наличии Ключей.
3. Точка_Выхода и Уровень предоставляет Игре информацию об условии прохождения уровня и Ключах Игрока.
4. Если проверка условия выигрыша в Игре прошла успешно

    4.1. Игра завершается победой игрока.
5. Иначе

    5.1. Игра продолжается.

    5.2. Ячейка с Выходом **остается активной** для повторного перемещения.
6. **Сценарий завершается.**

7) Дочерний сценарий “Попытка перемещения на Ячейку со Стеной”

1. **В ответ на запрос Игрока** Ячейка предоставляет свое содержимое.
2. На основе полученной информации **Игрок решает**, что он не может переместиться в соседнюю ячейку.
3. **Сценарий завершается.**

8) Дочерний сценарий “Позиция Игрока не имеет соседних доступных для перемещения Ячеек”

1. **Игра ожидает** от пользователя указания по ходу.
2. **Сценарий завершается.**

3.3 Словарь предметной области

Игра - знает о Поле и Лабиринте. Игра инициирует создание Поля и расстановку всех сущностей на нем с помощью Лабиринта. Игра окончание игры.

знает	● Поле ● Уровень
умеет	● инициировать создание Поля и расстановку всех сущностей на нем с помощью Уровня ● определять окончание игры (и победителя)
предназначение	● Организация общего игрового цикла

Поле - область, состоящая из Ячеек. Знает о Ключах, находящихся на Поле, позиции Игрока и Выходе.

знает	● свои размеры ● Ячейки ● Уровень
умеет	● создавать себя из Ячеек ● предоставлять доступ к Ячейкам ● получать Игрока(или позицию Игрока)
предназначение	● Контейнер Ячеек и сущностей, которые располагаются внутри Ячеек

Ячейка - шестиугольная область Поля. Знает о шести соседних Ячейка. На ней могут одновременно располагаться Игрок и Точка_Выхода

знает	● соседние Ячейки ● о своем содержимом
умеет	● устанавливать соседство с другой Ячейкой ● предоставлять доступ к сущностям, располагающимся на Ячейке (Игрок, Ключ, Выход) ● добавлять/извлекать элементы, располагающимся на Ячейке (Игрок, Ключ). На одной Ячейке одновременно может располагаться только Игрок и только один Ключ ● деактивироваться
предназначение	● Контейнер сущностей, которые располагаются на Ячейке

Уровень - создает Поле, умеет создавать Стены, Игрока, Точку_Выхода и Ключ(и) и размещать их на Поле в Ячейках. Позиции этих сущностей Уровень определяет несамостоятельно.

знает	● Поле ● требуемую расстановку всех сущностей на Поле, в том числе, Точки_Выхода
умеет	● создавать и размещать на Поле все сущности начальной обстановки: o Игрок o Ключ(и) o Стены o Точка_Выхода
предназначение	● Фабрика сущностей; и расстановка их на Поле в соответствии с начальной обстановкой

Игрок - умеет перемещаться на доступные соседние клетки. Попадая на ячейку с Ключом, подбирает его. Попадая на Ячейку - Выход, предоставляет информацию о наличии ключей(по необходимости)

знает	● свою позицию ● о наличии ключа(ей)
умеет	● перемещаться в соседнюю Ячейку, если они доступны для перемещения ● подбирать ключ ● предоставлять информацию о наличии ключей
предназначение	● Сущность, перемещающаяся по Полю, и стремящаяся к Точке_Выхода

Точка_Выхода - разновидность Ячейки. Сообщает игре о победе Игрока.

знает	● аналогично Ячейке ● наличие Игрока в ней
умеет	● аналогично Ячейке, но на ней не может располагаться Ключ или Стена ● предоставляет Игре данные для проверки выигрышного условия
предназначение	● Информатор Игры о состоянии игрового процесса

3.4 Структура программы на уровне классов

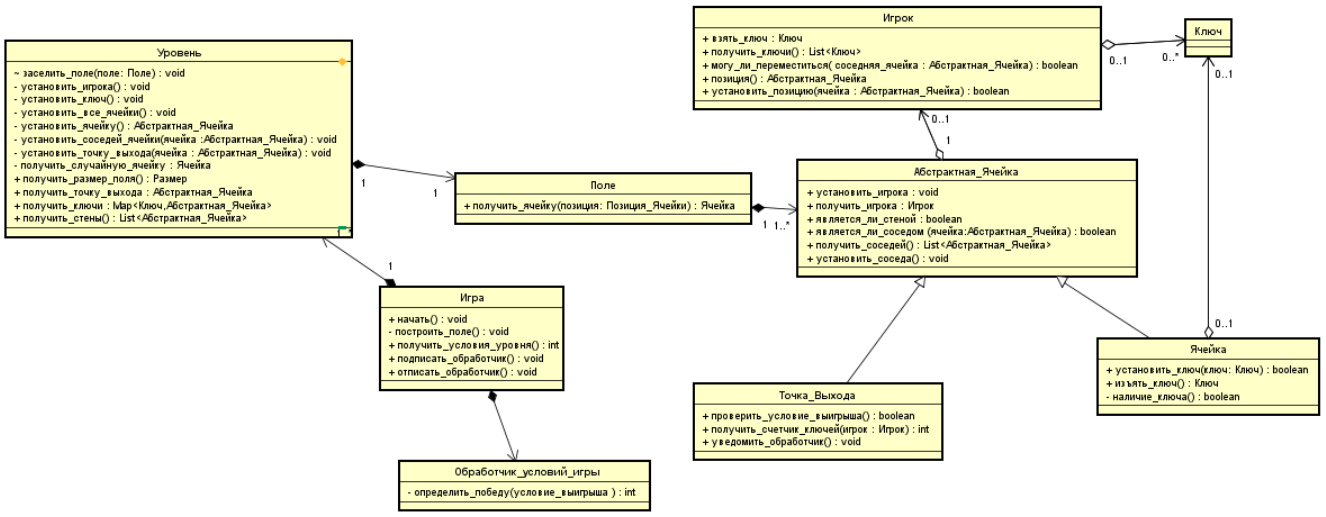

Рисунок 1 - Диаграмма классов вычислительной модели

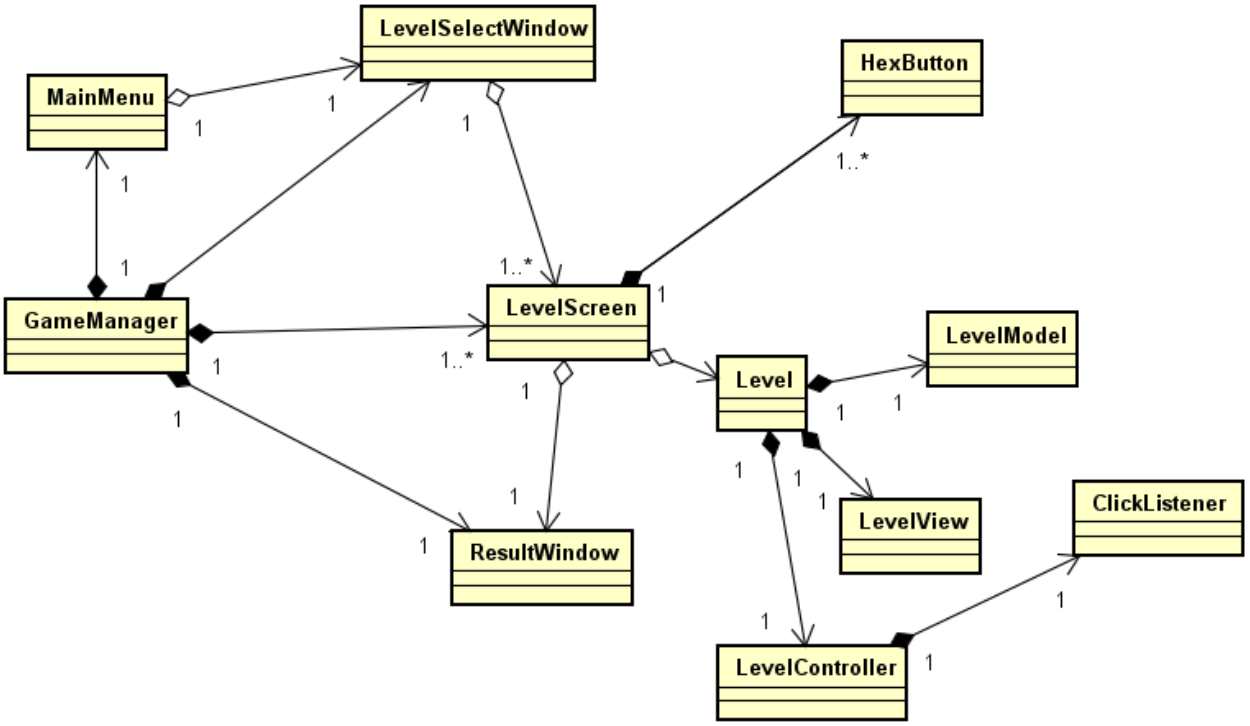

Рисунок 2 - Диаграмма классов представления

3.5 Типовые процессы в программе

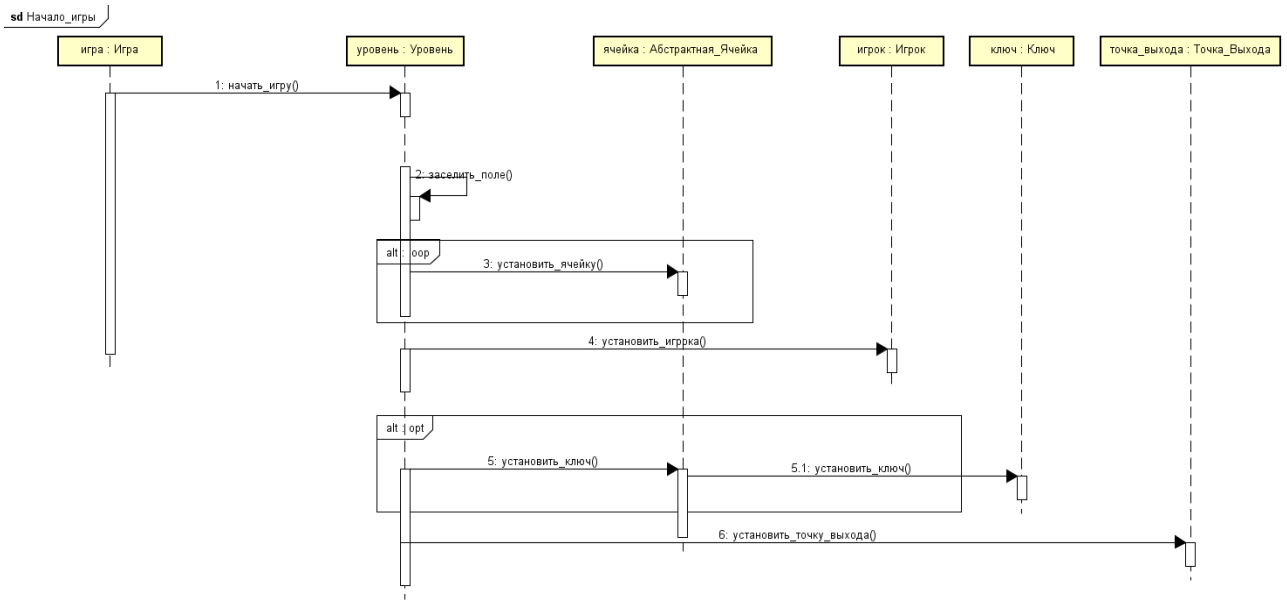

Рисунок 3 - Диаграмма последовательности для вычислительной модели. Начало игры

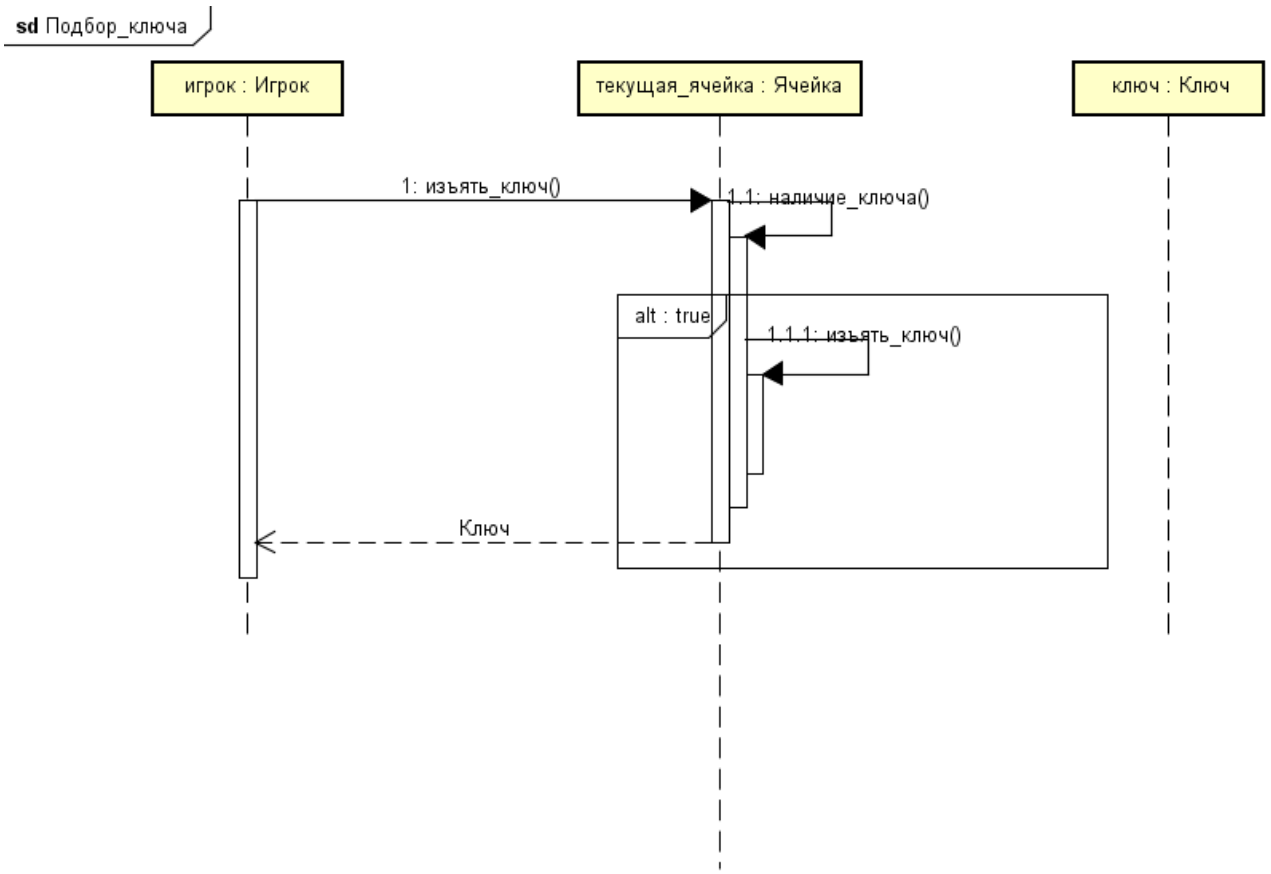

Рисунок 4 - Диаграмма последовательности для вычислительной модели. Подбор ключа

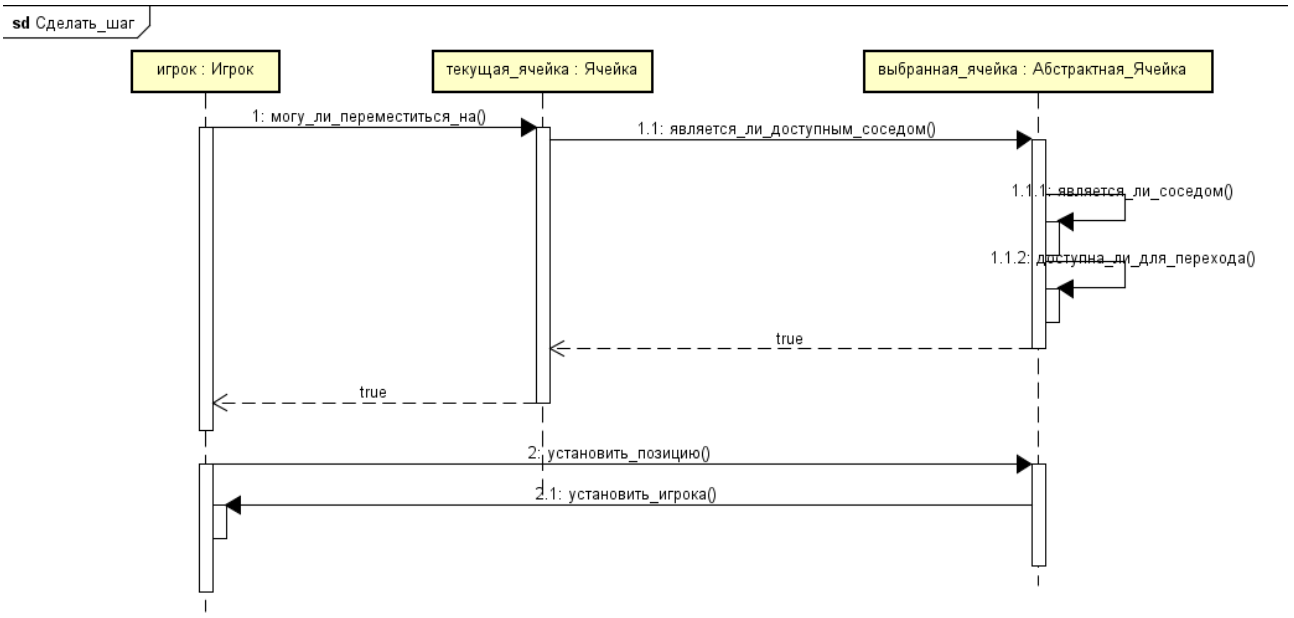

Рисунок 5 - Диаграмма последовательности для вычислительной модели. Сделать шаг

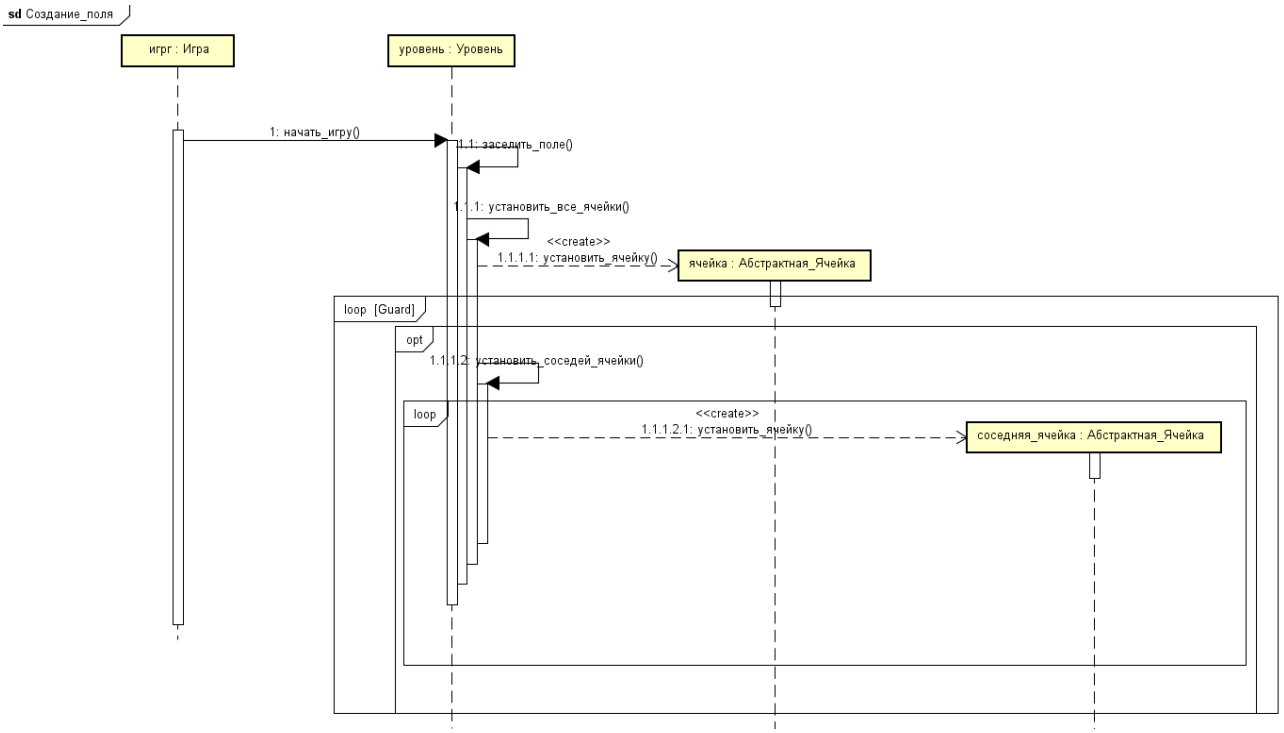

Рисунок 6 - Диаграмма последовательности для вычислительной модели. Создание поля

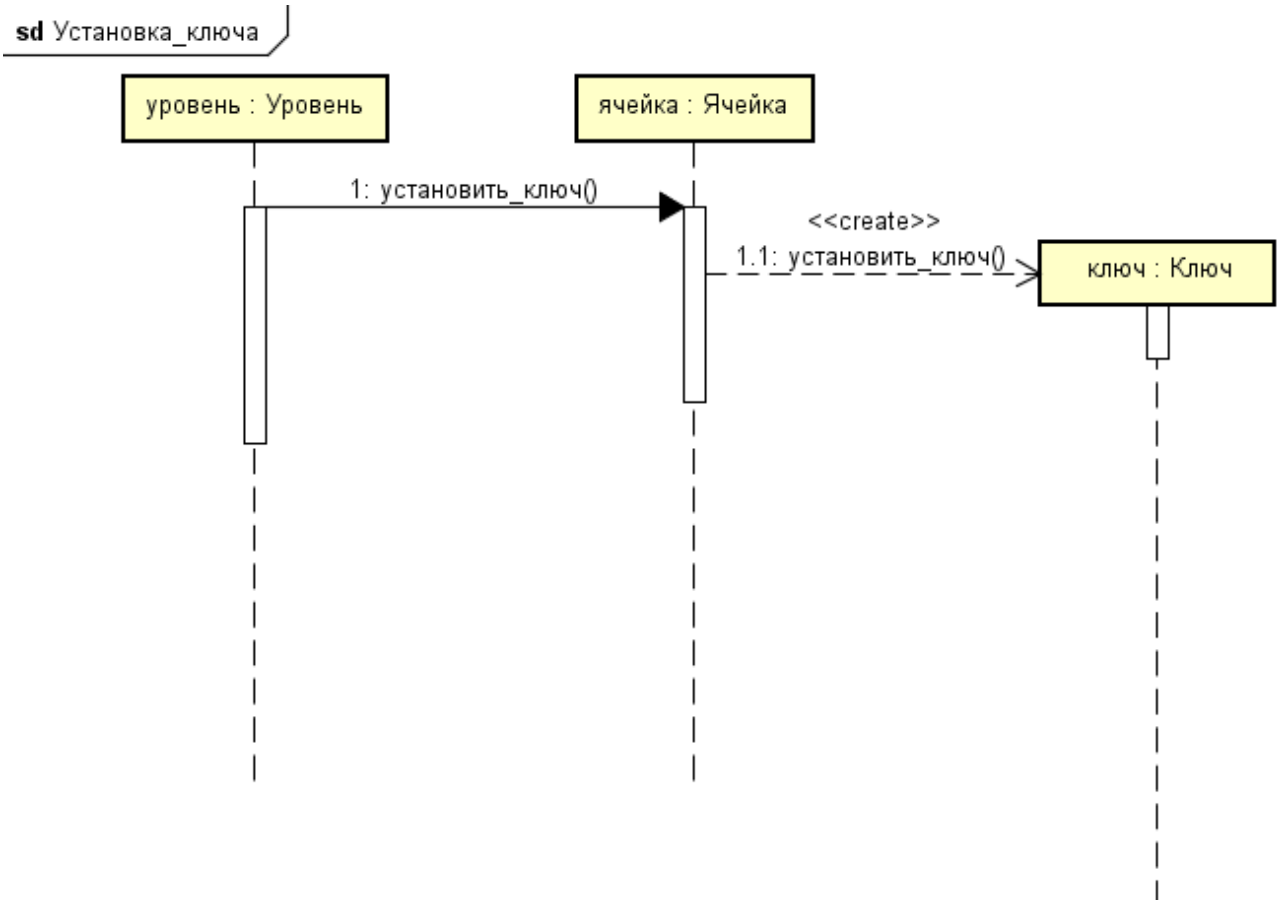

Рисунок 7 - Диаграмма последовательности для вычислительной модели. Установка ключа

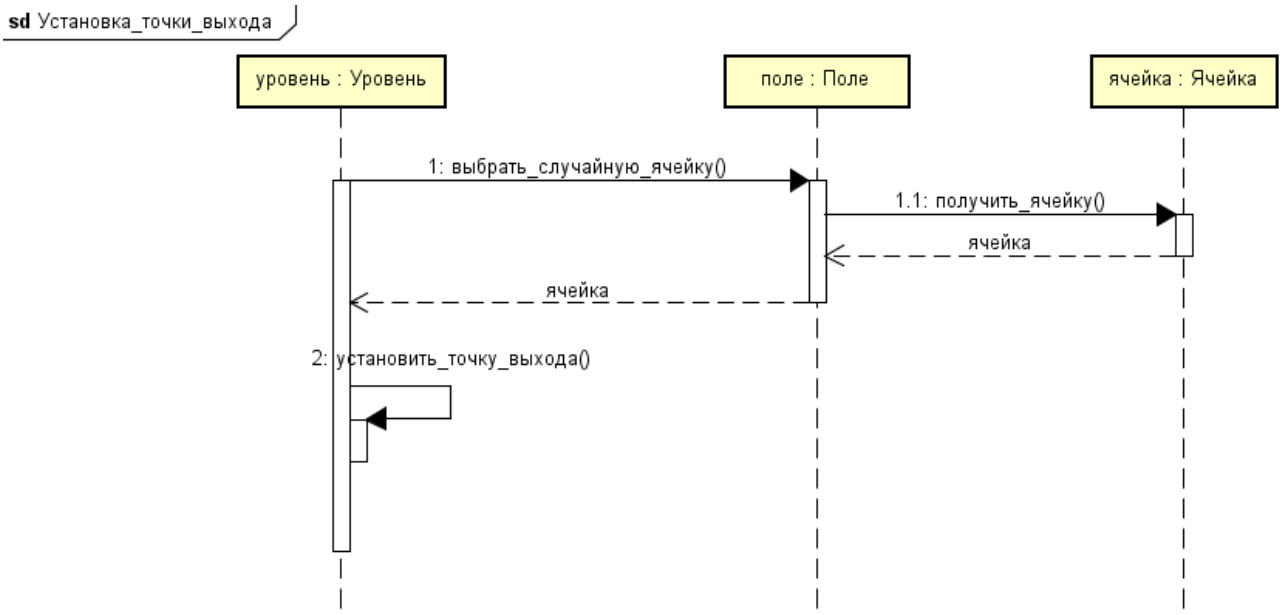

Рисунок 8 - Диаграмма последовательности для вычислительной модели. Установка точки выхода

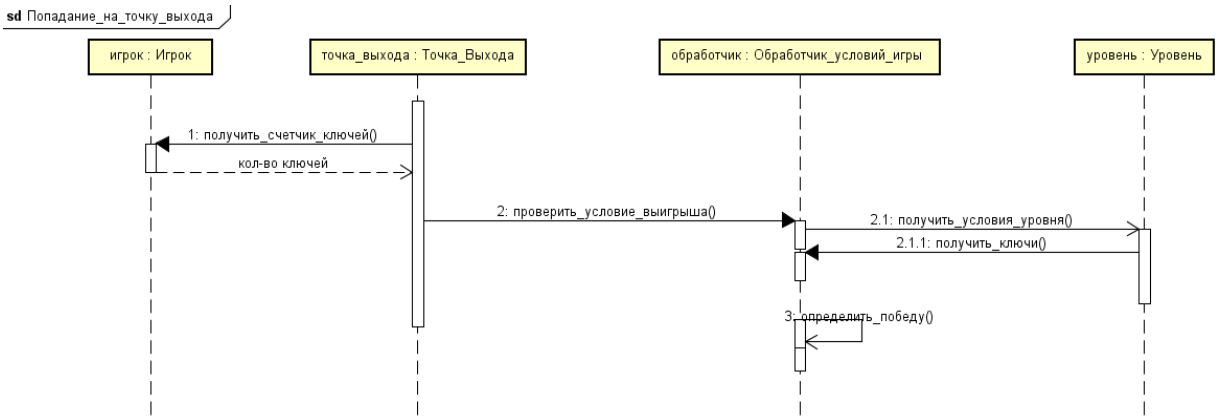

Рисунок 9 - Диаграмма последовательности для вычислительной модели. Попадание на точку выхода

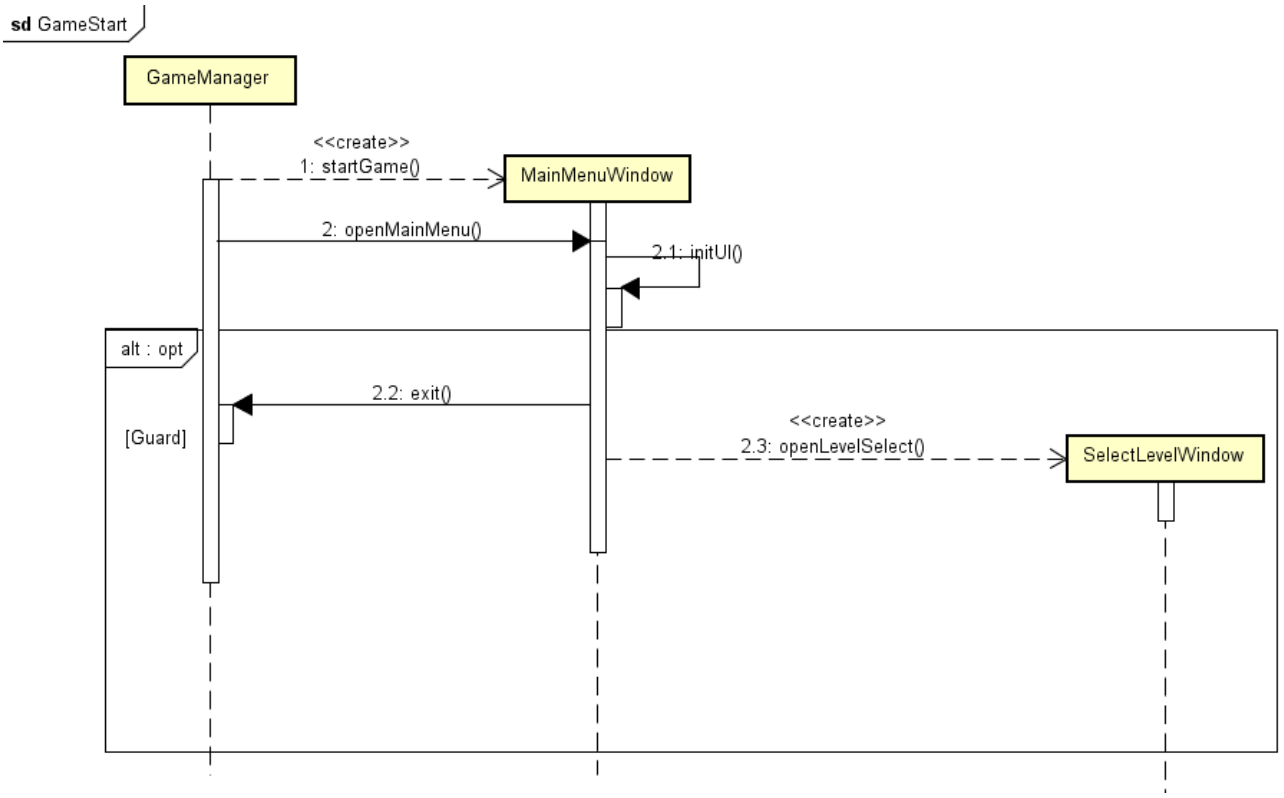

Рисунок 10 - Диаграмма последовательности с участием представления. Начало игры

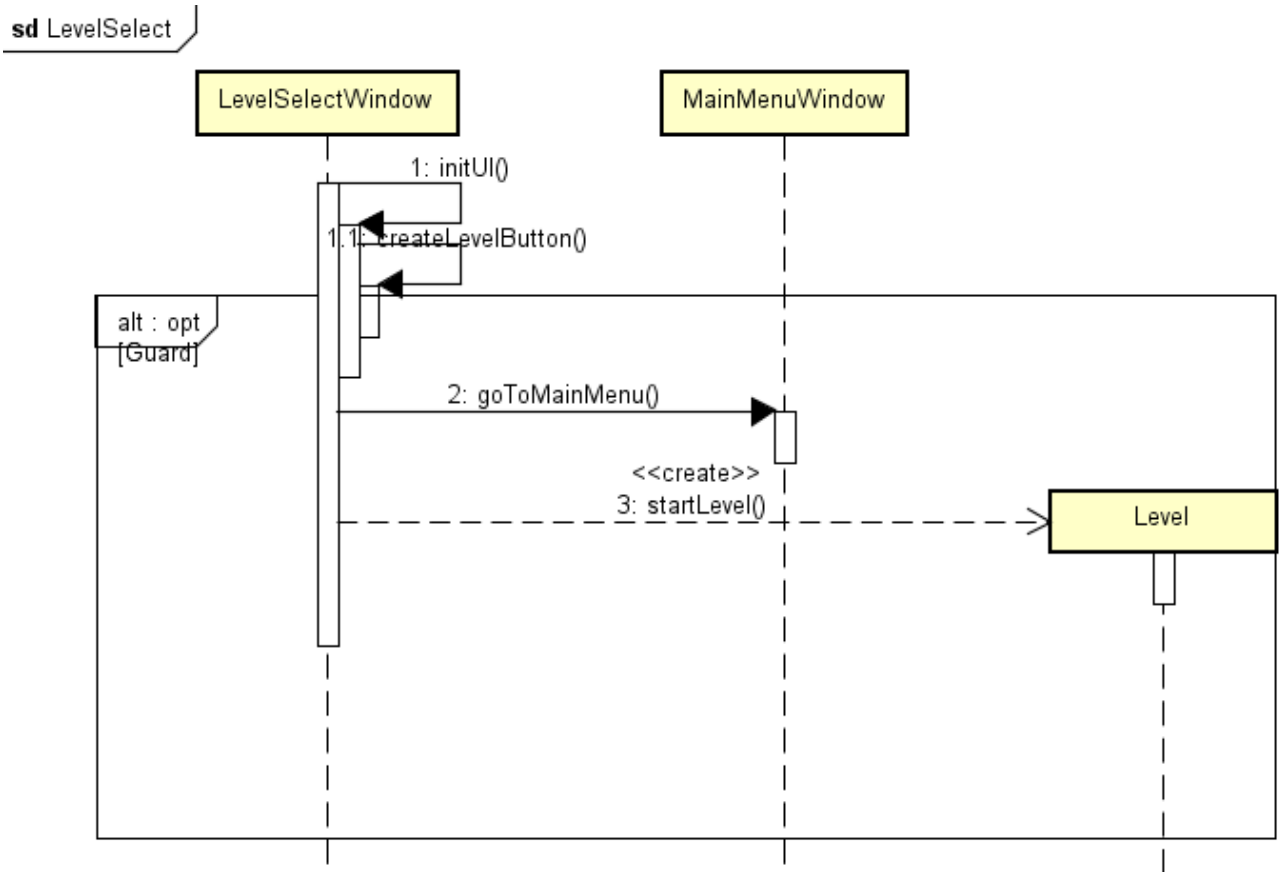

Рисунок 11 - Диаграмма последовательности с участием представления. Выбор уровня

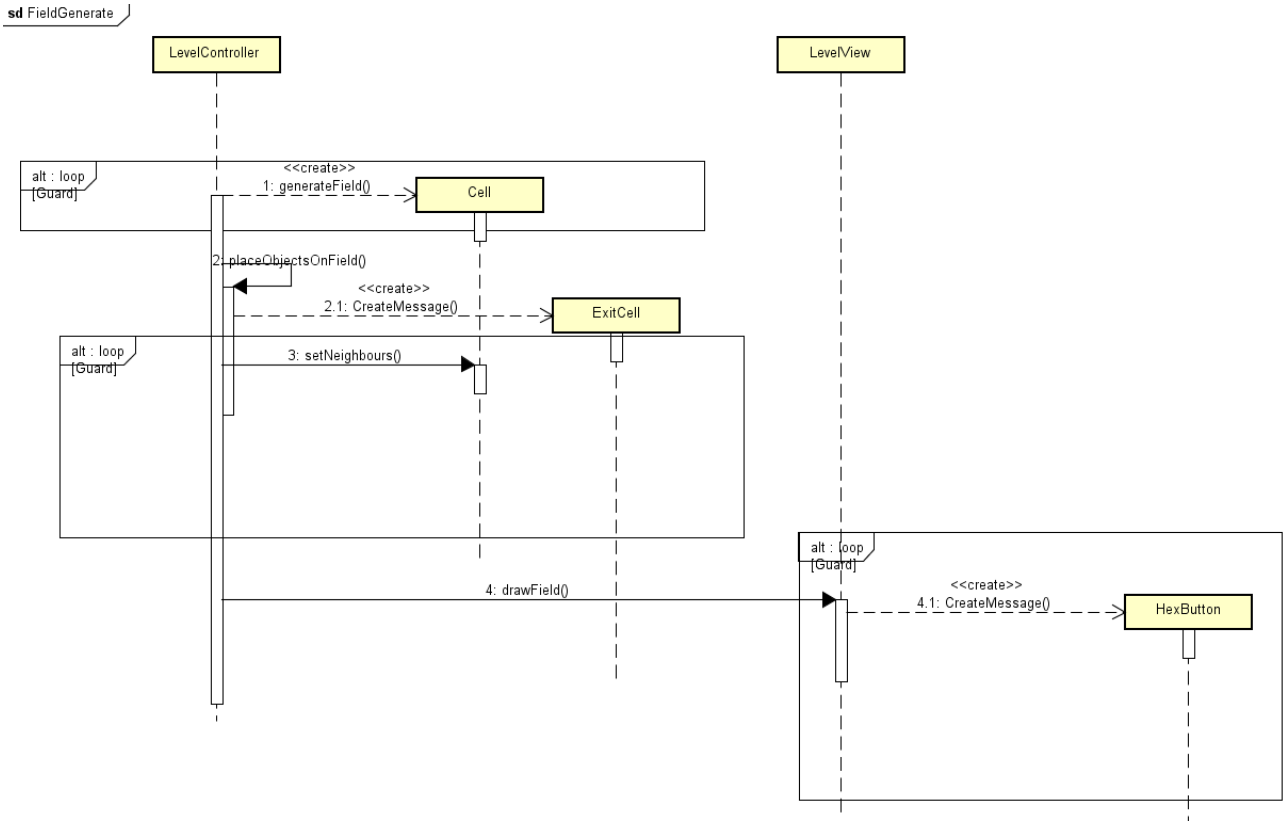

Рисунок 12 - Диаграмма последовательности с участием представления. Генерация поля

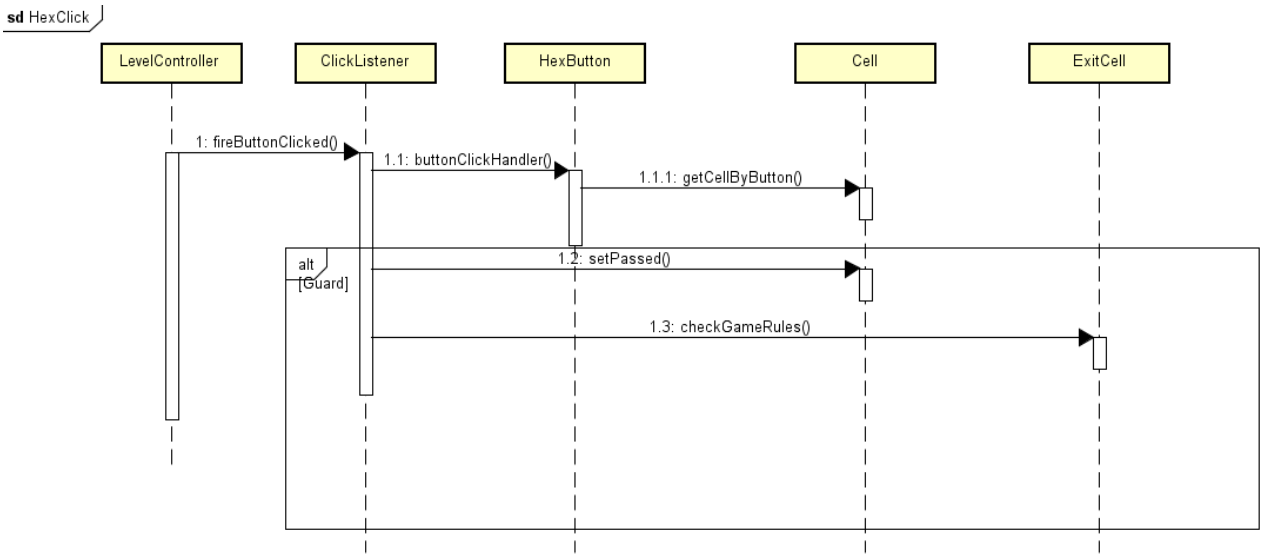

Рисунок 13 - Диаграмма последовательности с участием представления. Нажатие на клетку

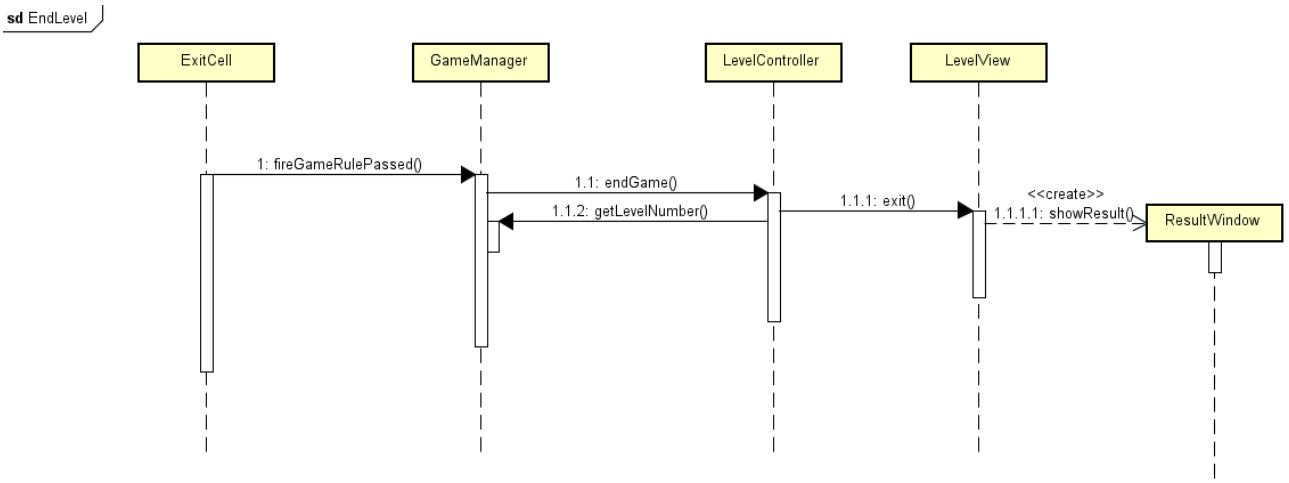

Рисунок 14 - Диаграмма последовательности с участием представления. Завершение уровня

3.6 Человеко-машинное взаимодействие

Главное меню программы состоит из двух кнопок:

- Начать игру - запускает уровень
- Выход - закрывает программу

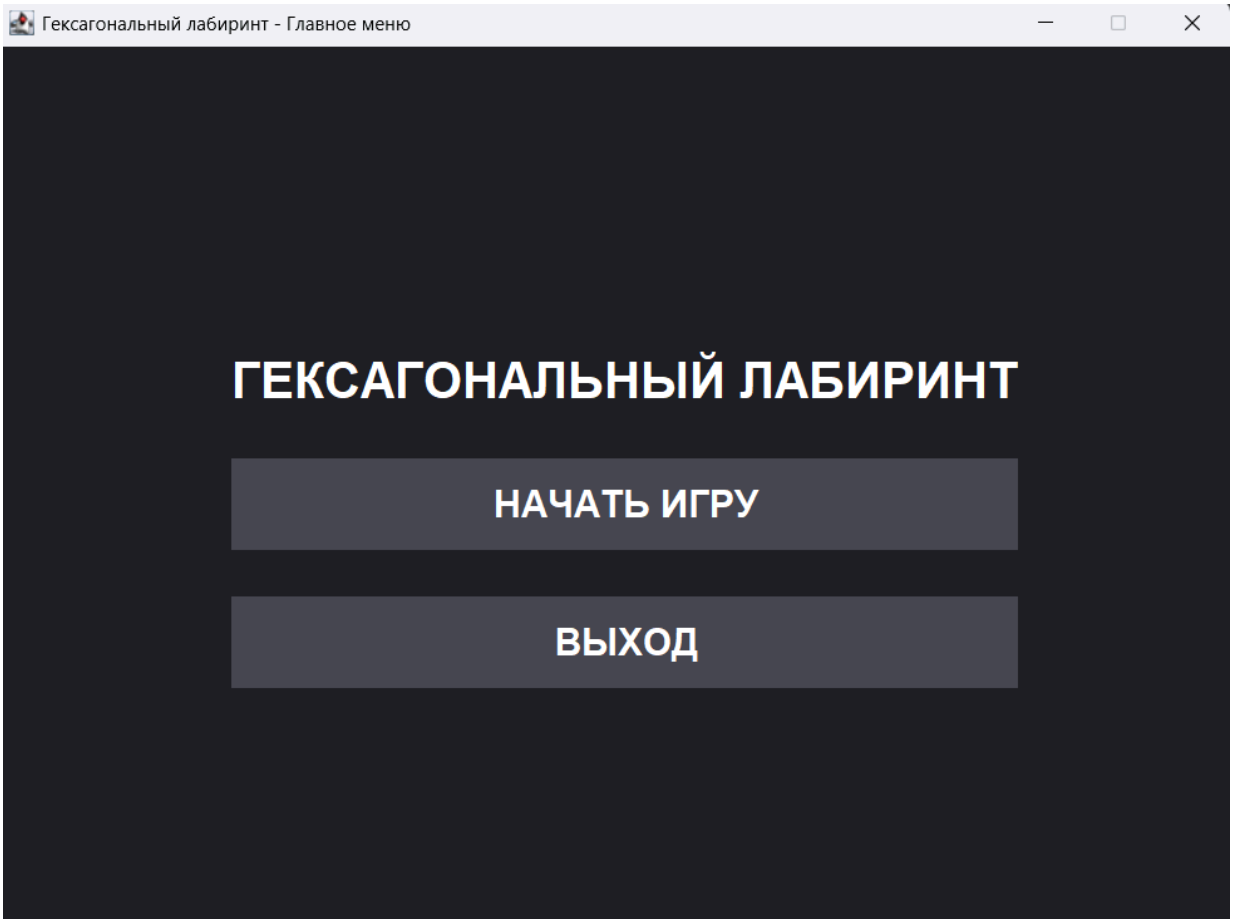

Рисунок 15 - Вид главного меню игры

Вид окна уровня представлен ниже. На нём располагается игровое поле, на котором изображена сетка из шестиугольных кнопок, состоящая из позиции игрока (красный), доступных для перемещения соседей(синий), ключей (символ “l” в оранжевой клетке) и стен(темно-серый).

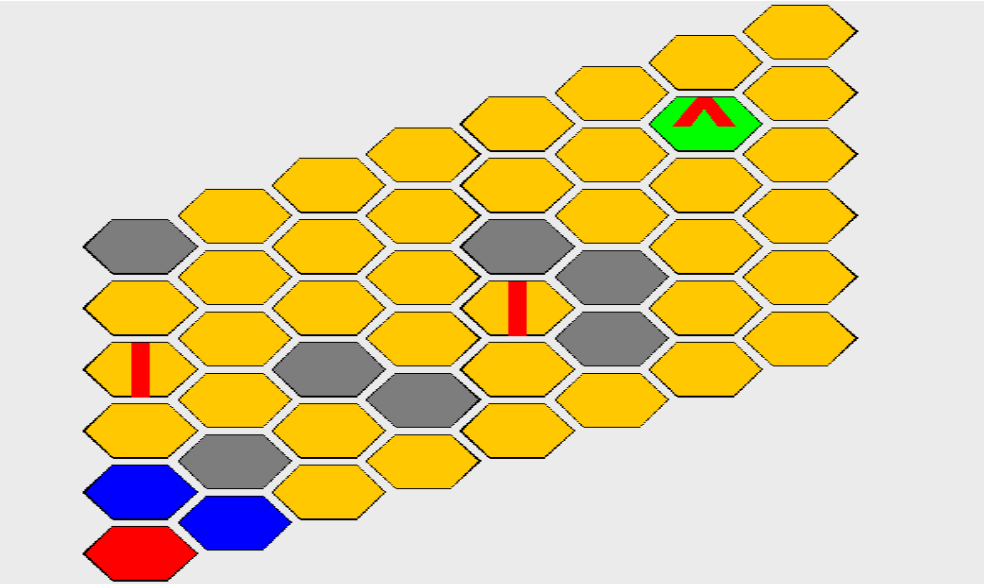

Рисунок 16 - Общий вид окна уровня

На рисунке ниже можно наблюдать изменение состояния клетку на “Пройденная”:

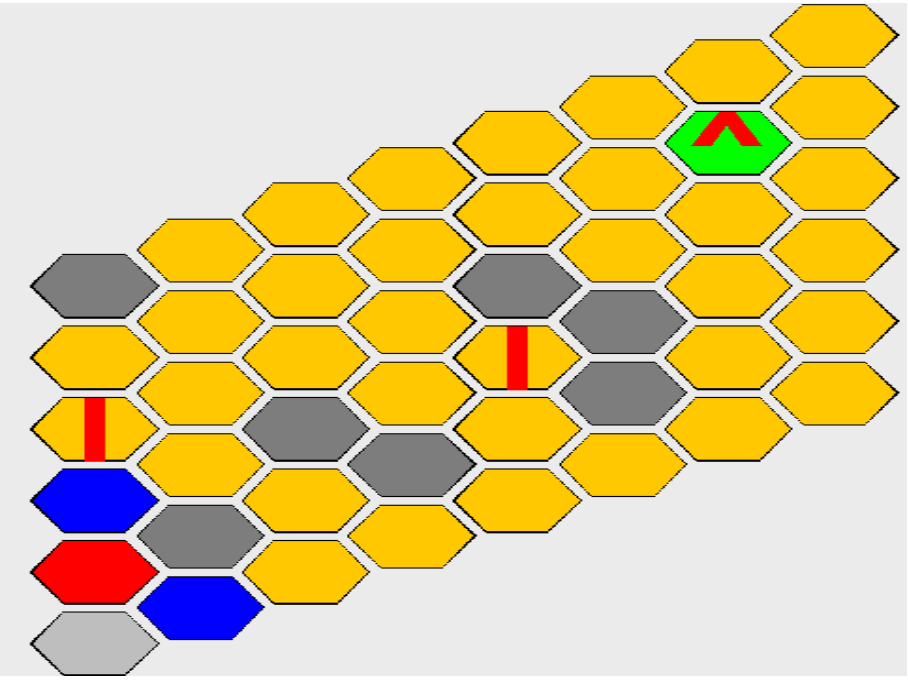

Рисунок 17 - Вид уровня после передвижения игрока

Изменение состояния клетки с ключом на “Пройденная”. Стоит отметить, что с пройденной клетки пропало “отображение” ключа, тем самым давая понять пользователю об его успешном подборе:

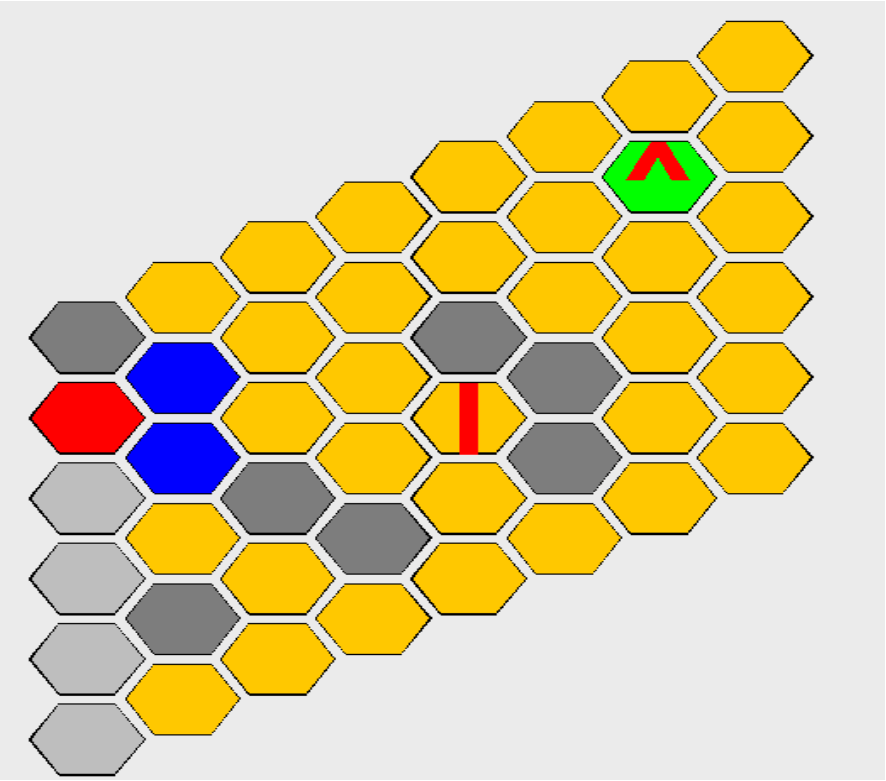

Рисунок 18 - Вид уровня после подбора ключа

В случае успешного завершения уровня, пользователю будет показано окно с результатом:

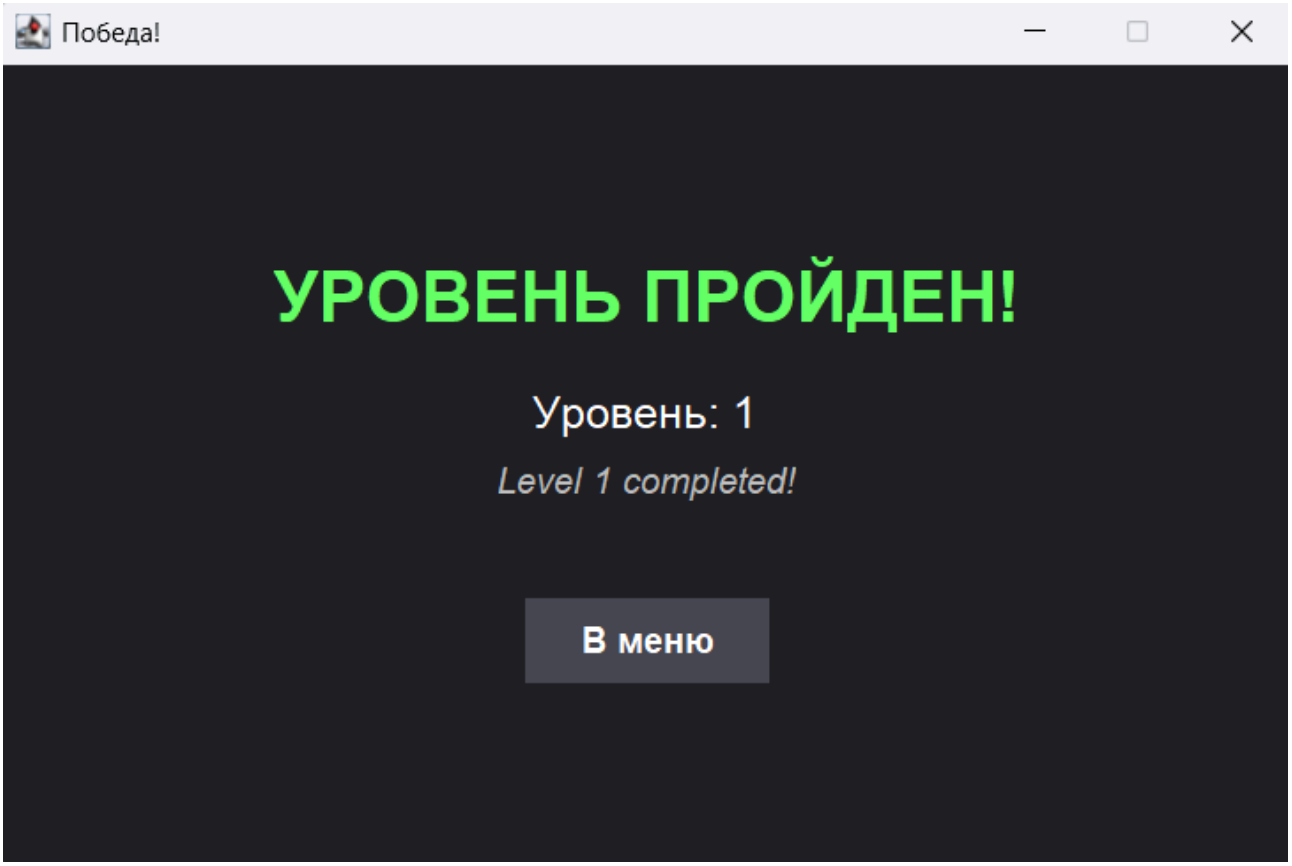

Рисунок 19 - Общий вид окна результатов

3.7 Реализация ключевых классов

```
/**
 * Класс абстрактной  ячейки
 */
public abstract class AbstractCell implements IInteractable
{

    private int q; // смещение по горизонтали
    private int r; // смещение по диагонали

    private Player _player;
    private boolean _isWall;
    private List<AbstractCell> _neighbours = new ArrayList<>();

    public AbstractCell(int q, int r) {
        this.q = q;
        this.r = r;
    }

    public AbstractCell(Point pos) {
        this.q = pos.x;
        this.r = pos.y;
    }

    public int getQ() { return q;}

    public int getR() {return r;}

    public void setQ(int q) {this.q = q;}

    public void setR(int r) {this.r = r;}


    public void SetPlayer(Player player){ _player = player; }

    public void unsetPlayer(){ _player = null; }

    public Player GetPlayer(){ return _player; }

    public void SetNeighbour(AbstractCell neighbour)
    {
        if (!GetNeighbours().contains(neighbour))
        {
            _neighbours.add(neighbour);
            neighbour.SetNeighbour(this);
        }
    }

    public boolean shouldEnableCell() {
        return this instanceof ExitCell || this instanceof TeleportCell ||
                (this instanceof Cell cell && !cell.getPassedInfo());
    }

    public List<AbstractCell> GetNeighbours(){ return _neighbours; }

    public boolean IsNeighbourOf(AbstractCell cell) { return _neighbours.contains(cell); }

    public abstract boolean handlePlayerInteraction(Player player, LevelModel model);
```

```
    public abstract void getButtonAppearance(HexButton btn);
}


/**
 * Класс игрока
 */
public class Player
{
    private AbstractCell _cell;

    private List<Key> _keys = new ArrayList<Key>();

    public void SetCell(AbstractCell cell)
    {
        _cell = cell;
        _cell.SetPlayer(this);
    }

    public AbstractCell GetCell(){ return _cell; }

    public void TakeKeyFromCell(Cell cell)
    {
        if (cell.GetKey()!=null)
        {
            _keys.add(cell.GetKey());
            cell.DeleteKey();
        }
    }

    public List<Key> GetKeys() { return _keys; }

    public void moveTo(AbstractCell cell)
    {
        if (_cell!=null) _cell.unsetPlayer();
        SetCell(cell);
        cell.SetPlayer(this);
    }


    public void handleRegularCell(Cell cell)
    {
        TakeKeyFromCell(cell);
        cell.setPassed();
    }
}


/**
 * Класс обрабатывающий процесс игры
 */
public class LevelController implements ILevelInputHandler {
    private final LevelModel model;
    private final LevelView view;
    private final GameManager gameManager;

    public LevelController(LevelModel model, LevelView view, GameManager gameManager) {
        this.model = model;
        this.view = view;
        this.gameManager = gameManager;
        setupExitCellListeners();
```

```
        handleCellClick(view.getStartButton());
    }

    private void setupExitCellListeners() {
        model.getField().stream()
                .filter(cell -> cell instanceof ExitCell)
                .forEach(cell -> {
                    ((ExitCell)cell).addExitCellActionListener(
                            gameManager.getExitCellObserver()
                    );
                });
    }

    @Override
    public void handleCellClick(HexButton btn) {
        AbstractCell cell = view.getCellByButton(btn);

        Player player = model.getPlayer();

        boolean win = researchCell(cell, player);
        if (win) view.close();

        view.update(player);
    }


    public boolean researchCell(AbstractCell cell, Player player) {
        player.moveTo(cell);
        return cell.handlePlayerInteraction(player, model);
    }
}


/**
 * Класс отображения шестиугольной клетки
 */
public class HexButton extends JButton {
    Polygon bounds;
    Character character;
    AbstractCell _cell;

    public HexButton(Character character) {
        this.calculateBounds();
        this.setBackground(Color.YELLOW);
        this.setForeground(Color.RED);
        this.character = character;
        this.clicked = false;
        this.setOpaque(true);
        this.setBorderPainted(false);
        this.setContentAreaFilled(false);
    }

    private Polygon hexagon(int width, int height, double ratio) {
        Polygon hexagon = new Polygon();
        for (int i = 0; i < 6; i++) {
            int x = width / 2 + (int)((width - 2) / 2 * Math.cos(i * 2 * Math.PI / 6) * ratio);
            int y = height / 2 + (int)((height - 2) / 2 * Math.sin(i * 2 * Math.PI / 6) * ratio);
            hexagon.addPoint(x,y);
        }
        return hexagon;
    }

    private void calculateBounds() {
```

```
        this.bounds = this.hexagon(this.getWidth(), this.getHeight(), 1.0);
    }

    @Override
    public boolean contains(Point p) {
        return this.bounds.contains(p);
    }

    @Override
    public boolean contains(int x, int y) {
        return this.bounds.contains(x, y);
    }

    @Override
    public void setSize(Dimension d) {
        super.setSize(d);
        this.calculateBounds();
    }

    @Override
    public void setSize(int w, int h) {
        super.setSize(w, h);
        this.calculateBounds();
    }

    @Override
    public void setBounds(int x, int y, int width, int height) {
        super.setBounds(x, y, width, height);
        this.calculateBounds();
    }

    @Override
    public void setBounds(Rectangle r) {
        super.setBounds(r);
        this.calculateBounds();
    }

    @Override
    protected void paintComponent(Graphics graphics) {

        graphics.setColor(Color.BLACK);
        Polygon stroke = this.hexagon(getWidth(), getHeight(), 0.95);
        graphics.drawPolygon(stroke);
        graphics.fillPolygon(stroke);

        Polygon inside = this.hexagon(getWidth(), getHeight(), 0.9);
        graphics.setColor(getBackground());
        graphics.drawPolygon(inside);
        graphics.fillPolygon(inside);

        Font font = new Font("Arial", Font.BOLD, 64);
        graphics.setFont(font);
        graphics.setColor(getForeground());

        FontMetrics fontMetrics = getFontMetrics( font );
        int width = fontMetrics.stringWidth(this.character + "");
        int height = fontMetrics.getHeight();

        graphics.drawString(this.character + "", (getWidth() - width) / 2 , (getHeight() + height
- 25) / 2);

    }
```

```
    public AbstractCell cell(){ return _cell;}

    public void setCell(AbstractCell cell){ _cell = cell; }

    public void setCharacter(char ch) { character = ch; }

    public char getCharacter() {  return character;}
}
```

3.8 Реализация ключевых тестовых случаев

```
/**
* Тесты механики передвижения игрока
*/
public class MovementTest {

    @Test
    public void testBasicMovement() {
        Cell startCell = new Cell(0, 0);
        Cell targetCell = new Cell(0, 1);
        startCell.SetNeighbour(targetCell);

        player.moveTo(startCell);
        player.moveTo(targetCell);

        assertEquals(targetCell, player.GetCell());
        assertNull(startCell.GetPlayer());
        assertEquals(player, targetCell.GetPlayer());
    }

    @Test
    public void testWallBlocking() {
        Cell startCell = new Cell(0, 0);
        Wall wallCell = new Wall(0, 1);
        startCell.SetNeighbour(wallCell);

        player.moveTo(startCell);
        boolean result = wallCell.handlePlayerInteraction(player, model);

        assertFalse(result);
        assertEquals(startCell, player.GetCell());
    }

    @Test
    public void testExitCellWithoutKeys() {
        List<Key> requiredKeys = new ArrayList<>();
        requiredKeys.add(new Key());

        ExitCell exitCell = new ExitCell(requiredKeys, new Cell(1, 1));
        // Игрок НЕ собрал ключи

        boolean result = exitCell.handlePlayerInteraction(player, model);
        assertFalse(result); // Уровень не должен завершиться
    }

    @Test
    public void testCellPassedState() {
        Cell cell = new Cell(2, 2);
```

```
        player.moveTo(cell);
        cell.handlePlayerInteraction(player, model);

        assertTrue(cell.getPassedInfo());
    }

    @Test
    public void testNeighborConnections() {
        Cell cell1 = new Cell(0, 0);
        Cell cell2 = new Cell(0, 1);

        cell1.SetNeighbour(cell2);

        assertTrue(cell1.IsNeighbourOf(cell2));
        assertTrue(cell2.IsNeighbourOf(cell1));
        assertEquals(1, cell1.GetNeighbours().size());
        assertEquals(1, cell2.GetNeighbours().size());
    }
}



/**
* Тесты механики подбора ключа
*/
public class PickupTest
{
    @Test
    public void testPickupKeyFromCell() {
        player.moveTo(cellWithKey);
        cellWithKey.handlePlayerInteraction(player, null);

        assertTrue("Ключ должен быть у игрока", player.GetKeys().contains(testKey));
        assertNull("Ключ должен быть удален из клетки", cellWithKey.GetKey());
    }

    @Test
    public void testPickupFromEmptyCell() {
        player.moveTo(emptyCell);
        emptyCell.handlePlayerInteraction(player, null);

        assertTrue("У игрока не должно быть ключей", player.GetKeys().isEmpty());
    }

    @Test
    public void testCellMarkedAsPassedAfterPickup() {
        player.moveTo(cellWithKey);
        cellWithKey.handlePlayerInteraction(player, null);

        assertTrue("Клетка должна быть помечена как пройденная", cellWithKey.getPassedInfo());
    }

    @Test
    public void testMultipleKeyPickups() {
        Key secondKey = new Key();
        Cell secondCellWithKey = new Cell(2, 2);
        secondCellWithKey.SetKey(secondKey);

        player.moveTo(cellWithKey);
        cellWithKey.handlePlayerInteraction(player, null);

        player.moveTo(secondCellWithKey);
```

```
        secondCellWithKey.handlePlayerInteraction(player, null);

        assertEquals("У игрока должно быть 2 ключа", 2, player.GetKeys().size());
        assertTrue("Должен быть первый ключ", player.GetKeys().contains(testKey));
        assertTrue("Должен быть второй ключ", player.GetKeys().contains(secondKey));
    }


    @Test
    public void testKeyRemovalFromCell() {
        player.moveTo(cellWithKey);
        cellWithKey.handlePlayerInteraction(player, null);

        assertFalse("Клетка должна вернуть false при попытке удалить несуществующий ключ",
                cellWithKey.DeleteKey());
    }

    @Test
    public void testKeySetToOccupiedCell() {
        cellWithKey.SetKey(testKey);
        Key newKey = new Key();

        boolean result = cellWithKey.SetKey(newKey);

        assertFalse("Нельзя установить ключ в клетку, где уже есть ключ", result);
        assertEquals("В клетке должен остаться старый ключ", testKey, cellWithKey.GetKey());
    }

    @Test
    public void testKeyEquality() {
        Key key1 = new Key();
        Key key2 = key1;

        assertTrue("Ключи должны быть равны между собой", key1.equals(key2));
        assertTrue("Равенство должно быть симметричным", key2.equals(key1));
    }
}
```

4. Вторая итерация разработки

4.1 Функциональные требования (сценарии)

1. Сценарий "Подвижная клетка со следом"
1. Игрок перемещается на TeleportCell
2. TeleportCell:
   - Вызывает стандартное поведение (оставляет след)
   - Получает доступного для перемещения соседа (getFirstEnableNeighbour())
   - Если такой находится
     * Меняется местами с выбранным соседом (swapWith(neighbour))
3. LevelView обновляет отображение обеих клеток
4. Игрок остается на новой позиции клетки

2. Сценарий "Подвижная клетка с выходом"
1. Игрок перемещается на ExitCell
2. ExitCell проверяет:
   - Если все ключи собраны:
     * Уведомляет GameManager о победе
   - Если ключи не собраны:
     * ExitCell получает первого доступного для перемещения соседа
     * ExitCell меняется местами с выбранным соседом
     * Игрок остается на старой позиции ExitCell
3. LevelView обновляет отображение

4.2 Словарь предметной области

Термин	Описание
GameManager	Управляет игровым процессом, переходами между уровнями и окнами. Загружает конфигурацию уровней (JSON/XML), создает уровни, обрабатывает победу/поражение.
LevelView	Отвечает за отображение игрового поля и UI (цвета клеток: синие — активные, оранжевые — неактивные, серые — стены, зеленые — выход). Обновляет визуальное состояние после действий игрока.
ExitCell	Клетка выхода (зеленая). Проверяет наличие всех ключей у игрока. Если ключи не собраны — меняется местами с соседней клеткой. Уведомляет GameManager о победе.
TeleportCell	Вариативная клетка. При попадании игрока телепортирует его в случайную позицию и меняется местами с соседней клеткой (если доступно).

4.3 Структура программы на уровне классов

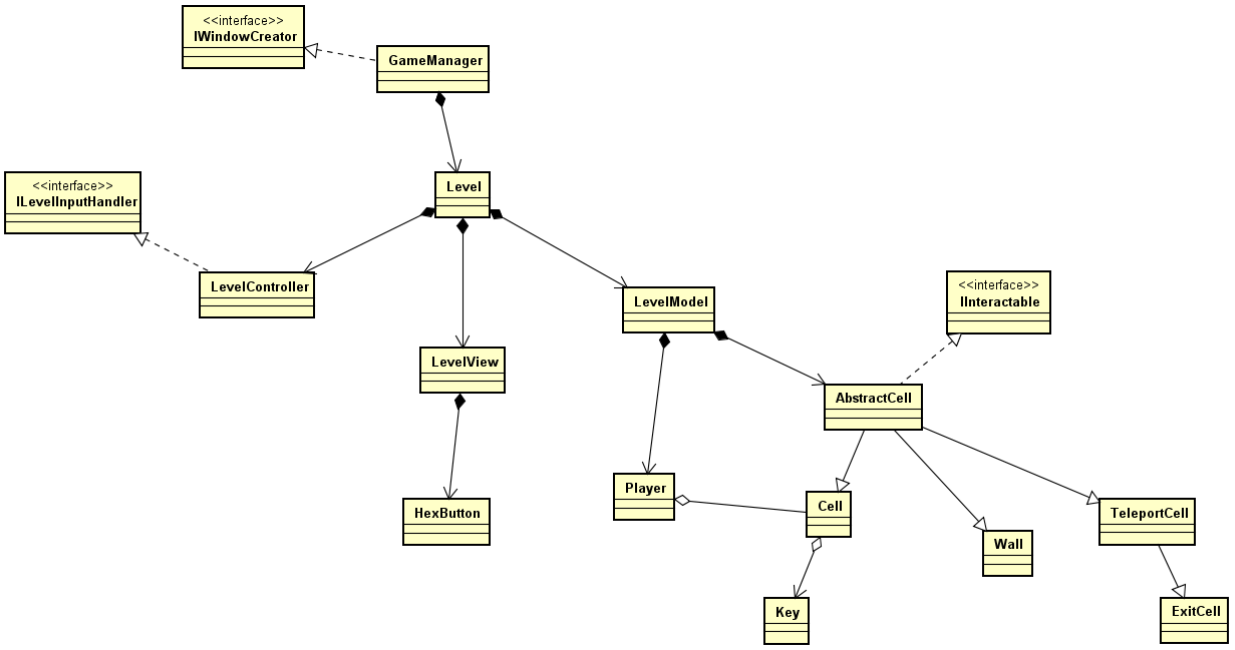

Рисунок 20 - Диаграмма классов полной вычислительной модели

3.5 Типовые процессы в программе

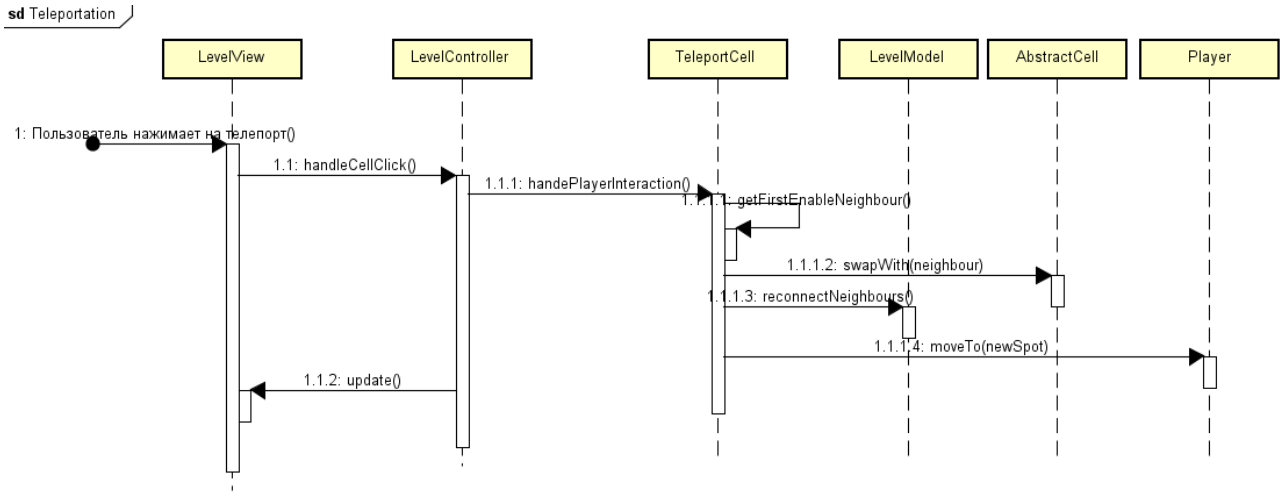

Рисунок 21 - Диаграмма последовательности для полной вычислительной модели. Телепортация

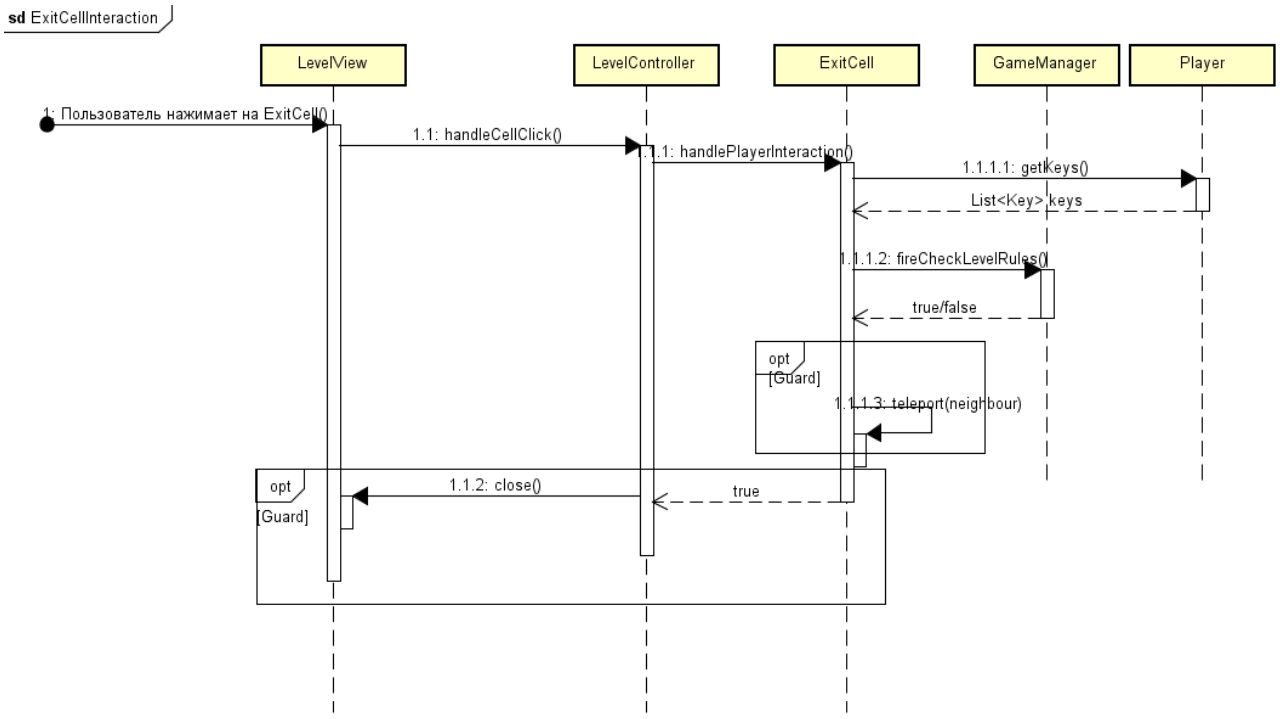

Рисунок 22 - Диаграмма последовательности для полной вычислительной модели. Попадание на клетку выхода

4.5 Человеко-машинное взаимодействие

Окно уровня второй итерации. Стоит отметить, что на поле появилась новая клетка (голубая) - это телепорт:

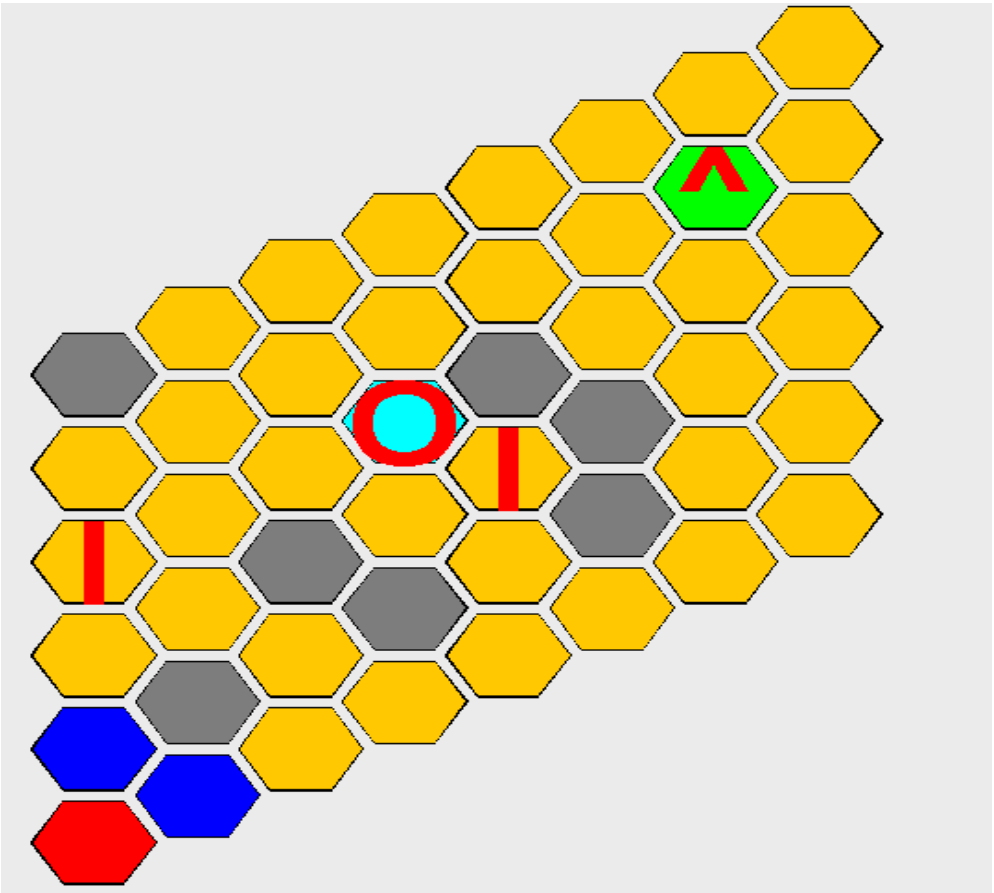

Рисунок 23 - Окно уровня после второй итерации

Момент телепортации: ячейка-телепорт вместе с игроком перемещается на позицию свободного соседа:

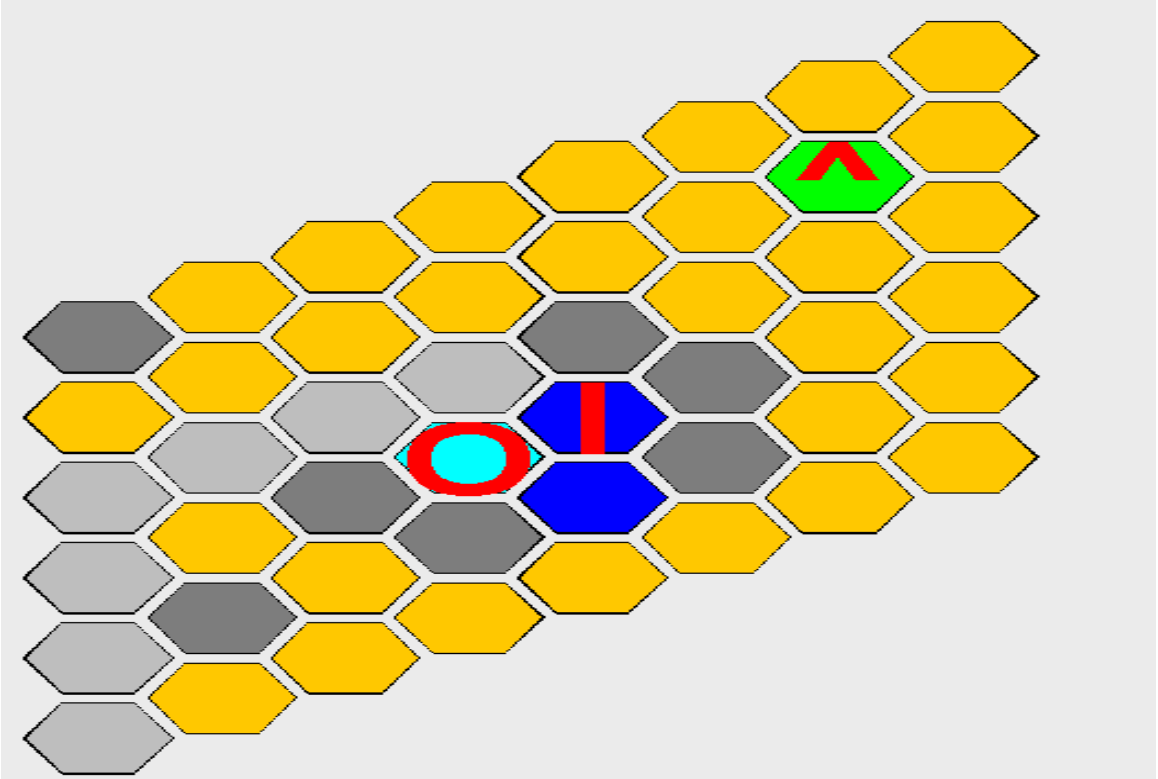

Рисунок 24 - Попадание на клетку-телепорт

Случай невыполнения правил уровня при попадании на ячейку-выход: данная ячейка перемещается на позицию свободного соседа без игрока:

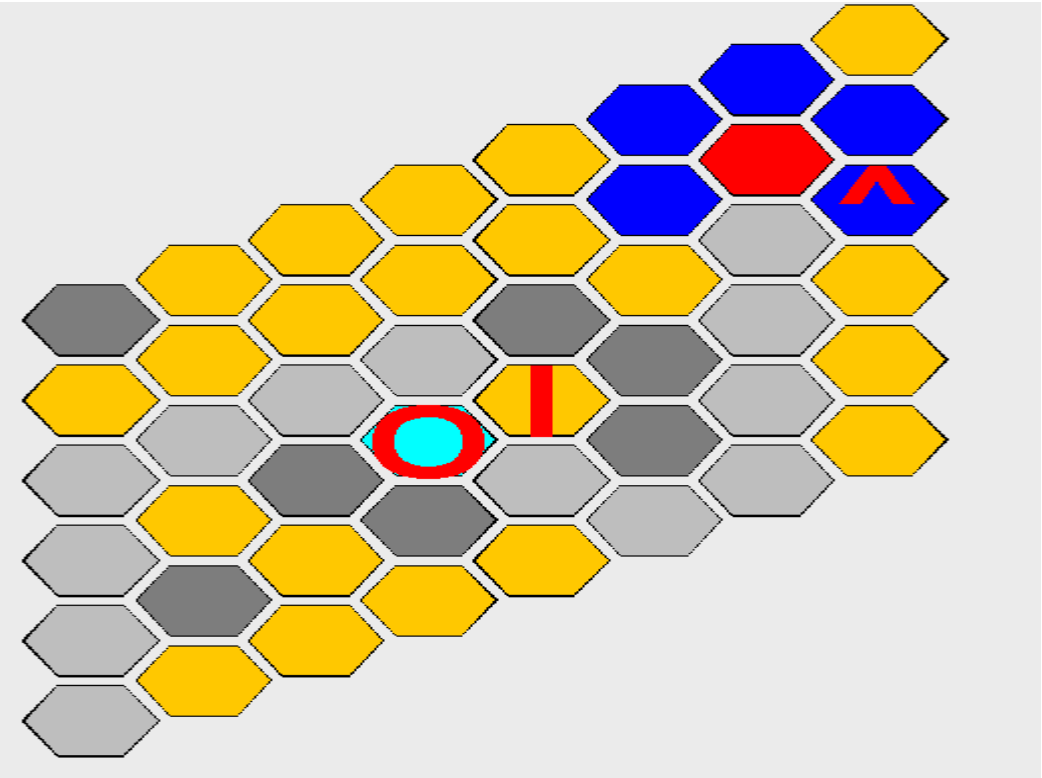

Рисунок 25 - Попадание на клетку-выход без всех собранных ключей

4.6 Реализация ключевых классов

```
/**
* Класс ячейки-телепорт
*/
public class TeleportCell extends AbstractCell
{

    public TeleportCell(int q, int r) { super(q, r); }

    public TeleportCell(AbstractCell cell) { super(cell.getQ(), cell.getR()); }

    public TeleportCell(Point pos) { super(pos); }

    public void swapWith(AbstractCell cellToSwap)
    {
        int tempQ= getQ();
        int tempR = getR();

        setQ(cellToSwap.getQ());
        setR(cellToSwap.getR());

        cellToSwap.setQ(tempQ);
        cellToSwap.setR(tempR);
    }

    @Override
    public boolean handlePlayerInteraction(Player player, LevelModel model) {
        AbstractCell newSpot = teleportToEnableNeighbour(model);
        if (newSpot!=null)
        {
            newSpot.handlePlayerInteraction(player, model);
        }
        return false;
    }

    @Override
    public void getButtonAppearance(HexButton btn) {
        btn.setBackground(Color.CYAN);
        btn.setCharacter('O');
    }

    private AbstractCell getFirstEnableNeighbour()
    {
        for (AbstractCell neighbour : GetNeighbours()) {
            if (neighbour.shouldEnableCell()) {
                return neighbour;
            }
        }
        return null;
    }

    protected AbstractCell teleportToEnableNeighbour(LevelModel model)
    {
        AbstractCell activeNeighbour = getFirstEnableNeighbour();
        if (activeNeighbour!=null)
        {
            this.swapWith(activeNeighbour);
```

```
            model.reconnectNeighbours(activeNeighbour);
            model.reconnectNeighbours(this  );
        }
        return activeNeighbour;
    }
}


/**
* Класс ячейки-выход
*/
public class ExitCell extends TeleportCell
{
    private List<Key> _levelKeys;

    public ExitCell(List<Key> levelKeys, AbstractCell cell)
    {
        super(cell.getQ(), cell.getR());
        _levelKeys = levelKeys;
    }


    // -------------------- События --------------------

    private ArrayList<IExitCellActionListener> exitCellListener = new ArrayList<>();

    public ExitCell(List<Key> keys, Point pos)
    {
        super(pos);
        _levelKeys = keys;
    }

    public void addExitCellActionListener(IExitCellActionListener listener) {
        exitCellListener.add(listener);
    }

    public void removeExitCellActionListener(IExitCellActionListener listener) {
        exitCellListener.remove(listener);
    }

    public boolean fireCheckLevelRules(List<Key> playerKeys) {
        boolean win = false;
        for(IExitCellActionListener listener: exitCellListener) {
            ExitCellActionEvent event = new ExitCellActionEvent(listener);
            event.setLevelKeys(_levelKeys);
            event.setCollectedKeys(playerKeys);
            win = listener.checkLevelRules(event);
        }
        return win;
    }

    @Override
    public void getButtonAppearance(HexButton btn) {
        btn.setBackground(Color.GREEN);
        btn.setCharacter('^');
    }


    @Override
    public boolean handlePlayerInteraction(Player player, LevelModel model) {
        boolean win = fireCheckLevelRules(player.GetKeys());
        if (win) return true;
        else
```

```
        {
            AbstractCell newSpot = teleportToEnableNeighbour(model);
            if (newSpot!=null)
            {
                player.moveTo(newSpot);
                newSpot.handlePlayerInteraction(player, model);
            }
        }
        return false;
    }
}
```

4.7 Реализация ключевых тестовых случаев

```
/**
* Тесты для ячейки-телепорта
*/
public class MovementTest {
    @Test
    public void testTeleportToWall() {
        TeleportCell teleportCell = new TeleportCell(1, 1);
        Wall wallCell = new Wall(1, 2);
        teleportCell.SetNeighbour(wallCell);

        player.moveTo(teleportCell);
        boolean result = teleportCell.handlePlayerInteraction(player, model);

        assertFalse(result);
        // Игрок должен остаться на телепорте, так как стена непроходима
        assertEquals(teleportCell, player.GetCell());
    }

    @Test
    public void testExitCellTeleportation() {
        List<Key> keys = new ArrayList<>();
        ExitCell exitCell = new ExitCell(keys, new Cell(1, 1));
        Cell neighborCell = new Cell(1, 2);
        exitCell.SetNeighbour(neighborCell);

        player.moveTo(exitCell);
        exitCell.handlePlayerInteraction(player, model);

        // При отсутствии ключей игрок должен телепортироваться
        assertEquals(neighborCell, player.GetCell());
    }
}

/**
* Тесты для ячейки-выхода
*/
public class ExitTest {
    @Test
    public void testReEnterAfterFailedAttempt() {
        requiredKeys.add(new Key());

        boolean firstResult = exitCell.handlePlayerInteraction(player, model);
        assertFalse(firstResult);

        player.GetKeys().add(requiredKeys.get(0));
        boolean secondResult = exitCell.handlePlayerInteraction(player, model);
        assertTrue("После сбора ключей должна быть победа", secondResult);
    }
```

```
    @Test
    public void testExitCellRemainsActiveAfterVisit() {
        assertTrue("Клетка должна быть активна изначально", exitCell.shouldEnableCell());

        exitCell.handlePlayerInteraction(player, model);
        assertTrue("Клетка должна оставаться активной после посещения",
exitCell.shouldEnableCell());
    }

    @Test
    public void testExitCellRemainsActiveAfterWin() {
        requiredKeys.add(new Key());
        player.GetKeys().add(requiredKeys.get(0));

        exitCell.handlePlayerInteraction(player, model);
        assertTrue("Клетка должна оставаться активной даже после победы",
exitCell.shouldEnableCell());
    }

    @Test
    public void testTeleportationWhenNoKeys() {
        Cell neighborCell = new Cell(1, 2);

        List<Key> keys = new ArrayList<Key>();
        keys.add(new Key());

        ExitCell tempExit = new ExitCell(keys, new Cell(0,0));
        tempExit.SetNeighbour(neighborCell);
        tempExit.handlePlayerInteraction(player, model);
        assertEquals("Игрок должен телепортироваться на соседнюю клетку", neighborCell,
player.GetCell());
    }

    @Test
    public void testNoTeleportationWhenBlocked() {

        List<Key> keys = new ArrayList<Key>();
        keys.add(new Key());

        ExitCell tempExit = new ExitCell(keys, new Cell(0,0));
        player.moveTo(tempExit);
        boolean result = tempExit.handlePlayerInteraction(player, model);
        assertFalse(result);
        assertEquals("Игрок должен остаться на клетке выхода", tempExit, player.GetCell());
    }

    @Test
    public void testListenerRemoval() {
        IExitCellActionListener listener = new ExitCellObserver(new GameManager());
        exitCell.addExitCellActionListener(listener);
        exitCell.removeExitCellActionListener(listener);

        exitCell.handlePlayerInteraction(player, model);
    }
}
```

5. Список использованной литературы и других источников

1. Логинова, Ф.С. Объектно-ориентированные методы программирования. [Электронный ресурс] : учеб. пособие — Электрон. дан. — СПб. : ИЭО СПбУТУиЭ, 2012. — 208 с. — Режим доступа: http://e.lanbook.com/book/64040

2. Васильев, А.Н. Самоучитель Java с примерами и программами. [Электронный ресурс] : самоучитель — Электрон. дан. — СПб. : Наука и Техника, 2016. — 368 с. — Режим доступа: http://e.lanbook.com/book/90231

3. Программирование на языке Java. Конспект лекций. [Электронный ресурс] : учеб. пособие / А.В. Гаврилов [и др.]. — Электрон. дан. — СПб.: НИУ ИТМО, 2015. — 126 с. — Режим доступа: http://e.lanbook.com/book/91488

Перечень замечаний к работе