

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 №19)

Студент: Зверьком М.К.
Группа: ПриИ-366

Работа зачтена с оценкой _____ «___» _____ 20__ г.

Руководитель проекта, нормоконтроллер _____ Литовкин Д.В.

Волгоград 2021 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: Зверьков М.К.
Группа: ПриИ-366

1. Тема: «Проектирование и реализация программы с использованием
объектно-ориентированного подхода» (индивидуальное задание – вариант №19)
Утверждена приказом от «__» _____ 20__ г. № 101-ст

2. Срок представления работы к защите «__» ____20__ 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 «__» _____ 20__ г.

Руководитель проекта: _____ Литовкин Д.В.

Задание принял к исполнению: _____ Зверьков М. К.

«__» _____ 20__ г.

3.4 Структура программы на уровне классов.....	14
Диаграмма классов вычислительной модели:.....	14
Диаграмма классов представления:.....	15
3.5 Типовые процессы в программе.....	15
3.6 Человеко-машинное взаимодействие.....	15

1 Формулировка задания

Правила игры «Водопроводчик»:

- - имеется поле $N \times M$, в котором располагаются разные сегменты трубы
- - имеется точка А из которой после открытия крана должна потечь вода
- - имеется точка Б в которую должна вода попасть
- - цель - соединить трубы таким образом, чтобы вода попала из точки А в точку Б
- - после открытия крана в точке А визуальное показывается, как вода течет по созданному водопроводу (посегментно); если вода не попадает в точку Б, то игра считается проигранной

Дополнительные требования:

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

Вариативность:

Вариативность: сегменты труб могут быть разного диаметра и материала. Для их соединения необходимы специальные фитинги. Сегменты труб разного диаметра и материала должны быть визуальным различимы. Возможны категории материалов, например:

- металл,
 - сталь,
 - углеродистая сталь,
 - нержавейка и т.д.
 - легированная сталь,
 - и т.д.
 - и т.д.
- пластик,
- и т.д.

НЕ изменяя ранее созданные классы, а используя точки расширения, реализовать: сегменты труб большого и малого диаметра, переходный фитинг между ними. Трубы

могут быть металлическими и пластиковыми. Для их соединения нужны специализированные фитинги: металл-металл, металл-пластик, пластик-пластик.

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 упрощённого варианта задания

Правила игры «Водопроводчик»:

- Имеется поле $N \times M$, где располагаются разные сегменты трубы.
- Есть точка А, из которой после открытия крана должна потечь вода (исток), и точка Б, в которую вода должна попасть (сток).
- Цель игры - соединить трубы так, чтобы вода из точки А попала в точку Б.
- После открытия крана в точке А визуальное показывается движение воды по созданному водопроводу (посегментно). Если у воды нет возможности попасть в точку Б, игра проиграна.

3.2 Функциональные требования (сценарии)

1) Сценарий «Играть»

- 1. По указанию пользователя, Игра стартует.**
- 2. В ответ на запрос пользователя Игра сообщает полную путь к файлу с уровнем,**
- 3. В ответ на сообщение Игры Поле размещает на себе Трубы, Сток и Исток.**
- 4. Делать**
 - 4.1. По указанию пользователя, одна из Труб поворачивается по часовой стрелке.**
- 5. По указанию пользователя, начинается симуляция течения Воды.**
- 6. Если Вода дотекает до Истока, то Игра считает пользователя победителем.**
- 7. Сценарий завершается.**

2) Дочерний сценарий «Поле размещает на себе Трубы, Сток и Исток.»

1. Поле создаёт и ставит исток в одну из ячеек
2. Поле создаёт и ставит сток в одну из ячеек
3. Поле создаёт и расставляет трубы определенного вида в оставшиеся ячейки, либо оставляет их пустыми.

4. Сценарий завершается.

3) Дочерний сценарий «одна из Труб поворачивается по часовой стрелке»

1. **По указанию** пользователя, Игрок сообщает игре какой трубе нужно повернуться
2. **В ответ на запрос** Игрока Игра поворачивает трубу
3. **Сценарий завершается.**

3.1) Альтернативный сценарий «Трубы нет в Ячейке, с которой взаимодействует пользователь. Сценарий **выполняется с п. 2** сценария 3

1. **В ответ на запрос** Игры, Ячейка **сообщает** об отсутствии Трубы в ней.
2. **Сценарий завершается.**

4) Дочерний сценарий «симуляция течения Воды.»

1. **В ответ на запрос** пользователя Игра сообщает Истоку запрос об испускании Воду.
2. **В ответ на сообщение** Игры, Исток создает внутри себя воду с определенным таймером течения.
3. Вода запускает таймер
4. Делать при достижении таймера
 - 4.1. Вода **запрашивает** у Труб, в которые она затекла на предыдущем шаге соединенные с ними Трубы
 - 4.2. Труба запрашивает у Ячейки, в которой она находится соседей по доступным направлениям
 - 4.3. Вода **отправляет сообщение** на получение Воды соединенным Трубам.
 - 4.4. Трубы наполняются и **сообщают** об этом Воде
 - 4.5. Вода обнуляет таймер.
5. **Сценарий завершается.**

4.1) Альтернативный сценарий «Ячейка пустая». Сценарий выполняется с п.

4.2 сценария 4

4.3 **В ответ на сообщение** Ячейка ничего не делает.

4.2) Альтернативный сценарий «Труба повернута так, что не может получить воду». Сценарий выполняется с п. 4.3 сценария 4

4.4 **В ответ на сообщение** Труба не получает воду.

4.2) Альтернативный сценарий «Окончание течения». Сценарий выполняется с п. 4 сценария 4

4.1 Вода сообщает Игре, что не затекла в другие трубы на предыдущем шаге

4.2 Вода останавливает таймер

4. **Сценарий завершается.**

5) Дочерний сценарий «Игра считает пользователя победителем если вода дошла до Стока»

1. Сток сообщает игре, что он наполнился водой
2. В ответ на сообщение Стока Игра останавливает Воду
3. Игра считает пользователя победителем, т.к. вода добралась до истока.
4. **Сценарий завершается.**

5.1) Альтернативный сценарий «Игра считает пользователя проигравшим если вода не дошла до Стока». Сценарий выполняется с п. 3 сценария 5

1. Вода сообщает Игре, что не затекла в новые Трубы
2. Игра считает, что пользователя проиграл, т.к. вода не дошла до стока.
3. **Сценарий завершается.**

3.3 Словарь предметной области

Сущность	Знает	Умеет	Предназначена
Игра	о Поле	Инициализировать начало игры. Определять окончание игры.	Для создания игры. Для определения победителя.
Поле	о Клетках	Находить Сток, Исток, и трубы по координатам	Для десериализации информации о расположении труб. Для коммуникации игры с трубами
Ячейка	о Трубе в ней и соседних Ячейках	Сообщать о трубе внутри себя	Предназначена для хранения трубы
Труба	может знать о Воде	Получать достижимые ячейки. Поворачиваться.	Предназначена для создания пути воды от истока к стоку.
Сток	может знать о Воде	Может сообщать о том, что вода затекла в него. Поворачиваться	Предназначена для определения победы
Исток	о Воде	Может создать воду. Поворачиваться.	Предназначена для испускания воды в трубы
Вода	о Трубе	Может перетекать из одной трубы в другую	Предназначена для пошагового заполнения труб
Игрок		Сообщать о том, что пользователь захотел повернуть трубу	Предназначен для коммуникации пользователя с игрой

3.4 Структура программы на уровне классов

Диаграмма классов вычислительной модели:

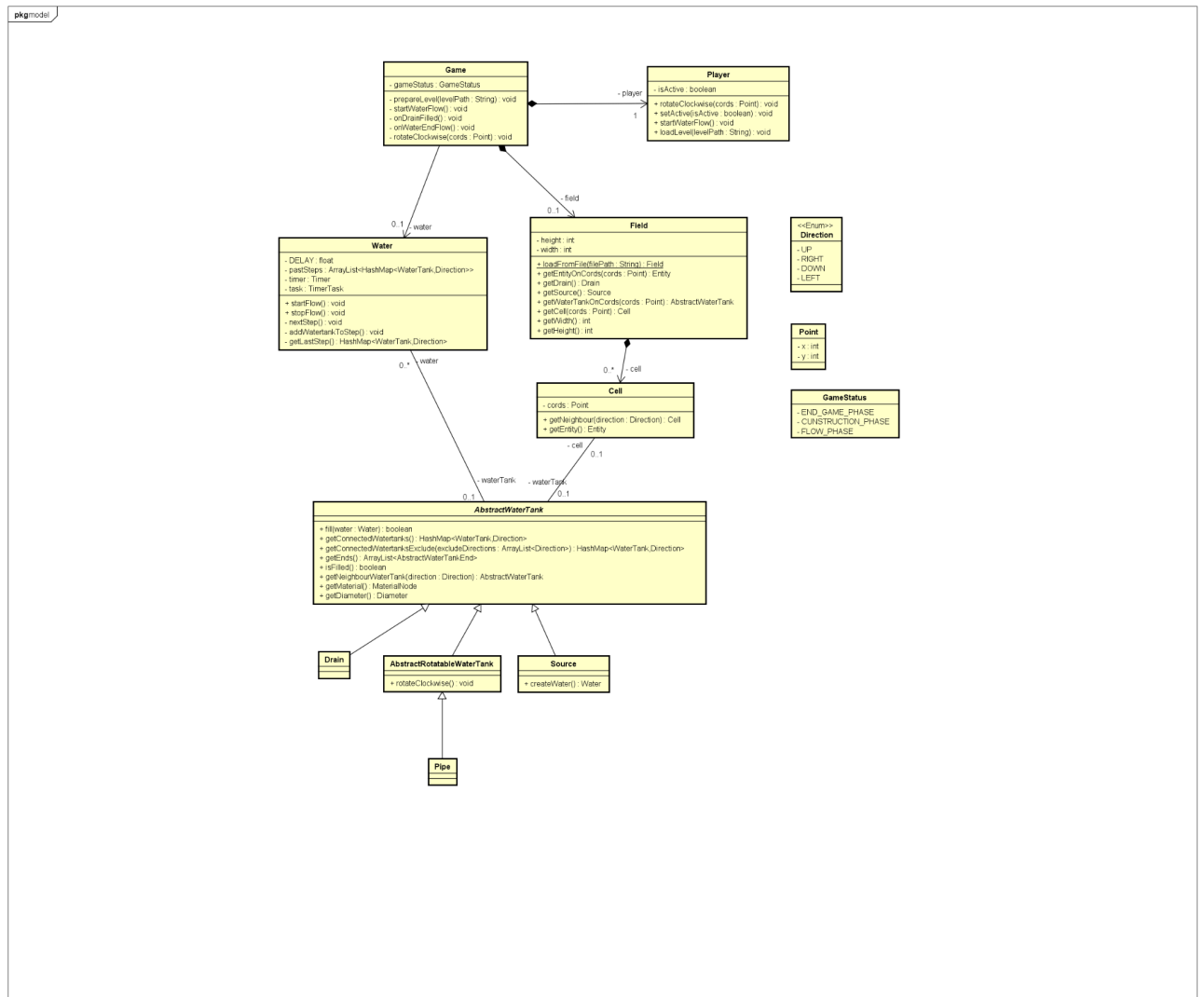
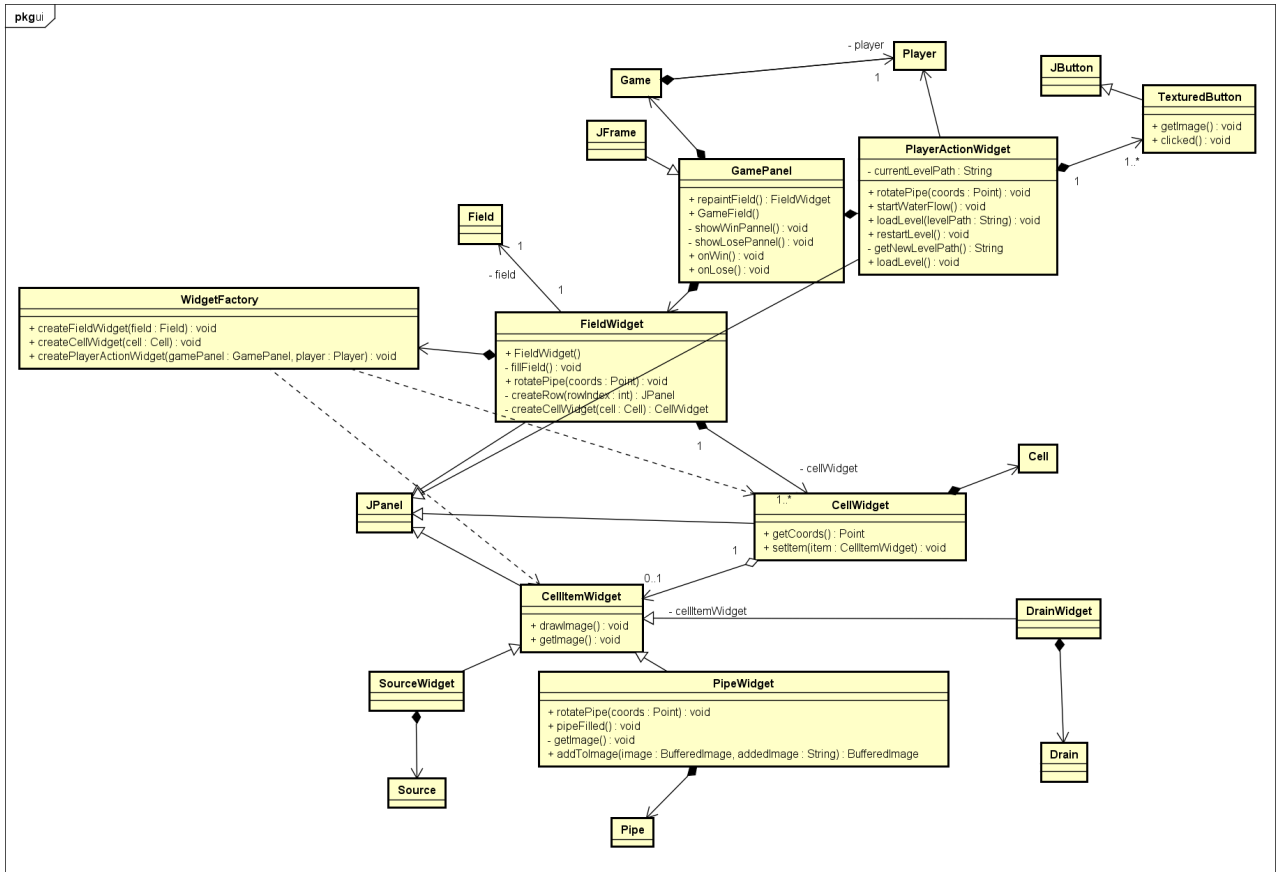
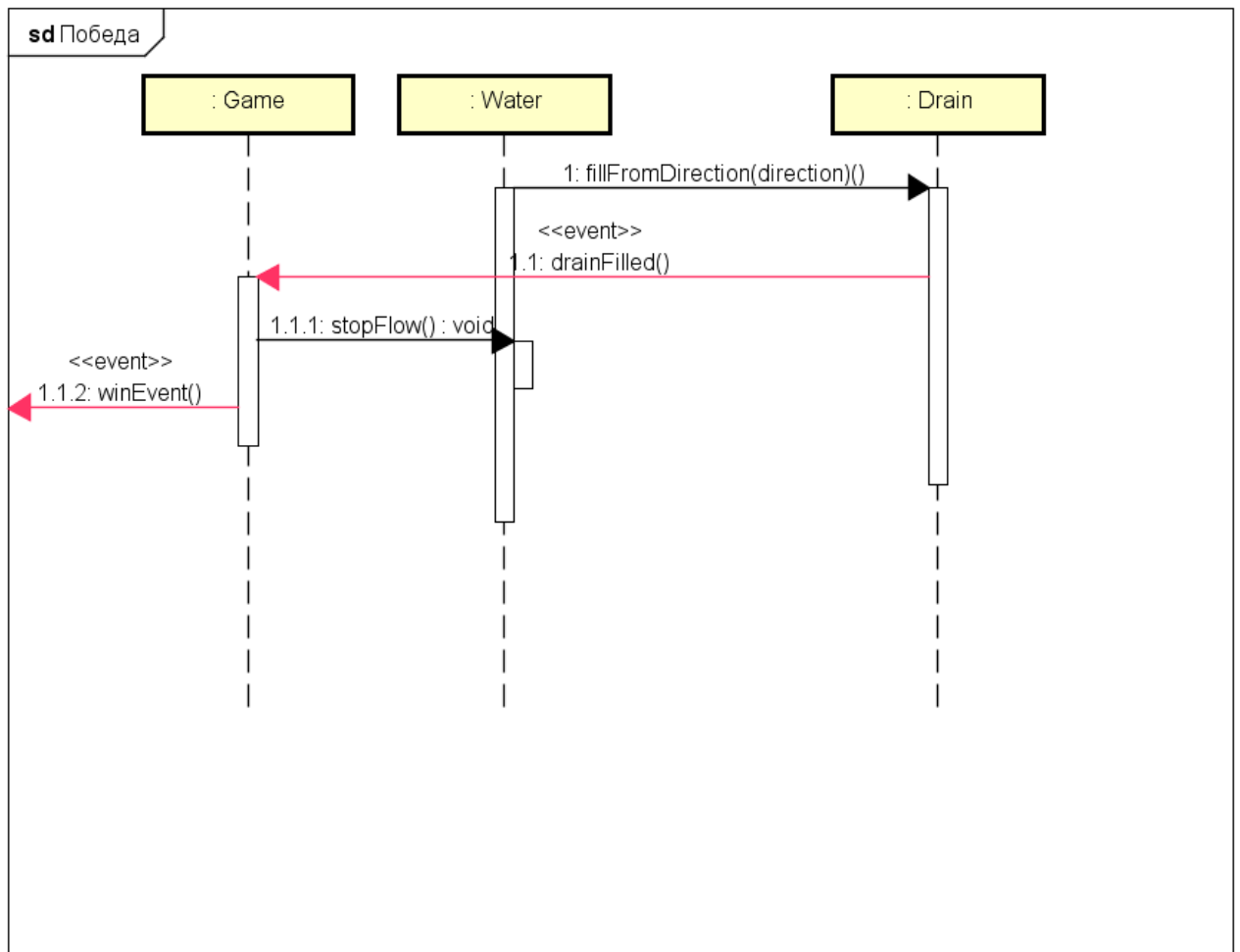


Диаграмма классов представления:

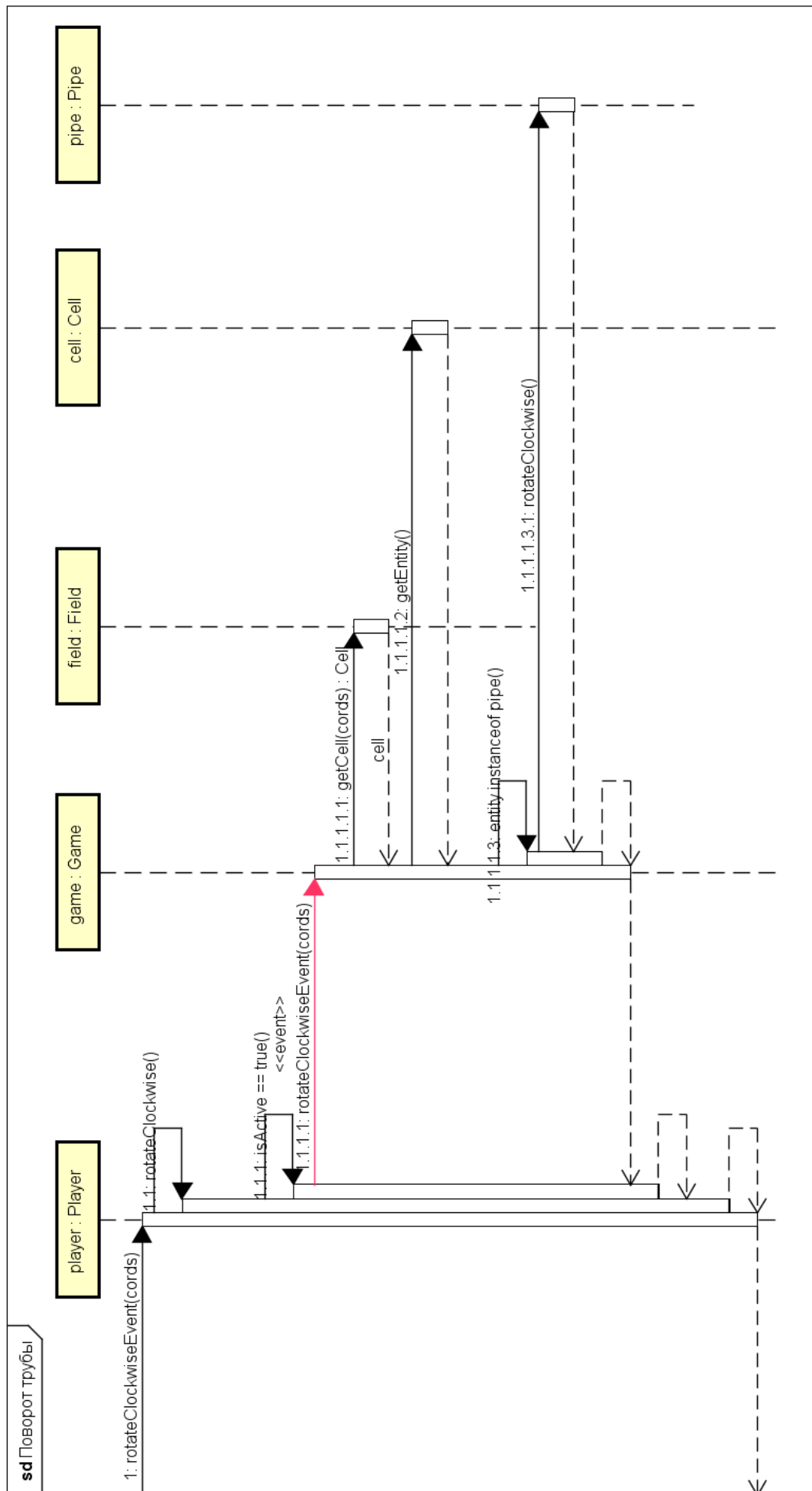


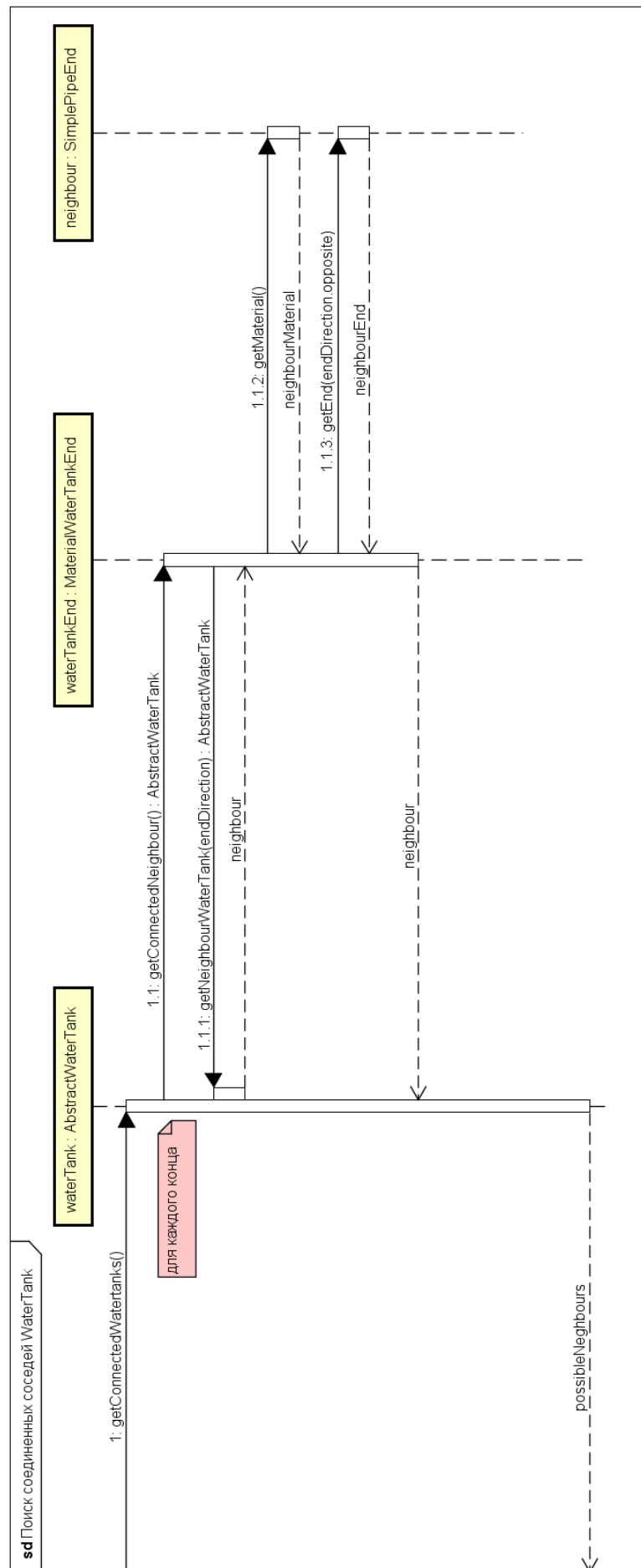
3.5 Типовые процессы в программе



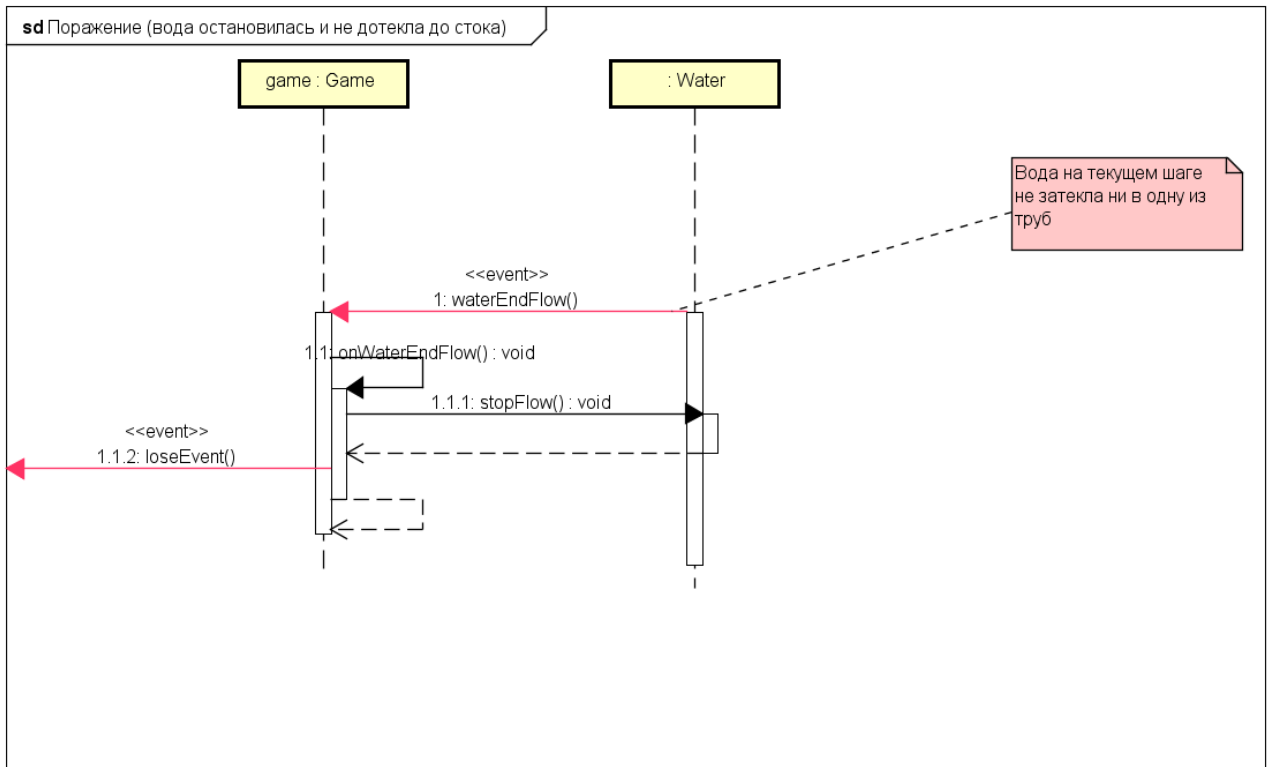
powered by Astah

Победа



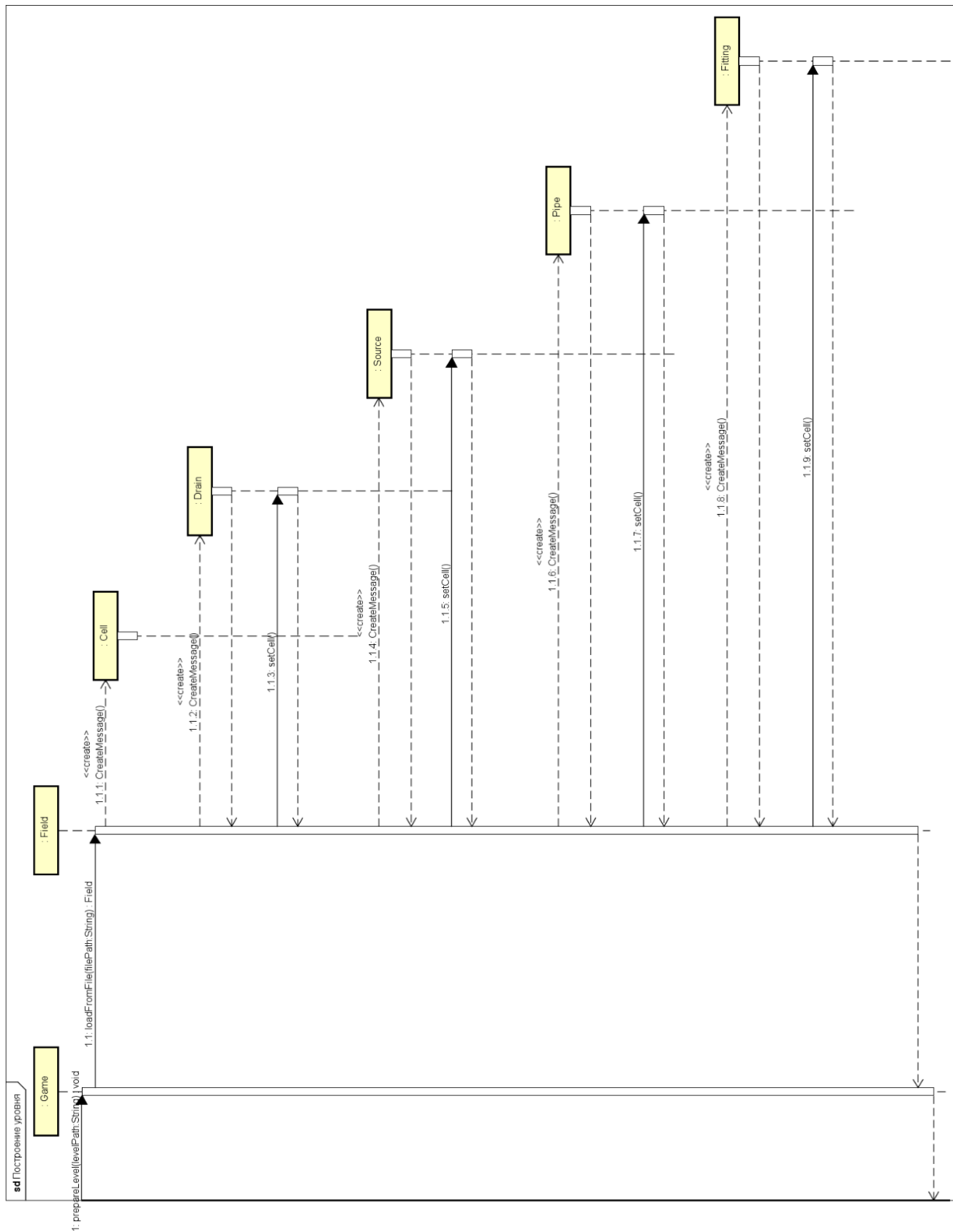


Поиск соединенных соседей резервуара

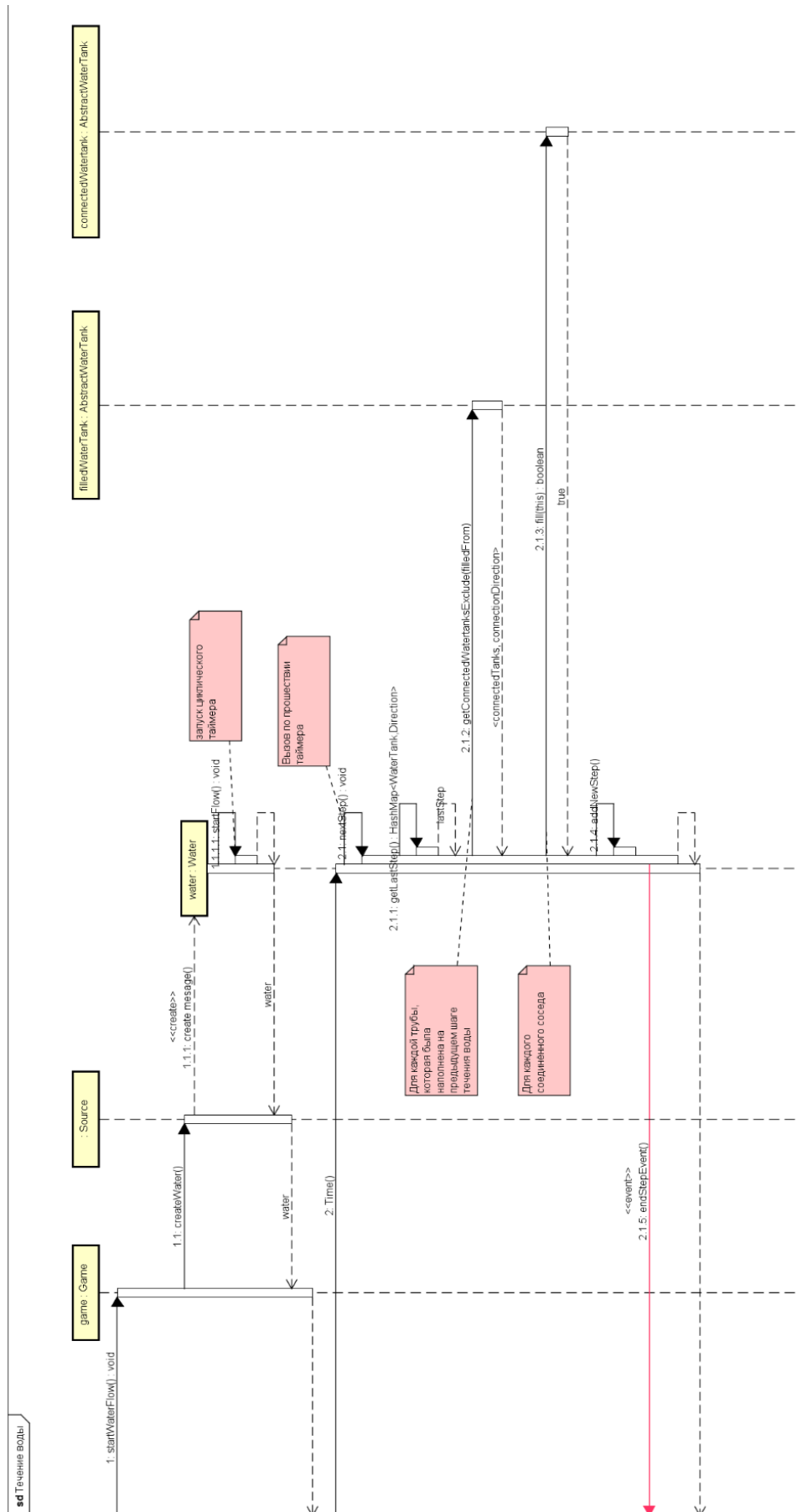


powered by Astah

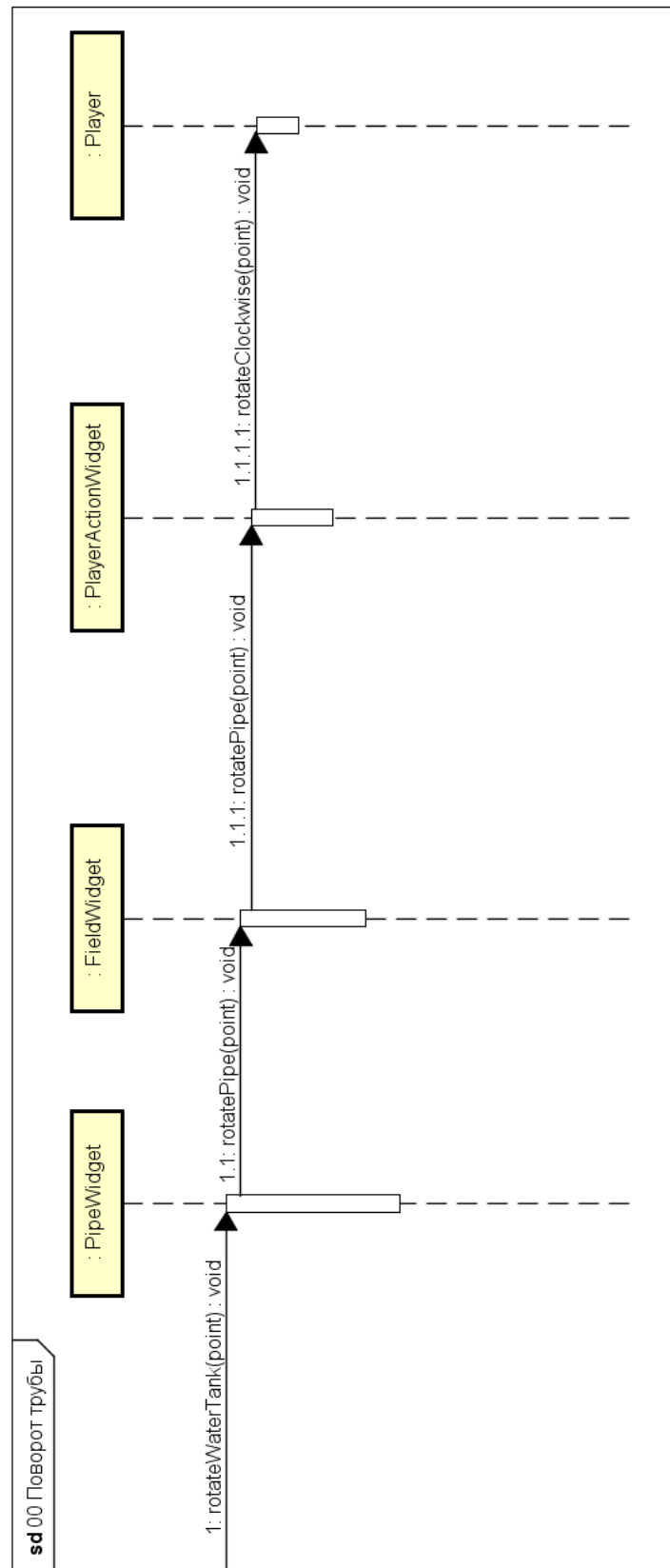
Поражение



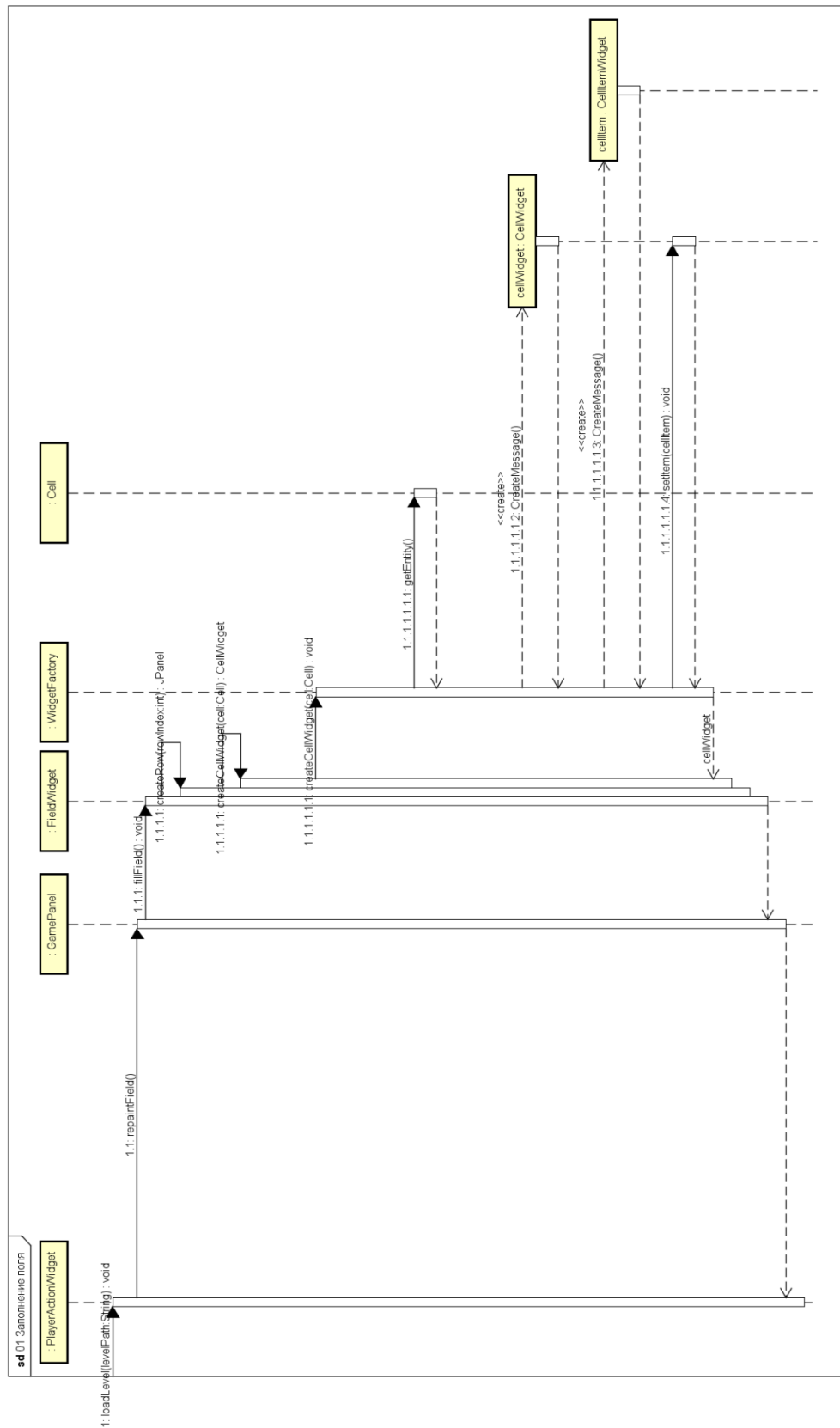
Построение уровня



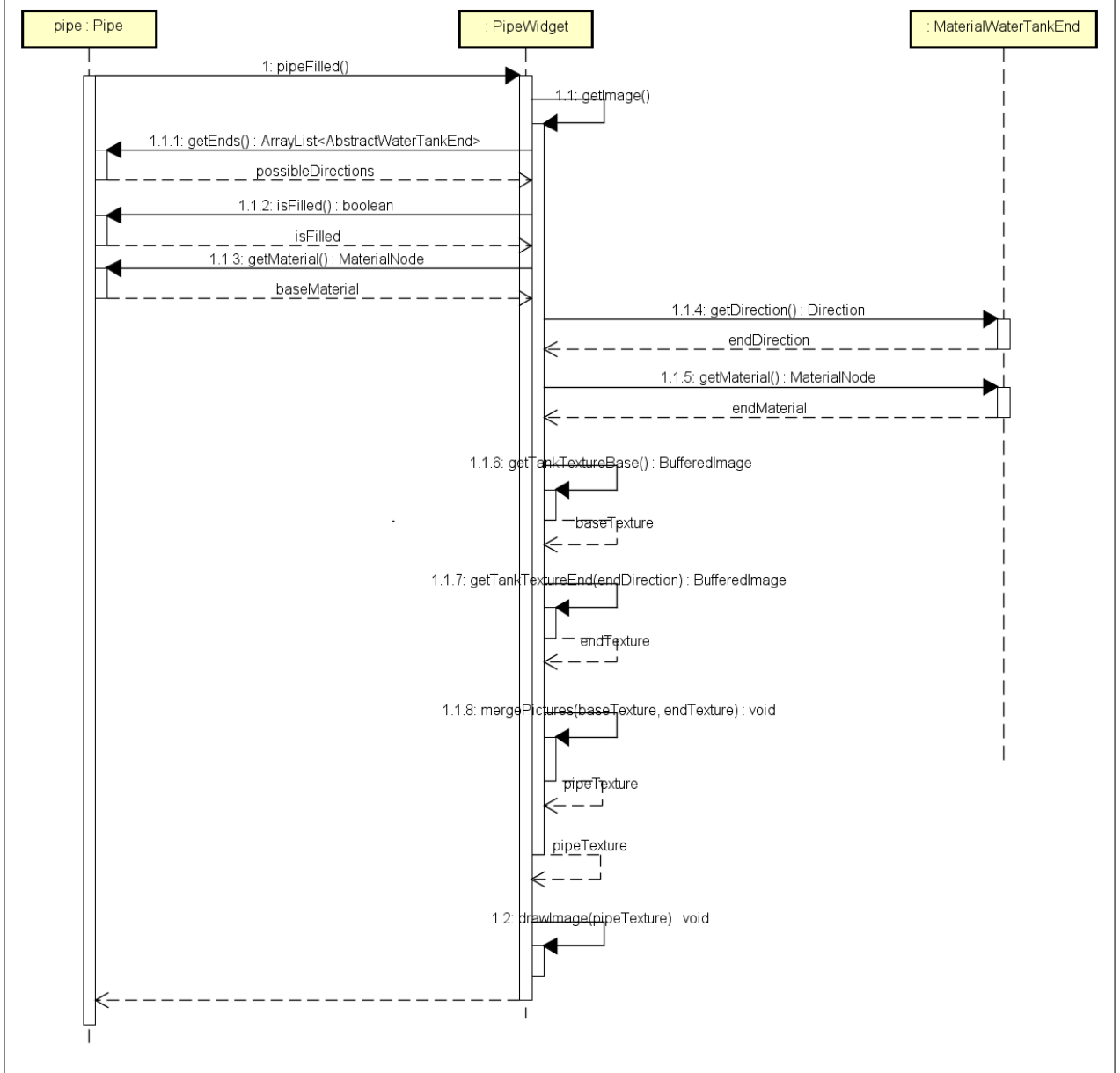
Течение воды



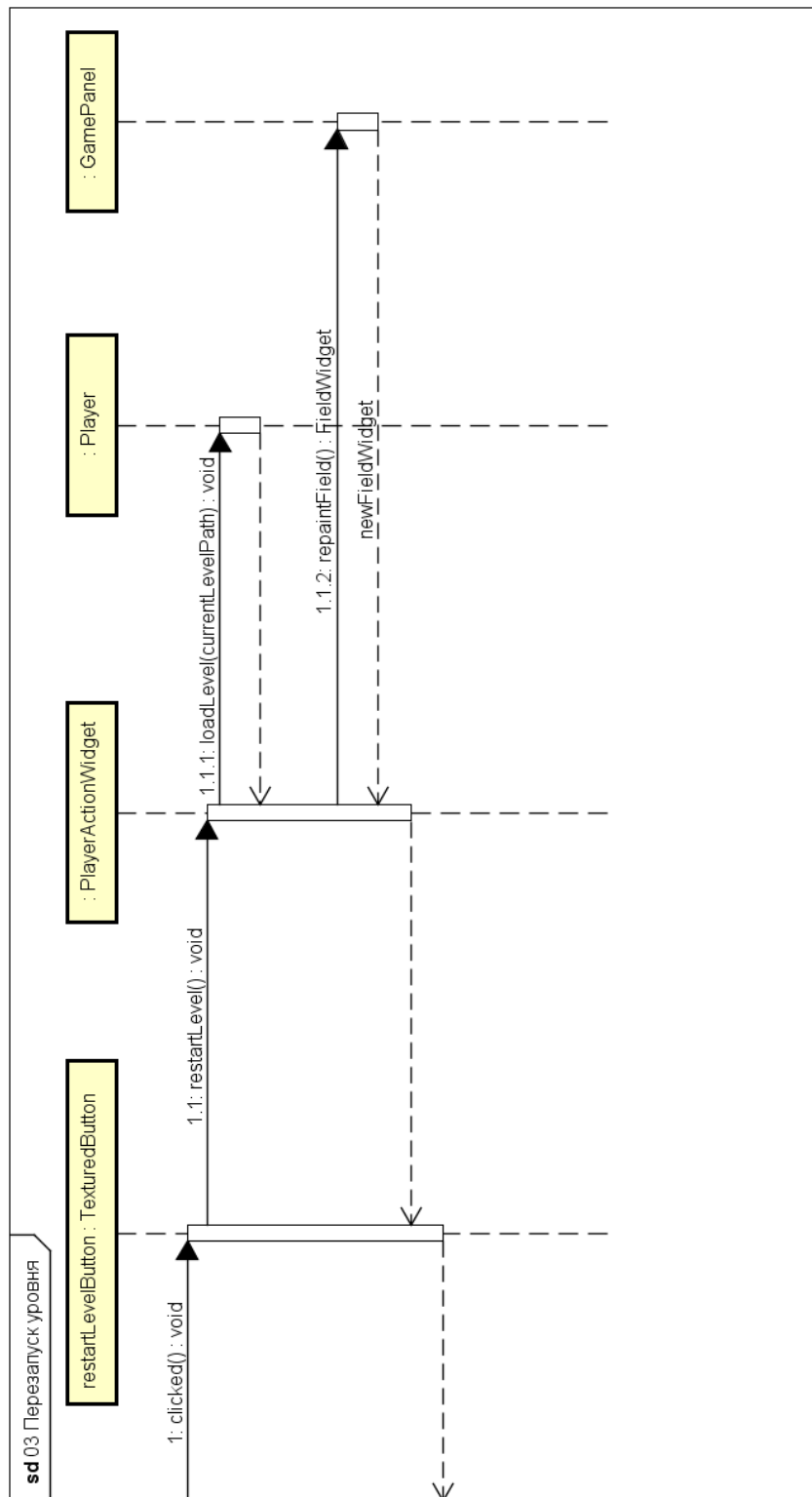
UI Поворот трубы



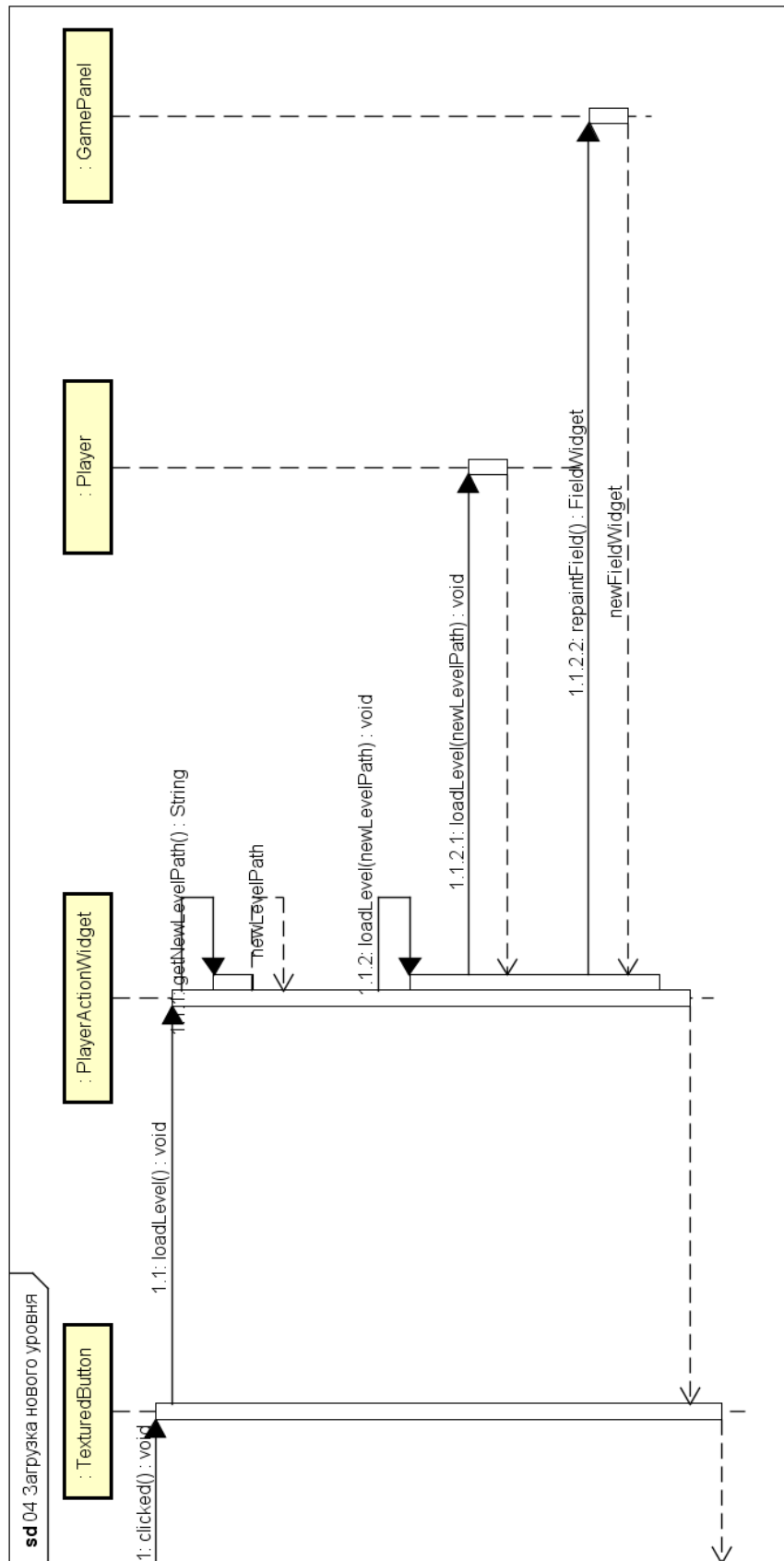
UI Заполнение поля



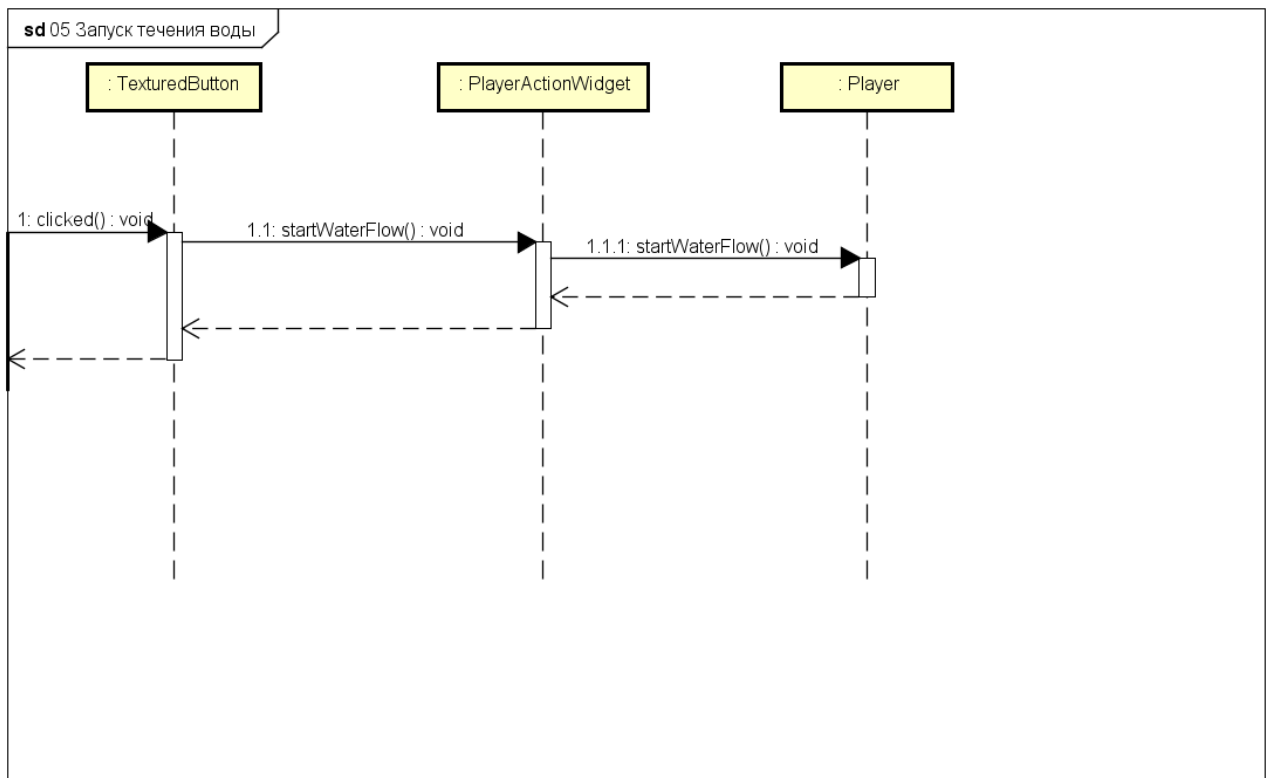
UI Отрисовка резервуара



UI Перезапуск уровня

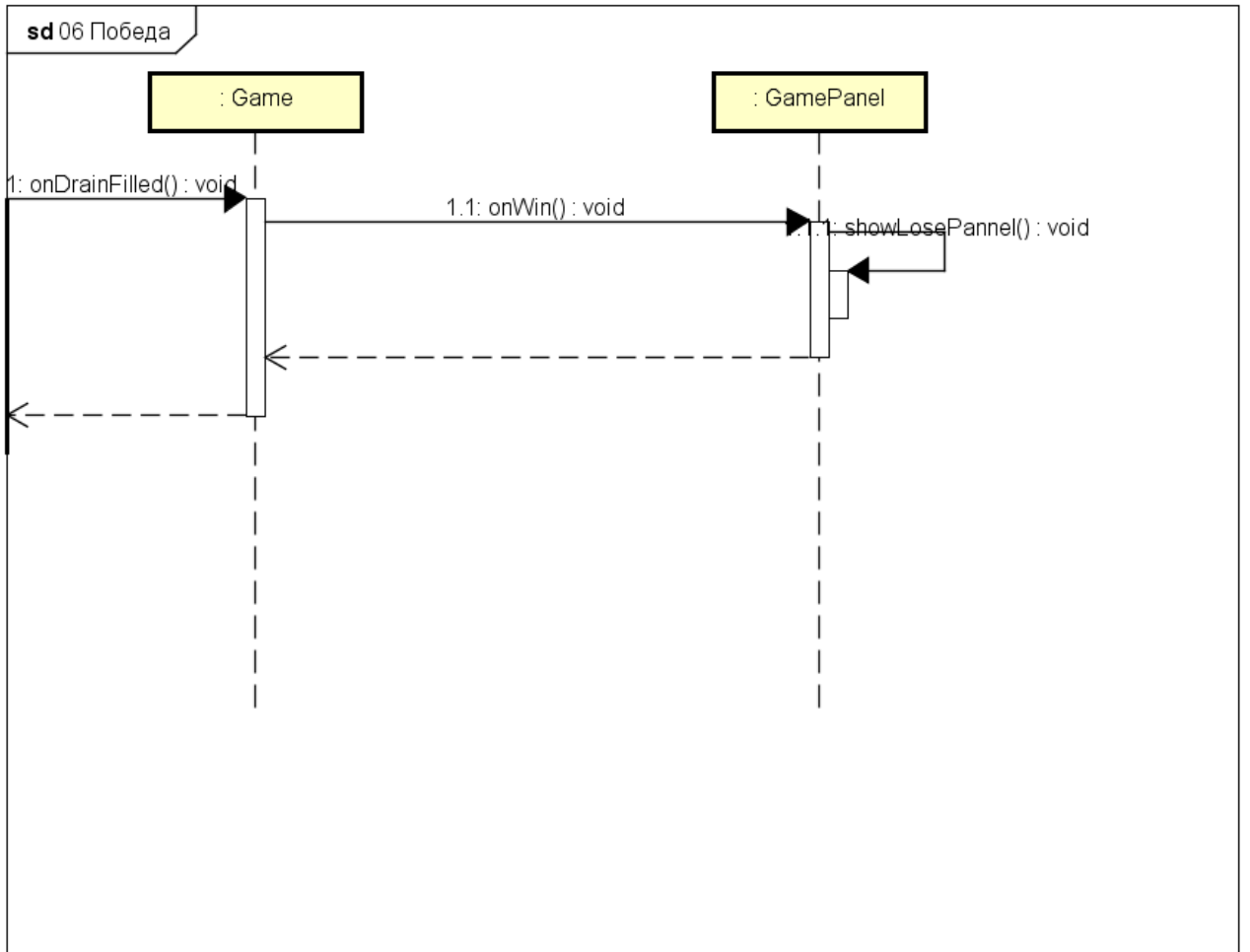


UI Загрузка нового уровня



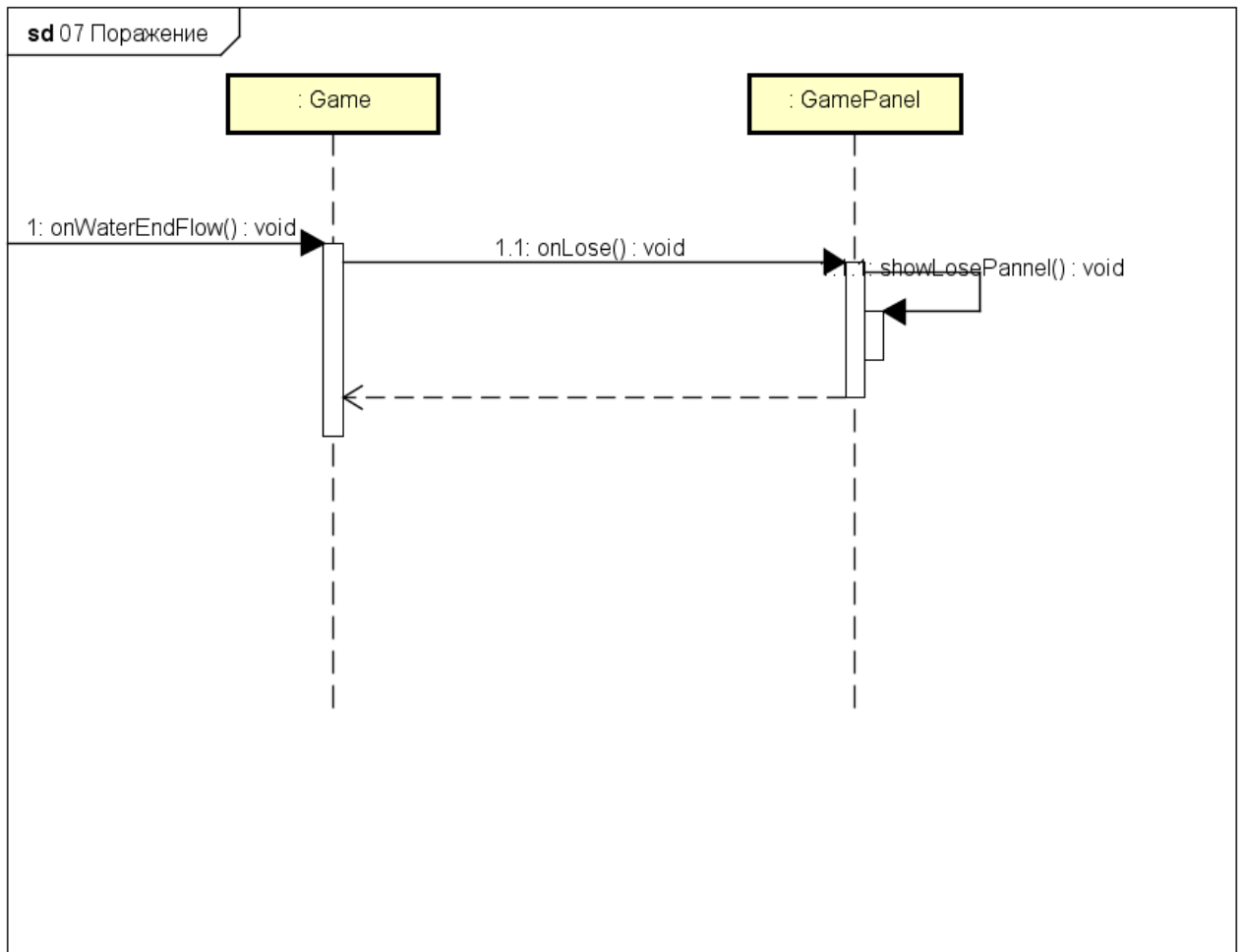
powered by Astah

UI Запуск течения воды



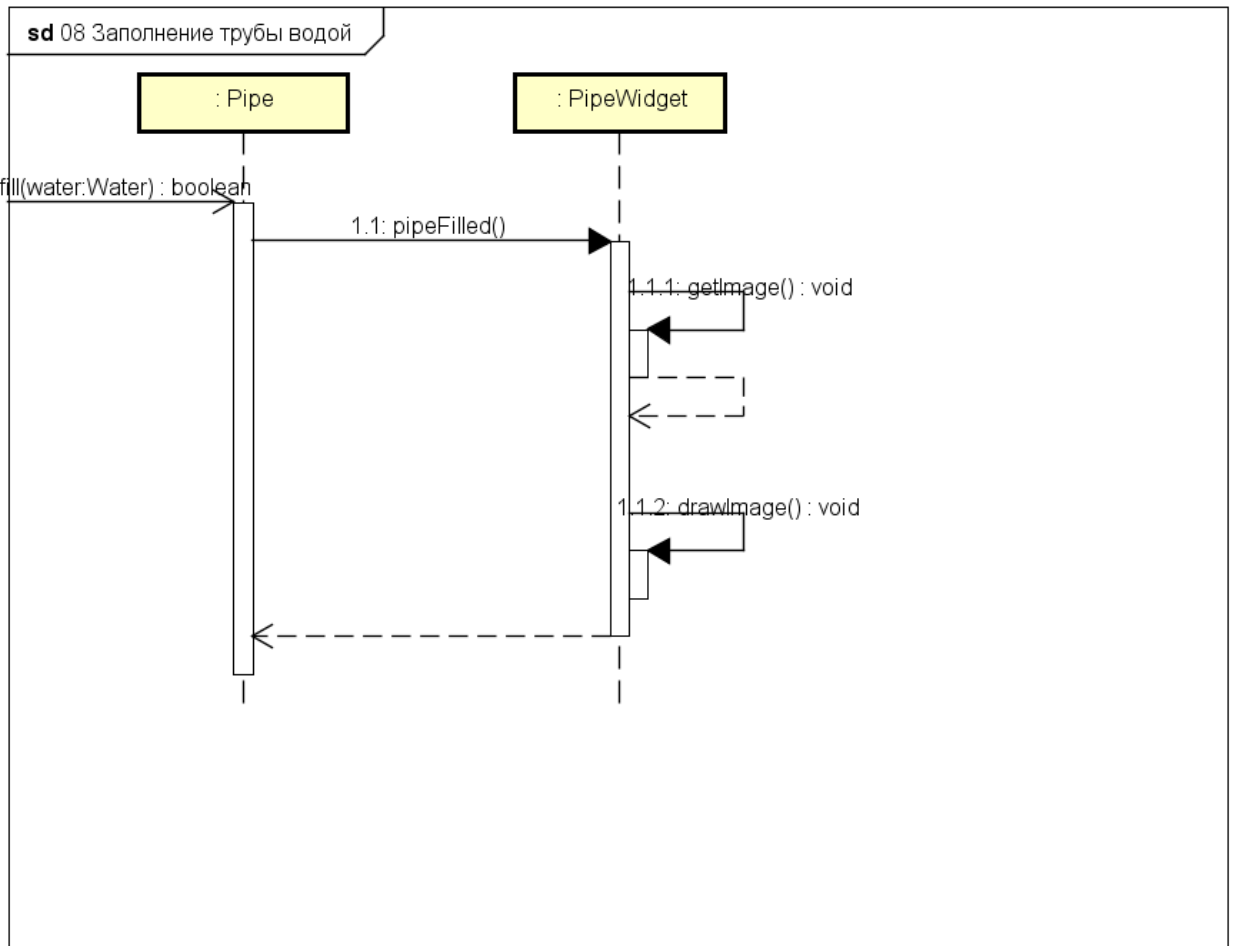
powered by Astah

UI Победа



powered by Astah

UI Поражение



powered by Astah

UI Заполнение трубы водой

3.6 Человеко-машинное взаимодействие

При запуске игры открывается главное меню в котором присутствует одна кнопка, которая позволяет пользователю выбрать уровень для загрузки.:

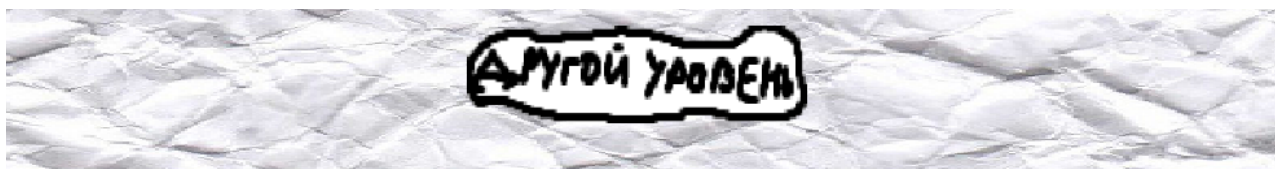


Рис. 1 Главное меню

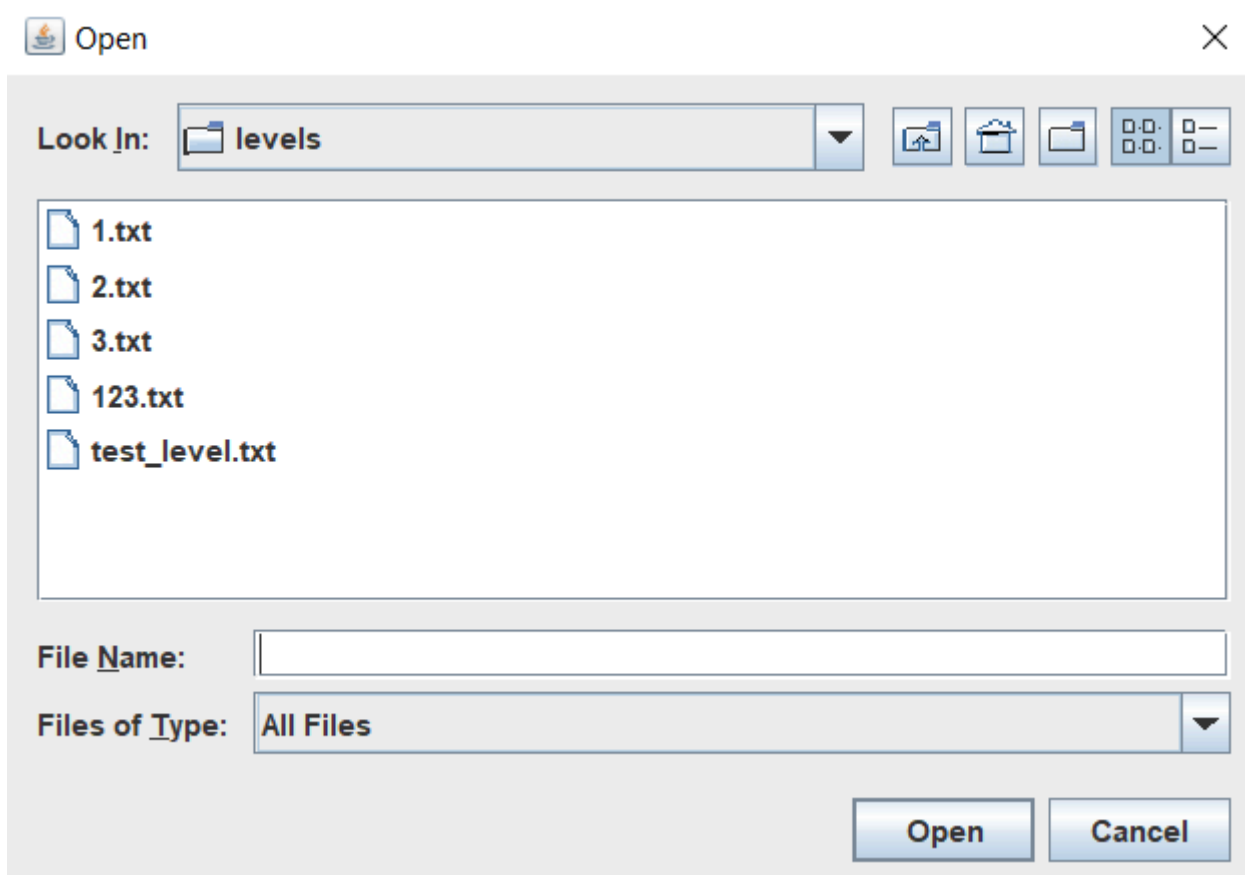


Рис. 2 Окно выбора файла с уровнем

Общий вид главного экрана программы представлен ниже. На нём изображено игровое поле на котором расположены трубы, сток (помечен буквой “D”) и исток (помечен буквой “S”), кнопки для взаимодействия пользователя с игрой (перезапустить уровень, запустить воду, загрузить другой уровень).
МИнимальный размер поля 2x2, максимальный 15x7

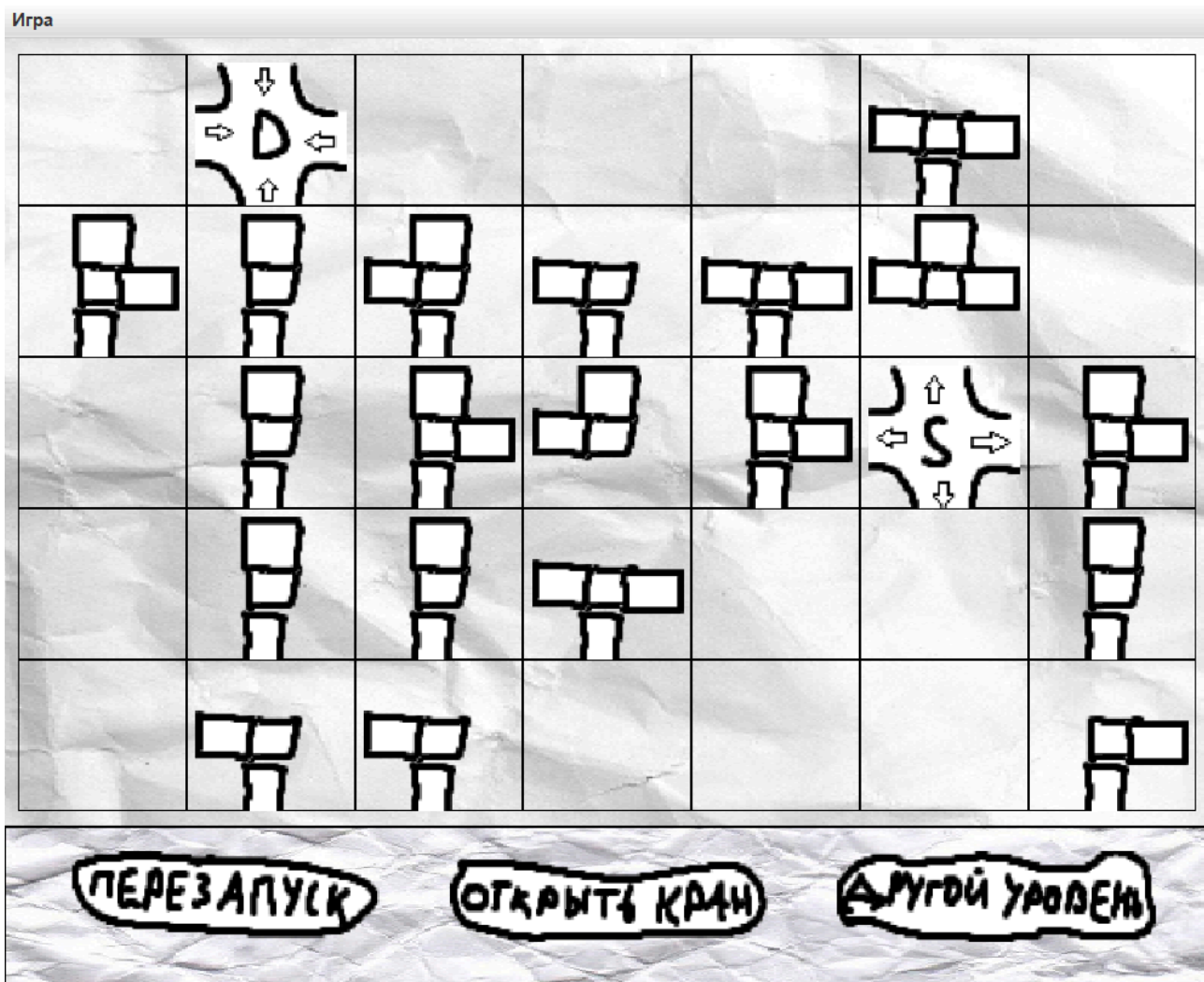


Рис. 3 Главное игровое поле (с размером поля 7x5).

Управление игровым процессом осуществляется с помощью кнопок интерфейса.

Кнопка перезапуска - кнопка, которая перезапускает текущий уровень.

Кнопка "Открыть кран" - кнопка, запускает течение воды.

Кнопка "Другой уровень" - кнопка, которая позволяет пользователю загрузить уровень из файла.

Кнопка "Клетка с трубой" - кнопка, которая позволяет повернуть находящуюся в клетку трубу по часовой стрелке.

Труба делится на 5 сегментов: центральный - есть у всех, остальные в зависимости от направлений трубы. Каждый сегмент имеет два вида текстуры: пустой и заполненный водой. На рисунке 4 представлены все возможные сегменты труб.

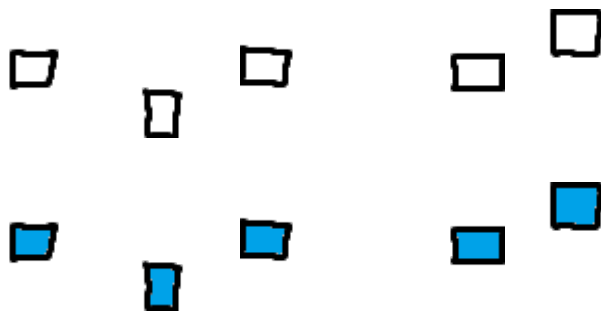


Рис. 4 Сегменты труб (незаполненные и заполненные) слева направо: основной, нижний, левый, правый, верхний.

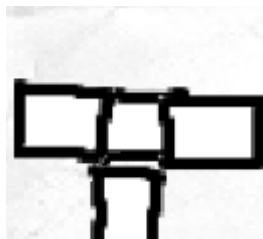


Рис.5 “Собранная труба” с возможными направлениями: влево, вниз, вправо.

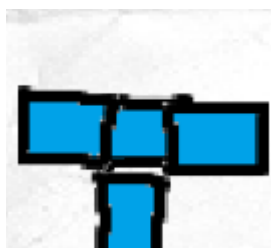


Рис.6 “Собранная труба” заполненная водой.

На рисунке 7 представлено всплывающее окно поражения. Оно появляется когда вода не дотекает до стока и пользователь проигрывает.

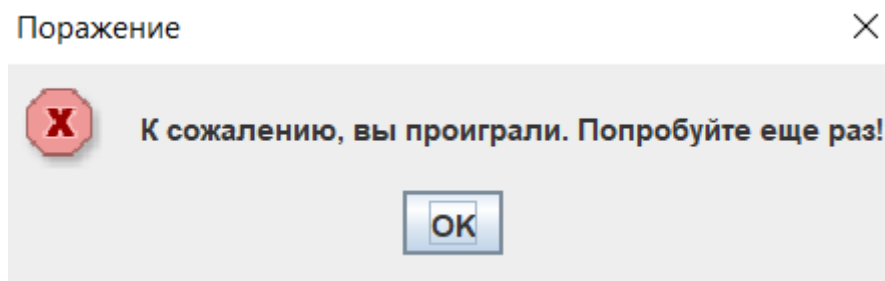


Рис.7 Окно поражения

На рисунке 8 представлено всплывающее окно победы. Оно появляется когда вода дотекает до стока и пользователь выигрывает.

Победа



Поздравляем! Вы выиграли!

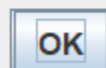


Рис.8 Окно победы

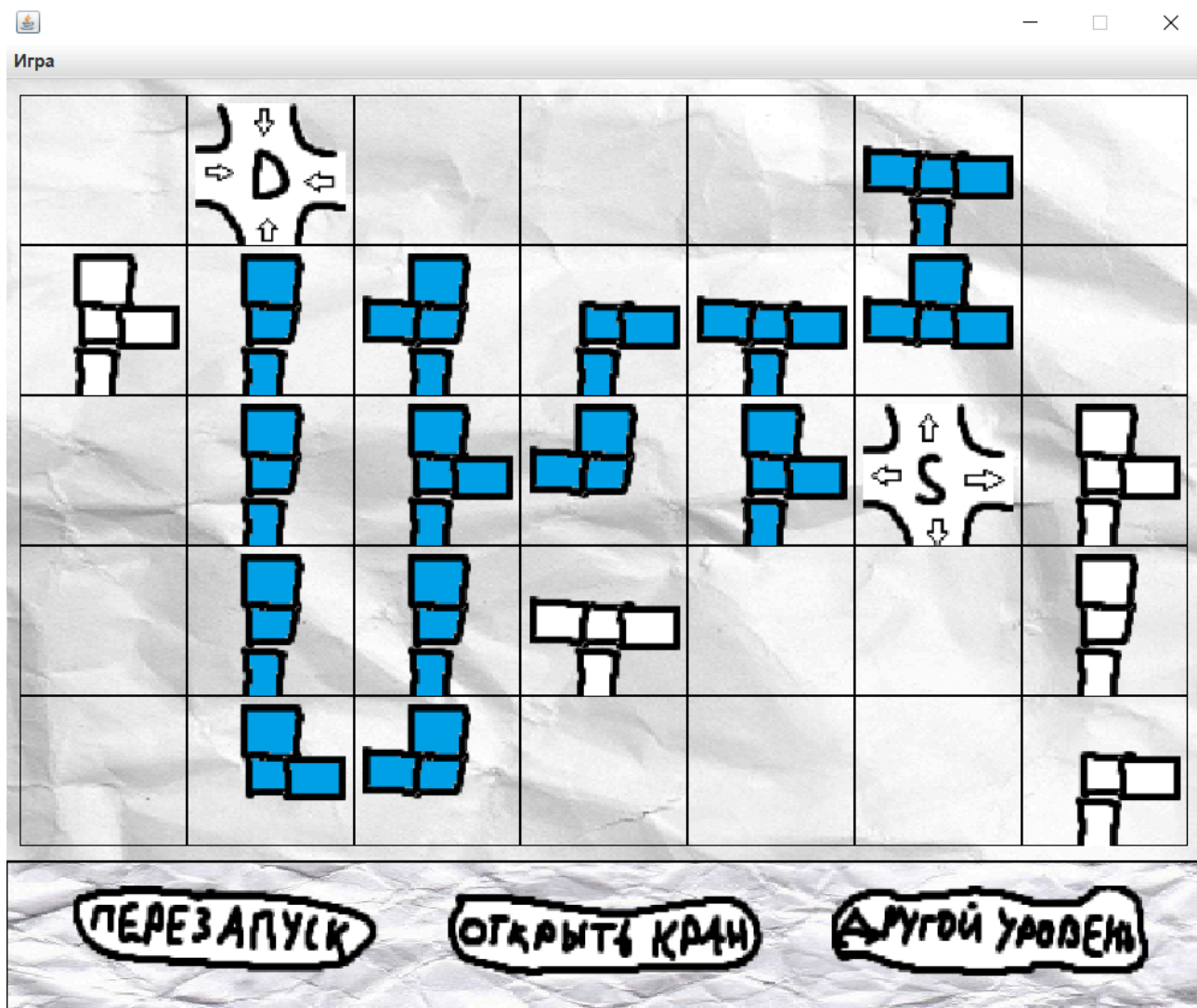


Рис. 9 Поле с заполненными трубами



Рис. 10 Сток



Рис. 11 Исток

3.7 Реализация ключевых классов

```
// Класс игры

public class Game {

    public Field getGameField() {

        return _field;

    }

    public enum GameStatus {

        END_GAME_PHASE,

        CONSTRUCTION_PHASE,

        FLOW_PHASE

    }

    private GameStatus _gameStatus;

    private Field _field;

    private Player _player;

    private Water _water;

    private ArrayList<GameActionListener> _listeners = new ArrayList<>();

    public Game() {

        _player = new Player();

        _player.addListener(new PlayerObserver());

        setGameStatus(GameStatus.END_GAME_PHASE);

    }

    private void setGameStatus(GameStatus gameStatus) {

        _gameStatus = gameStatus;

    }

    public void prepareLevel(String levelPath) throws IOException {

        if (_water != null)

        {

            _water.stop();

        }

    }

}
```

```

    }

    _field = Field.loadFromFile(levelPath);

    _field.getDrain().addListener(new DrainObserver());

    _player.setActive(true);

    setGameStatus(GameStatus.CONSTRUCTION_PHASE);
}

private void rotateClockwise(Point cords){

    if (_gameStatus == GameStatus.CONSTRUCTION_PHASE) {

        Pipe pipe = _field.getPipeOnCords(cords);

        if (pipe != null) {

            pipe.rotateClockwise();

        }

    }

}

public void startWaterFlow() {

    if (_gameStatus == GameStatus.CONSTRUCTION_PHASE) {

        _water = _field.getSource().createWater();

        _water.flow();

        _water.addListener(new WaterObserver());

        setGameStatus(GameStatus.FLOW_PHASE);

    }

}

private void onDrainFilled() {

    if (_gameStatus == GameStatus.FLOW_PHASE) {

        fireWinEvent();

        _water.stop();

        setGameStatus(GameStatus.END_GAME_PHASE);

    }

}

private void onWaterEndFlow() {

    if (_gameStatus == GameStatus.FLOW_PHASE) {

        fireLoseEvent();

        _water.stop();

    }

}

```

```

        setStatus(GameStatus.END_GAME_PHASE);
    }
}

public Field getField() {
    return _field;
}

public Player getPlayer() {
    return _player;
}

public void addListener(GameActionListener listener) {
    _listeners.add(listener);
}

private void fireWinEvent() {
    for (GameActionListener listener : _listeners) {
        listener.winEvent(new GameActionEvent(this));
    }
}

private void fireLoseEvent() {
    for (GameActionListener listener : _listeners) {
        listener.loseEvent(new GameActionEvent(this));
    }
}

private void fireWaterTickEvent() {
    for (GameActionListener listener : _listeners) {
        listener.waterTick(new GameActionEvent(this));
    }
}

private class WaterObserver implements WaterActionListener {

    @Override

```

```

    public void waterEndFlow(WaterActionEvent event) {

        onWaterEndFlow();

    }

    @Override

    public void stepEnd(WaterActionEvent event) {

        fireWaterTickEvent();

    }

}

private class DrainObserver implements DrainActionListener {

    @Override

    public void filled(DrainActionEvent event) {

        onDrainFilled();

    }

}

private class PlayerObserver implements PlayerActionListener {

    @Override

    public void rotateClockwise(PlayerActionEvent event) {

        Game.this.rotateClockwise(event.cords);

    }

    @Override

    public void startFlow(PlayerActionEvent event) {

        startWaterFlow();

    }

    @Override

    public void startLevel(PlayerActionEvent event) {

        try {

            prepareLevel(event.levelPath);

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

}

```

```

        }

    }

}

// Класс поля

public class Field {

    private final static int MAX_WIDTH = 15;

    private final static int MAX_HEIGHT = 7;

    private int _height;

    private int _width;

    private ArrayList<ArrayList<Cell>> _cells = new ArrayList<ArrayList<Cell>>();

    private ArrayList<Pipe> _pipes = new ArrayList<Pipe>();

    private ArrayList<Fitting> _fittings = new ArrayList<Fitting>();

    private Source _source;

    private Drain _drain;

    public Field(int _height, int _width) {

        this._height = _height;

        this._width = _width;

        for (int i = 0; i < _height; i++) {

            ArrayList<Cell> row = new ArrayList<Cell>();

            for (int j = 0; j < _width; j++) {

                row.add(new Cell(new Point(j, i), this));

            }

            _cells.add(row);

        }

    }

    public void setCell(Cell cell) {

        Point cords = cell.getCoords();

        if (cords.x > _width || cords.y > _height || cords.x < 0 || cords.y < 0) {

```



```

        throw new IllegalArgumentException("Invalid cell coordinates: " + cords.x + " " +
cords.y);
    }

    ArrayList<Cell> row = _cells.get(cords.y);
    row.add(cords.x, cell);

    if (cell.getField() == null) {
        cell.setField(this, cords);
    }

}

public int getHeight() {
    return _height;
}

public int getWidth() {
    return _width;
}

public Cell getCell(Point coords) {
    return _cells.get(coords.y).get(coords.x);
}

public Source getSource() {
    return _source;
}

public Drain getDrain() {
    return _drain;
}

public Pipe getPipeOnCords(Point cords) {
    Entity entity = getCell(cords).getEntity();

    if (entity instanceof Pipe) {
        return (Pipe) entity;
    } else {
        return null;
    }
}

```

```

    }

}

public ArrayList<Pipe> getPipes() {

    return _pipes;

}

public String toString() {

    String result = "";

    for (int i = 0; i < _height; i++) {

        for (int j = 0; j < _width; j++) {

            result += _cells.get(i).get(j).toString();

        }

        result += "\n";

    }

    return result;

}

public static Field loadFromFile(String filename) throws IOException {

    BufferedReader reader = new BufferedReader(new FileReader(filename));

    String[] size = reader.readLine().split(" ");

    int width = Integer.parseInt(size[0]);

    int height = Integer.parseInt(size[1]);

    if (height < 2 || width < 2 || height > MAX_HEIGHT || width > MAX_WIDTH) {

        reader.close();

        throw new IllegalArgumentException("Invalid field dimensions (min: 2x2, max: 15x7): "
+ height + " " + width);

    }

    //TODO: to separate method configureField

    Field field = new Field(height, width);

    for (int i = 0; i < height ; i++) {

        String[] cells = reader.readLine().split("");

        for (int j = 0; j < width; j++) {

            configurateCell(cells[j], field, new Point(j, i));

        }

    }

}

```

```

        if (field.getSource() == null) {

            reader.close();

            throw new IllegalArgumentException("No source in the field");

        }

        if (field.getDrain() == null) {

            reader.close();

            throw new IllegalArgumentException("No drain in the field");

        }

        //TODO: to separate method configureElements

        MaterialNode root = MaterialNode.configure();

        field.getSource().configureTankWithOneMaterial(reader.readLine(), root);

        field.getDrain().configureTankWithOneMaterial(reader.readLine(), root);

        for (Pipe pipe : field.getPipes()) {

            pipe.configureTankWithOneMaterial(reader.readLine(), root);

        }

        for (Fitting fitting : field.getFittings()) {

            fitting.configureTankEndsToMaterial(reader.readLine(), root);

        }

        reader.close();

        return field;

    }

    public ArrayList<Fitting> getFittings() {

        return _fittings;

    }

    private static void configurateCell(String cell, Field field, Point coords) {

        switch (cell) {

            case "□":

                break;

            case "s":

                field.addSimpleSource(coords);

                break;

            case "d":

```

```

        field.addSimpleDrain(coords);

        break;

    case "p":

        field.addSimplePipe(coords);

        break;

    case "f":

        field.addSimpleFitting(coords);

        break;

    default:

        throw new IllegalArgumentException("Invalid cell type: " + cell);

    }

}

private boolean addCellItem(Point coords, Entity entity) {

    Cell cell = getCell(coords);

    if (cell.getEntity() != null) {

        return false;

    }

    cell.setEntity(entity);

    return true;

}

public void addSimpleFitting(Point coords) {

    Fitting fitting = new Fitting();

    if (addCellItem(coords, fitting)) {

        _fittings.add(fitting);

    }

}

public void addSimplePipe(Point coords) {

    Pipe pipe = new Pipe();

    if (addCellItem(coords, pipe)) {

        _pipes.add(pipe);

    }

}

public void addSimpleSource(Point coords) {

```

```

        if (getSource() != null) {

            throw new IllegalArgumentException("Multiple sources in the field");

        }

        Source source = new Source();

        if (addCellItem(coords, source)) {

            _source = source;

        }

    }

    public void addSimpleDrain(Point coords) {

        if (getDrain() != null) {

            throw new IllegalArgumentException("Multiple drains in the field");

        }

        Drain drain = new Drain();

        if (addCellItem(coords, drain)) {

            _drain = drain;

        }

    }

}

//Класс направления
public enum Direction{

    UP, RIGHT, DOWN, LEFT;

    public Direction rotateClockwise() {

        int nextIndex = this.ordinal() + 1;

        if (nextIndex > 3)

        {

            nextIndex = 0;

        }

        return Direction.values()[nextIndex];

    }

    public Direction rotateCounterClockwise() {

```

```

        int nextIndex = this.ordinal() - 1;

        if (nextIndex < 0)

        {

            nextIndex = 3;

        }

        return Direction.values()[nextIndex];

    }

    public Direction turnAround() {

        int nextIndex = this.ordinal() - 2;

        if (nextIndex < 0)

        {

            nextIndex = nextIndex + 4;

        }

        return Direction.values()[nextIndex];

    }

    @Override

    public String toString() {

        switch (this) {

            case UP:

                return "UP";

            case RIGHT:

                return "RIGHT";

            case DOWN:

                return "DOWN";

            case LEFT:

                return "LEFT";

            default:

                return "UNKNOWN";

        }

    }

}

```

```

//Класс клетки

```

```

public class Cell {

    private Point _coords;

    private Entity _entity;

    private Field _field;

    public Cell() {

    }

    public Cell(Point coords, Field field) {

        _field = field;

        _coords = coords;

    }

    public void setField(Field field, Point coords) {

        _coords = coords;

        _field = field;

        _field.setCell(this);

    }

    public Field getField() {

        return _field;

    }

    public void setEntity(Entity entity) {

        _entity = entity;

        if (_entity.getCell() == null) {

            _entity.setCell(this);

        }

    }

    public Entity getEntity() {

        return _entity;

    }

    public Point getCoords() {

        return _coords;

    }

```

```

public Cell getNeighbor(Direction direction) {

    if (_field == null)

        return null;

    Point neighborCoords = new Point(_cords);

    switch (direction) {

        case UP:

            neighborCoords.translate(0, -1);

            break;

        case DOWN:

            neighborCoords.translate(0, 1);

            break;

        case LEFT:

            neighborCoords.translate(-1, 0);

            break;

        case RIGHT:

            neighborCoords.translate(1, 0);

            break;

    }

    if (neighborCoords.x < 0 || neighborCoords.x >= _field.getWidth() || neighborCoords.y < 0
    || neighborCoords.y >= _field.getHeight()) {

        return null;

    }

    return _field.getCell(neighborCoords);

}

public void setCords(Point cords) {

    _cords = cords;

}

@Override

public String toString() {

    String result = "\u25A1";

    if (_entity != null) {

        result = _entity.toString();

    }

    return result;
}

```



```

    }
}

//Класс игрока
public class Player {

    private boolean _isActive;

    private ArrayList<PlayerActionListener> _listeners = new ArrayList<>();

    public Player() {

    }

    public void setActive(boolean isActive) {

        _isActive = isActive;

    }

    public void rotateClockwise(Point cords) {

        if (!_isActive)

            return;

        fireRotateClockwiseEvent(cords);

    }

    public void startLevel(String levelPath) {

        fireStartLevelEvent(levelPath);

    }

    public void startWaterFlow() {

        if (!_isActive)

            return;

        fireStartWaterFlowEvent();

    }

    private void fireStartLevelEvent(String levelPath) {

        PlayerActionEvent event = new PlayerActionEvent(this);

```

```

        event.levelPath = levelPath;

        for (PlayerActionListener listener : _listeners) {

            listener.startLevel(event);

        }

    }

    private void fireRotateClockwiseEvent(Point cords) {

        PlayerActionEvent event = new PlayerActionEvent(this);

        event.cords = cords;

        for (PlayerActionListener listener : _listeners) {

            listener.rotateClockwise(event);

        }

    }

    private void fireStartWaterFlowEvent() {

        PlayerActionEvent event = new PlayerActionEvent(this);

        for (PlayerActionListener listener : _listeners) {

            listener.startFlow(event);

        }

    }

    public void addListener(PlayerActionListener listener) {

        _listeners.add(listener);

    }

}

//Класс воды

public class Water {

    private ArrayList<HashMap<AbstractWaterTank,Direction>> _pastSteps = new ArrayList<>();

    private ArrayList<WaterActionListener> _listeners = new ArrayList<>();

    private final TimerTask _task = new TimerTask() {

        public void run() {

            nextStep();

        }

    }

```

```

};

private final Timer _timer = new Timer("Flow timer");

private static int DELAY = 1000;

public ArrayList<HashMap<AbstractWaterTank,Direction>> getAllSteps() {

    return _pastSteps;

}

public static void setWaterDelay(int delay) {

    DELAY = delay;

}

public Water(AbstractWaterTank source) {

    HashMap<AbstractWaterTank,Direction> zeroStep = new HashMap<>();

    zeroStep.put(source, null);

    _pastSteps.add(zeroStep);

}

public void flow() {

    _timer.scheduleAtFixedRate(_task, DELAY, DELAY);

}

public void stop() {

    _timer.cancel();

}

public void nextStep()

{

    HashMap<AbstractWaterTank,Direction> newStep = new HashMap<>();

    for (AbstractWaterTank abstractWaterTank : lastStep().keySet()) {

        HashMap<Direction, AbstractWaterTank> neighbours =
abstractWaterTank.getConnectedWaterTanks();

        for (Direction direction : neighbours.keySet()) {

            boolean isFilled = neighbours.get(direction).fill(Water.this);

            if (isFilled) {

                newStep.put(neighbours.get(direction), direction.turnAround());

            }

}

```

```

        }

    }

    if (newStep.isEmpty()) {

        stop();

        fireEndFlowEvent();

    } else {

        _pastSteps.add(newStep);

    }

    fireEndStepEvent();

}

private HashMap<AbstractWaterTank,Direction> lastStep() {

    return _pastSteps.get(_pastSteps.size()-1);

}

public void addListener (WaterActionListener listener) {

    _listeners.add(listener);

}

private void fireEndStepEvent() {

    for (WaterActionListener listener : _listeners) {

        listener.stepEnd(new WaterActionEvent(this));

    }

}

private void fireEndFlowEvent() {

    for (WaterActionListener listener : _listeners) {

        listener.waterEndFlow(new WaterActionEvent(this));

    }

}

}

```

```

//Класс абстрактного резервуара

public class AbstractWaterTank extends Entity {

```

```

private ArrayList<AbstractWaterTankEnd> _ends = new ArrayList<AbstractWaterTankEnd>();

private Water _water;

protected ArrayList<WaterTankActionListener> _listeners = new
ArrayList<WaterTankActionListener>();

protected MaterialNode _material;

private Diameter _diameter;

public AbstractWaterTank() {

    super();

}

public Diameter getDiameter() {

    return _diameter;

}

public void setDiameter(Diameter diameter) {

    _diameter = diameter;

}

public boolean addEnd(AbstractWaterTankEnd end) {

    if (_ends.contains(end)) {

        System.out.println("End already exists");

        return false;

    }

    if (end.setParentWaterTank(this)) {

        _ends.add(end);

        return true;

    }

    return false;

}

public void addEnds(ArrayList<AbstractWaterTankEnd> ends) {

    for (AbstractWaterTankEnd end : ends) {

        addEnd(end);

    }

}

```

```

public void removeEnd(AbstractWaterTankEnd end) {
    _ends.remove(end);
}

public void removeEnds(ArrayList<AbstractWaterTankEnd> ends) {
    for (AbstractWaterTankEnd end : ends) {
        removeEnd(end);
    }
}

public void clearEnds() {
    _ends.clear();
}

public ArrayList<AbstractWaterTankEnd> getEnds() {
    return _ends;
}

public AbstractWaterTankEnd getEnd(Direction direction) {
    for (AbstractWaterTankEnd end : _ends) {
        if (end.getDirection() == direction) {
            return end;
        }
    }
    return null;
}

public boolean fill(Water water){
    if (_water == null)
    {
        _water = water;
        fireFilledEvent();
        return true;
    }
    return false;
}

```

```

public AbstractWaterTank getNeighbour(Direction direction) {

    Cell cell = getCell().getNeighbor(direction);

    if (cell == null)

    {

        return null;

    }

    Entity entity = cell.getEntity();

    if (entity instanceof AbstractWaterTank abstractWaterTank) {

        return abstractWaterTank;

    }

    return null;

}

public HashMap<Direction, AbstractWaterTank> getConnectedWaterTanks() {

    HashMap<Direction, AbstractWaterTank> connectedWaterTanks = new HashMap<>();

    for (AbstractWaterTankEnd end : _ends) {

        if (end.getConnectionedNeighbour() != null) {

            connectedWaterTanks.put(end.getDirection(), end.getConnectionedNeighbour());

        }

    }

    return connectedWaterTanks;

}

public Water getWater() {

    return _water;

}

public void addListener(WaterTankActionListener listener) {

    _listeners.add(listener);

}

private void fireFilledEvent() {

    for (WaterTankActionListener listener : _listeners) {

        listener.tankFilled(new WaterTankActionEvent(this));

    }

}

```

```

private HashMap<String, String> _dict = null;

@Override

public String toString() {

    if (_dict == null)

    {

        _dict = new HashMap<String, String>();

        _dict.put("00000", "o");

        _dict.put("00010", "-");

        _dict.put("00100", "- ");

        _dict.put("00110", "=");

        _dict.put("01000", "l");

        _dict.put("01010", "ff");

        _dict.put("01100", "fl");

        _dict.put("01110", "ff");

        _dict.put("10000", "l");

        _dict.put("10010", "ll");

        _dict.put("10100", "ll");

        _dict.put("10110", "ll");

        _dict.put("11000", "ll");

        _dict.put("11010", "ll");

        _dict.put("11100", "ll");

        _dict.put("11110", "ll");


        _dict.put("00001", "●");

        _dict.put("00011", "-");

        _dict.put("00101", "-");

        _dict.put("00111", "—");

        _dict.put("01001", "l");

        _dict.put("01011", "l");

        _dict.put("01101", "l");

        _dict.put("01111", "l");

        _dict.put("10001", "l");

        _dict.put("10011", "l");

        _dict.put("10101", "l");

        _dict.put("10111", "l");

        _dict.put("11001", "l");
    }
}

```



```

        _dict.put("11011", "┌");
        _dict.put("11101", "└");
        _dict.put("11111", "⊕");
    }

    String key = "";

    key += isHaveEnd(Direction.UP) ? "1" : "0";

    key += isHaveEnd(Direction.DOWN) ? "1" : "0";

    key += isHaveEnd(Direction.LEFT) ? "1" : "0";

    key += isHaveEnd(Direction.RIGHT) ? "1" : "0";

    key += _water == null ? "0" : "1";

    return _dict.get(key);
}

private boolean isHaveEnd(Direction direction) {
    for (AbstractWaterTankEnd end : _ends) {
        if (end.getDirection() == direction) {
            return true;
        }
    }

    return false;
}

public MaterialNode getMaterial() {
    return _material;
}

public void setMaterial(MaterialNode material) {
    _material = material;
}

public void configureTankWithOneMaterial(String config, MaterialNode materialRoot) {
    // <<Материал трубы>>; <<Диаметр>>; <<Верхний выход (1 или 0)>>; <<Правый выход (1 или 0)>>; <<Нижний выход (1 или 0)>>; <<Левый выход (1 или 0)>>

    String[] configSplit = config.split(";");

    if (configSplit.length != 6) {

```

```

        throw new IllegalArgumentException("Invalid one material tank configuration: " +
config);
    }

    String materialName = configSplit[0];
    String diameter = configSplit[1];
    String newConfig = materialName + ";" + diameter + ";";

    for (int i = 2; i < 6; i++) {
        newConfig += (configSplit[i].equals("1") ? materialName + ";" + diameter: "null;0")
+ ";";
    }

    configureTankEndsToMaterial(newConfig, materialRoot);
}

public void configureTankEndsToMaterial(String config, MaterialNode materialRoot) {
    // <<Материал танка>>; <<диаметр танка>>; <<материал верхнего конца>>; <<диаметр верхнего
конца>>; <<материал правого конца>>; <<диаметр правого конца>>; <<материал нижнего конца>>;
<<диаметр нижнего конца>>; <материал левого конца>>; <<диаметр левого конца>>

    String[] configParts = config.split(";");

    if (configParts.length != 10) {
        throw new IllegalArgumentException("Invalid tank configuration: " + config);
    }

    MaterialNode pipeMaterial =
MaterialNode.getMaterial(MaterialType.valueOf(configParts[0]));

    if (pipeMaterial == null) {
        throw new IllegalArgumentException("Invalid material: " + configParts[0]);
    }

    setMaterial(pipeMaterial);

    setDiameter(new Diameter(Integer.parseInt(configParts[1])));

    clearEnds();

    for (int i = 2; i < configParts.length; i += 2) {
        if (configParts[i].equals("null")) {
            continue;
        }
    }
}

```

```

        MaterialNode material =
MaterialNode.getMaterial(MaterialType.valueOf(configParts[i]));

        if (material == null) {

            throw new IllegalArgumentException("Invalid material: " + configParts[i]);

        }

        Diameter diameter = new Diameter(Integer.parseInt(configParts[i + 1]));

        addEnd(new MaterialWaterTankEnd(Direction.values()[i - 2] / 2], material,
diameter));

    }

}

```

//Класс поворачивающегося резервуара

```

public abstract class AbstractRotatableWaterTanks extends AbstractWaterTank {

    public void rotateClockwise() {

        for (AbstractWaterTankEnd end : getEnds()) {

            end.rotateClockwise();

        }

        fireRotatedEvent();

    }

    private void fireRotatedEvent() {

        RotatableWaterTankActionEvent event = new RotatableWaterTankActionEvent(this);

        for (WaterTankActionListener listener : _listeners) {

            if (listener instanceof RotatableWaterTankActionListener) {

                ((RotatableWaterTankActionListener) listener).pipeRotated(event);

            }

        }

    }

}

```

//Класс абстрактного конца резервуара

```

public abstract class AbstractWaterTankEnd {

    Direction _direction;

```

```

AbstractWaterTank _parentAbstractWaterTank;

public AbstractWaterTankEnd(Direction direction) {
    _direction = direction;
}

public Direction getDirection() {
    return _direction;
}

public void rotateClockwise() {
    _direction = _direction.rotateClockwise();
}

public boolean setParentWaterTank(AbstractWaterTank parentAbstractWaterTank) {
    _parentAbstractWaterTank = parentAbstractWaterTank;
    return true;
}

public abstract AbstractWaterTank getConnectedNeighbour();

public AbstractWaterTank getParentWaterTank() {
    return _parentAbstractWaterTank;
}

@Override
public String toString() {
    return _direction.toString();
}

}

//Класс стандартного конца резервуара
public class DefaultWaterTankEnd extends AbstractWaterTankEnd {
    public DefaultWaterTankEnd(Direction direction) {
        super(direction);
    }
}

```

```

    }

    @Override
    public AbstractWaterTank getConnectedNeighbour() {
        AbstractWaterTank neighAbstractWaterTank = getNeighbour(getDirection());

        if (neighAbstractWaterTank != null) {
            if (neighAbstractWaterTank.getEnd(getDirection().turnAround()) instanceof
DefaultWaterTankEnd) {
                return neighAbstractWaterTank;
            }
        }

        return null;
    }

    private AbstractWaterTank getNeighbour(Direction direction) {
        return getParentWaterTank().getNeighbour(direction);
    }
}

//Класс истока
public class Source extends AbstractWaterTank {

    public Source(){
        super();
    }

    public Water createWater(){
        if (getWater() != null){
            throw new IllegalStateException("Water already exists");
        }

        Water water = new Water(this);

        fill(water);

        return water;
    }
}

```

```

@Override

public String toString(){

    return "s";

}

}

//Класс стока

public class Drain extends AbstractWaterTank {

    private ArrayList<DrainActionListener> _listeners = new ArrayList<>();

    public Drain(){

        super();

    }

    @Override

    public boolean fill(Water water){

        boolean isFilled = super.fill(water);

        if (isFilled){

            fireFilledEvent();

        }

        return isFilled;

    }

    @Override

    public String toString(){

        return "d";

    }

    private void fireFilledEvent() {

        for (DrainActionListener listener : _listeners) {

            listener.filled(new DrainActionEvent(this));

        }

    }

    public void addListener(DrainActionListener listener) {

```

```

        _listeners.add(listener);
    }
}

//Класс трубы
public class Pipe extends AbstractRotatableWaterTanks {

    public Pipe() {

        super();
    }

}

//Тест клетки
public class CellTests {

    @Test

    void setField() {

        Cell cell = new Cell();

        Field field = new Field(1, 1);

        cell.setField(field, new Point(0, 0));

        assertEquals(field, cell.getField());

        assertEquals(new Point(0, 0), cell.getCoords());

        assertEquals(cell, field.getCell(new Point(0, 0)));
    }

    @Test

    void setPipeOnCell() {

        Cell cell = new Cell();

        Field field = new Field(1, 1);

        cell.setField(field, new Point(0, 0));

        AbstractWaterTank pipe = new AbstractWaterTank();

        cell.setEntity(pipe);

        assertEquals(pipe, cell.getEntity());
    }

    @Test

```

```

void getNeighborAllNeighboursReachable() {

    Field field = new Field(3, 3);

    Cell cell = new Cell();

    cell.setField(field, new Point(1, 1));

    HashMap<Direction, Cell> expected = new HashMap<Direction, Cell>();

    expected.put(Direction.UP, field.getCell(new Point(1, 0)));

    expected.put(Direction.DOWN, field.getCell(new Point(1, 2)));

    expected.put(Direction.LEFT, field.getCell(new Point(0, 1)));

    expected.put(Direction.RIGHT, field.getCell(new Point(2, 1)));

    for (Direction direction : Direction.values()) {

        assertEquals(expected.get(direction), cell.getNeighbor(direction));

    }

}

@Test

void getNeighbourNoNeighbours() {

    Field field = new Field(1, 1);

    Cell cell = new Cell();

    cell.setField(field, new Point(0, 0));

    for (Direction direction : Direction.values()) {

        assertEquals(null, cell.getNeighbor(direction));

    }

}

@Test

void getNeighbourNoField() {

    Cell cell = new Cell();

    for (Direction direction : Direction.values()) {

        assertEquals(null, cell.getNeighbor(direction));

    }

}

@Test

void getNeighborAtTheCorner() {

    Field field = new Field(3, 3);

```



```

        Cell cell = new Cell();

        cell.setField(field, new Point(0, 0));

        HashMap<Direction, Cell> expected = new HashMap<Direction, Cell>();

        expected.put(Direction.UP, null);

        expected.put(Direction.DOWN, field.getCell(new Point(0, 1)));

        expected.put(Direction.LEFT, null);

        expected.put(Direction.RIGHT, field.getCell(new Point(1, 0)));

        for (Direction direction : Direction.values()) {

            assertEquals(expected.get(direction), cell.getNeighbor(direction));

        }

    }

    @Test

    void toStringNoEntity() {

        Cell cell = new Cell();

        assertEquals("□", cell.toString());

    }

    @Test

    void toStringWithEntity() {

        Cell cell = new Cell();

        AbstractWaterTank pipe = new AbstractWaterTank();

        cell.setEntity(pipe);

        assertEquals("o", cell.toString());

    }

}

//Тест направления

public class DirectionTests {

    @Test

    void turnAroundUP() {

        Direction dir = Direction.UP;

        Direction expectedDir = Direction.DOWN;

        Direction actualDir = dir.turnAround();

        assertEquals(expectedDir, actualDir);

    }

}

```

```

    }

    @Test
    void turnAroundLEFT() {
        Direction dir = Direction.LEFT;

        Direction expectedDir = Direction.RIGHT;

        Direction actualDir = dir.turnAround();

        assertEquals(expectedDir, actualDir);
    }

    @Test
    void turnAroundDOWN() {
        Direction dir = Direction.DOWN;

        Direction expectedDir = Direction.UP;

        Direction actualDir = dir.turnAround();

        assertEquals(expectedDir, actualDir);
    }

    @Test
    void turnAroundRIGHT() {
        Direction dir = Direction.RIGHT;

        Direction expectedDir = Direction.LEFT;

        Direction actualDir = dir.turnAround();

        assertEquals(expectedDir, actualDir);
    }

    @Test
    void rotateClockwiseUP() {
        Direction dir = Direction.UP;

        Direction expectedDir = Direction.RIGHT;

        Direction actualDir = dir.rotateClockwise();

        assertEquals(expectedDir, actualDir);
    }

    @Test
    void rotateClockwiseLEFT() {
        Direction dir = Direction.LEFT;

```

```

        Direction expectedDir = Direction.UP;

        Direction actualDir = dir.rotateClockwise();

        assertEquals(expectedDir, actualDir);
    }

    @Test
    void rotateClockwiseDOWN() {

        Direction dir = Direction.DOWN;

        Direction expectedDir = Direction.LEFT;

        Direction actualDir = dir.rotateClockwise();

        assertEquals(expectedDir, actualDir);
    }

    @Test
    void rotateClockwiseRIGHT() {

        Direction dir = Direction.RIGHT;

        Direction expectedDir = Direction.DOWN;

        Direction actualDir = dir.rotateClockwise();

        assertEquals(expectedDir, actualDir);
    }

    @Test
    void rotateCounterClockwiseUP() {

        Direction dir = Direction.UP;

        Direction expectedDir = Direction.LEFT;

        Direction actualDir = dir.rotateCounterClockwise();

        assertEquals(expectedDir, actualDir);
    }

    @Test
    void rotateCounterClockwiseLEFT() {

        Direction dir = Direction.LEFT;

        Direction expectedDir = Direction.DOWN;

        Direction actualDir = dir.rotateCounterClockwise();

        assertEquals(expectedDir, actualDir);
    }

```

```

@Test
void rotateCounterClockwiseDOWN() {
    Direction dir = Direction.DOWN;
    Direction expectedDir = Direction.RIGHT;
    Direction actualDir = dir.rotateCounterClockwise();
    assertEquals(expectedDir, actualDir);
}

@Test
void rotateCounterClockwiseRIGHT() {
    Direction dir = Direction.RIGHT;
    Direction expectedDir = Direction.UP;
    Direction actualDir = dir.rotateCounterClockwise();
    assertEquals(expectedDir, actualDir);
}

@Test
void turnAroundTwice() {
    Direction dir = Direction.UP;
    Direction expectedDir = Direction.UP;
    Direction actualDir = dir.turnAround().turnAround();
    assertEquals(expectedDir, actualDir);
}
}

//Тест стока
class DrainTests {
    private Drain _drain;
    private MockDrainActionListener _mockDrainActionListener;

    @BeforeEach
    void setUp() {
        _drain = new Drain();
        _mockDrainActionListener = new MockDrainActionListener();
        _drain.addListener(_mockDrainActionListener);
    }
}

```

```

@Test

void testFill() {

    Water water = new Water(new Source());

    boolean isFilled = _drain.fill(water);

    assertTrue(isFilled);

    assertSame(_mockDrainActionListener.receivedEvent.getSource(), _drain);
}

@Test

void DoubleFill() {

    Water water = new Water(new Source());

    _drain.fill(water);

    boolean isFilled = _drain.fill(water);

    assertFalse(isFilled);
}

@Test

void testToString() {

    Drain drain = new Drain();

    String expected = "d";

    String actual = drain.toString();

    assertEquals(expected, actual);
}

private class MockDrainActionListener implements DrainActionListener {

    public DrainActionEvent receivedEvent;

    @Override

    public void filled(DrainActionEvent event) {

        receivedEvent = event;
    }
}

```

```

    }

}

//Тест поля

public class FieldTests {

    private Field _field;

    @BeforeEach

    void setUp() {

        _field = new Field(3, 3);

    }

    @Test

    void testFieldInitialization() {

        assertNotNull(_field);

        assertEquals(3, _field.getHeight());

        assertEquals(3, _field.getWidth());

    }

    @Test

    void testSourceLoadFromFile() throws IOException {

        // prepare file

        String level = "3 3\n" +

            "s□□\n" +

            "□□□\n" +

            "d□□\n" +

            "UNIVERSAL;10;1;1;1;1\n" +

            "UNIVERSAL;10;1;1;1;1\n";

        String fileName = "test_level.txt";

        try {

            // write to file

            FileWriter fileWriter = new FileWriter(fileName);

            fileWriter.write(level);

            fileWriter.close();

        } catch (IOException e) {

```

```

        fail("Failed to create test file");
    }

    // load from file

    Field field = Field.loadFromFile(fileName);

    assertNotNull(field);

    assertEquals(3, field.getHeight());

    assertEquals(3, field.getWidth());

    // check entities

    Source source = field.getSource();

    assertNotNull(source);

    assertEquals(new Point(0, 0), source.getCell().getCoords());

    // delete file

    File file = new File(fileName);

    file.delete();
}

@Test
void testDrainLoadFromFile() throws IOException {

    // prepare file

    String level = "3 3\n" +

        "s□□\n" +

        "□□□\n" +

        "d□□\n" +

        "UNIVERSAL;10;1;1;1;1\n" +

        "UNIVERSAL;10;1;1;1;1\n";

    String fileName = "test_level.txt";

    try {

        // write to file

        FileWriter fileWriter = new FileWriter(fileName);

        fileWriter.write(level);

        fileWriter.close();

    } catch (IOException e) {

        fail("Failed to create test file");

    }
}

```

```

// load from file

Field field = Field.loadFromFile(fileName);

assertNotNull(field);

assertEquals(3, field.getHeight());

assertEquals(3, field.getWidth());

// check entities

Drain drain = field.getDrain();

assertNotNull(drain);

assertEquals(new Point(0, 2), drain.getCell().getCoords());

// delete file

File file = new File(fileName);

file.delete();
}

```

@Test

void testZeroFieldLoadFromFile() throws IOException {

```

// prepare file

String level = "0 0\n";

String fileName = "test_level.txt";

try {

    // write to file

    FileWriter fileWriter = new FileWriter(fileName);

    fileWriter.write(level);

    fileWriter.close();

} catch (IOException e) {

    fail("Failed to create test file");

}

```

```

// load from file

try {

    Field field = Field.loadFromFile(fileName);

    fail("Should throw exception");

} catch (IllegalArgumentException e) {

    // expected

}

```



```

        // delete file

        File file = new File(fileName);

        file.delete();
    }

@Test
void testNegativeFieldLoadFromFile() throws IOException {

    // prepare file

    String level = "-1 -1\n";

    String fileName = "test_level.txt";

    try {

        // write to file

        FileWriter fileWriter = new FileWriter(fileName);

        fileWriter.write(level);

        fileWriter.close();

    } catch (IOException e) {

        fail("Failed to create test file");

    }

    // load from file

    try {

        Field field = Field.loadFromFile(fileName);

        fail("Should throw exception");

    } catch (IllegalArgumentException e) {

        // expected

    }

    // delete file

    File file = new File(fileName);

    file.delete();
}

@Test
void testInvalidSymbolField() throws IOException {

    // prepare file

    String level = "3 3\n" +

        "□□□\n" +

```

```

        "□□□\n" +

        "x□□\n";

String fileName = "test_level.txt";

try {

    // write to file

    FileWriter fileWriter = new FileWriter(fileName);

    fileWriter.write(level);

    fileWriter.close();

} catch (IOException e) {

    fail("Failed to create test file");

}

// load from file

try {

    Field field = Field.loadFromFile(fileName);

    fail("Should throw exception");

} catch (IllegalArgumentException e) {

    // expected

}

// delete file

File file = new File(fileName);

file.delete();

}

@Test

void veryBigFieldLoadFromFile() throws IOException {

    // prepare file

    int xSize = 10;

    int ySize = 10;

    StringBuilder level = new StringBuilder();

    level.append(xSize).append(" ").append(ySize).append("\n");

    level.append("s");

    level.append("d");

    for (int i = 0; i < xSize-2; i++) {

        level.append("□");
    }
}

```

```

    }

    level.append("\n");

    for (int i = 1; i < ySize; i++) {
        for (int j = 0; j < xSize; j++) {
            level.append("□");
        }
        level.append("\n");
    }

    String fileName = "test_level.txt";

    try {
        // write to file

        FileWriter fileWriter = new FileWriter(fileName);

        fileWriter.write(level.toString());

        fileWriter.close();
    } catch (IOException e) {
        fail("Failed to create test file");
    }

    // load from file

    try {
        Field field = Field.loadFromFile(fileName);

        fail("Should throw exception");
    } catch (IllegalArgumentException e) {

    }

    // delete file

    File file = new File(fileName);

    file.delete();
}

@Test
void testComplexLoadFromFile() throws IOException {
    // prepare file

    String level = "5 5\n" +
        "□□d□□\n" +

```

```

        "\pp\n" +

        "\p\n" +

        "\p\n" +

        "\s\n" +

        "UNIVERSAL;10;1;1;1;1\n" +

        "UNIVERSAL;10;1;1;1;1\n" +

        "UNIVERSAL;10;0;1;1;0\n" +

        "UNIVERSAL;10;1;0;1;1\n" +

        "UNIVERSAL;10;1;0;1;0\n" +

        "UNIVERSAL;10;1;0;1;0\n";

String fileName = "test_level.txt";

try {

    // write to file

    FileWriter fileWriter = new FileWriter(fileName);

    fileWriter.write(level);

    fileWriter.close();

} catch (IOException e) {

    fail("Failed to create test file");

}

// load from file

Field field = Field.loadFromFile(fileName);

assertNotNull(field);

assertEquals(5, field.getHeight());

assertEquals(5, field.getWidth());

// check entities

Drain drain = field.getDrain();

assertNotNull(drain);

assertEquals(new Point(2, 0), drain.getCell().getCoords());

Source source = field.getSource();

assertNotNull(source);

assertEquals(new Point(1, 4), source.getCell().getCoords());

Pipe pipe = field.getPipeOnCords(new Point(1, 2));

assertNotNull(pipe);

ArrayList<Direction> pipeExpectedDirections = new ArrayList<>();

```

```

pipeExpectedDirections.add(Direction.UP);

pipeExpectedDirections.add(Direction.DOWN);

pipeExpectedDirections.add(Direction.LEFT);

for (Direction direction : getTankDirections(pipe)) {

    assertTrue(pipeExpectedDirections.contains(direction));

}

pipe = field.getPipeOnCords(new Point(1, 1));

assertNotNull(pipe);

pipeExpectedDirections = new ArrayList<>();

pipeExpectedDirections.add(Direction.RIGHT);

pipeExpectedDirections.add(Direction.DOWN);

for (Direction direction : getTankDirections(pipe)) {

    assertTrue(pipeExpectedDirections.contains(direction));

}

pipe = field.getPipeOnCords(new Point(1, 2));

assertNotNull(pipe);

pipeExpectedDirections = new ArrayList<>();

pipeExpectedDirections.add(Direction.UP);

pipeExpectedDirections.add(Direction.DOWN);

for (Direction direction : getTankDirections(pipe)) {

    assertTrue(pipeExpectedDirections.contains(direction));

}

pipe = field.getPipeOnCords(new Point(1, 3));

assertNotNull(pipe);

pipeExpectedDirections = new ArrayList<>();

pipeExpectedDirections.add(Direction.UP);

pipeExpectedDirections.add(Direction.DOWN);

for (Direction direction : getTankDirections(pipe)) {

    assertTrue(pipeExpectedDirections.contains(direction));

}

// delete file

File file = new File(fileName);

```

```

        file.delete();

    }

    @Test
    void twoSourcesOnField() throws IOException {

        // prepare file

        String level = "3 3\n" +

            "s□□\n" +

            "□□□\n" +

            "s□□\n" +

            "UNIVERSAL;10;1;0;1;0\n" +

            "UNIVERSAL;10;1;0;1;0\n";

        String fileName = "test_level.txt";

        try {

            // write to file

            FileWriter fileWriter = new FileWriter(fileName);

            fileWriter.write(level);

            fileWriter.close();

        } catch (IOException e) {

            fail("Failed to create test file");

        }

        // load from file

        try {

            Field field = Field.loadFromFile(fileName);

            fail("Should throw exception");

        } catch (IllegalArgumentException e) {

            // expected

        }

        // delete file

        File file = new File(fileName);

        file.delete();

    }

    @Test

```

```

void twoDrainsOnField() throws IOException {

    // prepare file

    String level = "3 3\n" +

        "d□□\n" +

        "□□□\n" +

        "d□□\n" +

        "UNIVERSAL;10;1;0;1;0\n" +

        "UNIVERSAL;10;1;0;1;0\n";

    String fileName = "test_level.txt";

    try {

        // write to file

        FileWriter fileWriter = new FileWriter(fileName);

        fileWriter.write(level);

        fileWriter.close();

    } catch (IOException e) {

        fail("Failed to create test file");

    }

    // load from file

    try {

        Field field = Field.loadFromFile(fileName);

        fail("Should throw exception");

    } catch (IllegalArgumentException e) {

        // expected

    }

    // delete file

    File file = new File(fileName);

    file.delete();

}

@Test

void noSourcesOnField() throws IOException {

    // prepare file

    String level = "3 3\n" +

        "□□□\n" +

        "□□□\n" +

```

```

        "d□□\n" +

        "UNIVERSAL;10;1;0;1;0\n" +

        "UNIVERSAL;10;1;0;1;0\n";

String fileName = "test_level.txt";

try {

    // write to file

    FileWriter fileWriter = new FileWriter(fileName);

    fileWriter.write(level);

    fileWriter.close();

} catch (IOException e) {

    fail("Failed to create test file");

}

// load from file

try {

    Field field = Field.loadFromFile(fileName);

    fail("Should throw exception");

} catch (IllegalArgumentException e) {

    // expected

}

// delete file

File file = new File(fileName);

file.delete();

}

@Test

void noDrainsOnField() throws IOException {

    // prepare file

    String level = "3 3\n" +

        "s□□\n" +

        "□□□\n" +

        "□□□\n" +

        "UNIVERSAL;10;1;0;1;0\n" +

        "UNIVERSAL;10;1;0;1;0\n";

    String fileName = "test_level.txt";

    try {

```



```

        // write to file

        FileWriter fileWriter = new FileWriter(fileName);

        fileWriter.write(level);

        fileWriter.close();

    } catch (IOException e) {

        fail("Failed to create test file");

    }

    // load from file

    try {

        Field field = Field.loadFromFile(fileName);

        fail("Should throw exception");

    } catch (IllegalArgumentException e) {

        // expected

    }

    // delete file

    File file = new File(fileName);

    file.delete();

}

@Test

void getPipeOnEmptyField() {

    assertNull(_field.getPipeOnCords(new Point(1, 1)));

}

@Test

void getPipeTest() {

    Cell cell = new Cell(new Point(1, 1), _field);

    Pipe pipe = new Pipe();

    cell.setEntity(pipe);

    _field.setCell(cell);

    assertEquals(pipe, _field.getPipeOnCords(new Point(1, 1)));

}

@Test

void setCell() {

```

```

        Cell cell = new Cell(new Point(1, 1), _field);

        _field.setCell(cell);

        assertEquals(cell, _field.getCell(new Point(1, 1)));
    }

    @Test
    void cellOutOfField() {

        Cell cell = new Cell(new Point(5, 5), _field);

        try {

            _field.setCell(cell);

            fail("Should throw exception");

        } catch (IllegalArgumentException e) {

            // expected

        }

    }

    @Test
    void negativeCellCords() {

        Cell cell = new Cell(new Point(-1, -1), _field);

        try {

            _field.setCell(cell);

            fail("Should throw exception");

        } catch (IllegalArgumentException e) {

            // expected

        }

    }

    @Test
    void setCellOnField() {

        Cell cell = new Cell();

        cell.setCords(new Point(1, 1));

        _field.setCell(cell);

        assertEquals(cell, _field.getCell(new Point(1, 1)));

    }

    @Test

```

```

void toStringTest() {

    String expected = "███\n███\n███\n";

    assertEquals(expected, _field.toString());

}

private ArrayList<Direction> getTankDirections(AbstractWaterTank tank) {

    ArrayList<Direction> directions = new ArrayList<>();

    for (AbstractWaterTankEnd end : tank.getEnds()) {

        directions.add(end.getDirection());

    }

    return directions;

}

}

//Тест игры

public class GameTests {

    private Game _game;

    private class MockGameActionListener implements GameActionListener {

        public boolean winEventCalled = false;

        public boolean loseEventCalled = false;

        public GameActionEvent receivedEvent;

        @Override

        public void winEvent(GameActionEvent event) {

            winEventCalled = true;

        }

        @Override

        public void loseEvent(GameActionEvent event) {

            loseEventCalled = true;

        }

        @Override

        public void waterTick(GameActionEvent event) {

```

```

    }

}

private MockGameActionListener _mockGameActionListener;

@BeforeEach

void setUp() {

    _game = new Game();

    _mockGameActionListener = new MockGameActionListener();

    _game.addListener(_mockGameActionListener);

    Water.setWaterDelay(1);

}

private void loadLevel (String level) throws Exception {

    String fileName = "test_level.txt";

    try {

        // write to file

        FileWriter fileWriter = new FileWriter(fileName);

        fileWriter.write(level);

        fileWriter.close();

    } catch (IOException e) {

        fail("Failed to create test file");

    }

    _game.getPlayer().startLevel(fileName);

    // delete file

    File file = new File(fileName);

    file.delete();

}

private void startWaterFlow() {

    _game.getPlayer().startWaterFlow();

}

@Test

void testWinEvent() throws Exception {

    String level = "3 3\n" +

```

```

        "s□□\n" +

        "p□□\n" +

        "ppd\n" +

        "UNIVERSAL;10;0;0;1;0\n" +

        "UNIVERSAL;10;0;0;0;1\n" +

        "UNIVERSAL;10;1;0;1;0\n" +

        "UNIVERSAL;10;1;1;0;0\n" +

        "UNIVERSAL;10;0;1;0;1\n";

    loadLevel(level);

    startWaterFlow();

    sleep(100);

    assertEquals(true, _mockGameActionListener.winEventCalled);
}

@Test
void testLoseEvent() throws Exception {
    String level = "3 3\n" +

        "s□□\n" +

        "p□□\n" +

        "ppd\n" +

        "UNIVERSAL;10;0;0;1;0\n" +

        "UNIVERSAL;10;0;0;0;1\n" +

        "UNIVERSAL;10;1;0;1;0\n" +

        "UNIVERSAL;10;1;1;0;0\n" +

        "UNIVERSAL;10;0;0;0;0\n";

    loadLevel(level);

    startWaterFlow();

    sleep(100);

    assertEquals(true, _mockGameActionListener.loseEventCalled);
}

@Test
void rotateClockwiseTest() throws Exception {

```

```

String level = "3 3\n" +
    "s□□\n" +
    "p□□\n" +
    "□□d\n" +
    "UNIVERSAL;10;1;0;1;0\n" +
    "UNIVERSAL;10;1;1;0;0\n" +
    "UNIVERSAL;10;1;0;1;0\n";

loadLevel(level);

_game.getPlayer().rotateClockwise(new Point(0,1));

ArrayList<Direction> expectedDirs = new ArrayList<Direction>();

expectedDirs.add(Direction.LEFT);

expectedDirs.add(Direction.RIGHT);

ArrayList<AbstractWaterTankEnd> ends = _game.getField().getPipeOnCords( new
Point(0,1)).getEnds();

for (AbstractWaterTankEnd end : ends) {

    assertEquals(true, expectedDirs.contains(end.getDirection()));

}

}

}

//Тест трубы

public class PipeTests {

    Pipe _pipe;

    @BeforeEach

    public void setUp() {

        MaterialNode.configure();

        _pipe = new Pipe();

        _pipe.setMaterial(MaterialNode.getRoot());

        _pipe.setDiameter(new Diameter(10));

        Water.setWaterDelay(1);

    }

    @Test

```

```

    public void testAddEnd() {

        MaterialWaterTankEnd expectedEnd = new MaterialWaterTankEnd(Direction.UP,
MaterialNode.getRoot(), new Diameter(10));

        _pipe.addEnd(expectedEnd);

        assertEquals(1, _pipe.getEnds().size());

        assertEquals(expectedEnd, _pipe.getEnd(Direction.UP));

    }

    @Test

    public void testAddEnds() {

        ArrayList<AbstractWaterTankEnd> ends = new ArrayList<>();

        MaterialWaterTankEnd expectedEnd1 = new MaterialWaterTankEnd(Direction.UP,
MaterialNode.getRoot(), new Diameter(10));

        MaterialWaterTankEnd expectedEnd2 = new MaterialWaterTankEnd(Direction.DOWN,
MaterialNode.getRoot(), new Diameter(10));

        ends.add(expectedEnd1);

        ends.add(expectedEnd2);

        _pipe.addEnds(ends);

        assertEquals(2, _pipe.getEnds().size());

        assertEquals(expectedEnd1, _pipe.getEnd(Direction.UP));

        assertEquals(expectedEnd2, _pipe.getEnd(Direction.DOWN));

    }

    @Test

    public void testRemoveEnd() {

        MaterialWaterTankEnd expectedEnd = new MaterialWaterTankEnd(Direction.UP,
MaterialNode.getRoot(), new Diameter(10));

        _pipe.addEnd(expectedEnd);

        _pipe.removeEnd(expectedEnd);

        assertEquals(0, _pipe.getEnds().size());

        assertEquals(null, _pipe.getEnd(Direction.UP));

    }

    @Test

    public void testRemoveEnds() {

        ArrayList<AbstractWaterTankEnd> ends = new ArrayList<>();

        MaterialWaterTankEnd expectedEnd1 = new MaterialWaterTankEnd(Direction.UP,
MaterialNode.getRoot(), new Diameter(10));

        MaterialWaterTankEnd expectedEnd2 = new MaterialWaterTankEnd(Direction.DOWN,
MaterialNode.getRoot(), new Diameter(10));

```

```

        ends.add(expectedEnd1);

        ends.add(expectedEnd2);

        _pipe.addEnds(ends);

        _pipe.removeEnds(ends);

        assertEquals(0, _pipe.getEnds().size());

        assertEquals(null, _pipe.getEnd(Direction.UP));

        assertEquals(null, _pipe.getEnd(Direction.DOWN));
    }

    @Test

    public void testClearEnds(){

        MaterialWaterTankEnd expectedEnd1 = new MaterialWaterTankEnd(Direction.UP,
MaterialNode.getRoot(), new Diameter(10));

        MaterialWaterTankEnd expectedEnd2 = new MaterialWaterTankEnd(Direction.DOWN,
MaterialNode.getRoot(), new Diameter(10));

        _pipe.addEnd(expectedEnd1);

        _pipe.addEnd(expectedEnd2);

        _pipe.clearEnds();

        assertEquals(0, _pipe.getEnds().size());

        assertEquals(null, _pipe.getEnd(Direction.UP));

        assertEquals(null, _pipe.getEnd(Direction.DOWN));
    }

    @Test

    public void testGetConnectedFromOneDirection() throws IOException {

        String level = "3 3\n" +

            "sp\n" +

            "p\n" +

            "d\n" +

            "UNIVERSAL;10;1;1;1;1\n" +

            "UNIVERSAL;10;1;1;1;1\n" +

            "UNIVERSAL;10;1;1;1;1\n" +

            "UNIVERSAL;10;1;1;1;1\n";

        String fileName = "test_level.txt";

        try {

            // write to file

            FileWriter fileWriter = new FileWriter(fileName);

```



```

        fileWriter.write(level);

        fileWriter.close();

    } catch (IOException e) {

        fail("Failed to create test file");

    }

// load from file

Field field = Field.loadFromFile(fileName);

AbstractWaterTank start = field.getPipeOnCords(new Point(1, 1));

HashMap<Direction, AbstractWaterTank> expectedConnected = new HashMap<>();

expectedConnected.put(Direction.UP, field.getPipeOnCords(new Point(1, 0)));

HashMap<Direction, AbstractWaterTank> actualConnected = start.getConnectedWaterTanks();

assertEquals(expectedConnected.size(), actualConnected.size());

for (Direction direction : expectedConnected.keySet()) {

    assertTrue(actualConnected.containsKey(direction));

    assertEquals(expectedConnected.get(direction), actualConnected.get(direction));

}

// delete file

File file = new File(fileName);

file.delete();

}

@Test

void getConnectedFromTwoDirections() throws IOException {

    String level = "3 3\n" +

        "sp\n" +

        "p\n" +

        "dp\n" +

        "UNIVERSAL;10;1;1;1;1\n" +

        "UNIVERSAL;10;1;1;1;1\n" +

        "UNIVERSAL;10;1;1;1;1\n" +

        "UNIVERSAL;10;1;1;1;1\n" +

        "UNIVERSAL;10;1;1;1;1\n";

```

```

String fileName = "test_level.txt";

try {

    // write to file

    FileWriter fileWriter = new FileWriter(fileName);

    fileWriter.write(level);

    fileWriter.close();

} catch (IOException e) {

    fail("Failed to create test file");

}

// load from file

Field field = Field.loadFromFile(fileName);

AbstractWaterTank start = field.getPipeOnCords(new Point(1, 1));

HashMap<Direction, AbstractWaterTank> expectedConnected = new HashMap<>();

expectedConnected.put(Direction.UP, field.getPipeOnCords(new Point(1, 0)));

expectedConnected.put(Direction.DOWN, field.getPipeOnCords(new Point(1, 2)));

HashMap<Direction, AbstractWaterTank> actualConnected = start.getConnectedWaterTanks();

assertEquals(expectedConnected.size(), actualConnected.size());

for (Direction direction : expectedConnected.keySet()) {

    assertTrue(actualConnected.containsKey(direction));

    assertEquals(expectedConnected.get(direction), actualConnected.get(direction));

}

// delete file

File file = new File(fileName);

file.delete();

}

@Test

void getConnectedFromThreeDirections() throws IOException {

    String level = "3 3\n" +

        "sp\n" +

        "pp\n" +

        "dp\n" +

```

```

        "UNIVERSAL;10;1;1;1;1\n" +

        "UNIVERSAL;10;1;1;1;1\n" +

        "UNIVERSAL;10;1;1;1;1\n" +

        "UNIVERSAL;10;1;1;1;1\n" +

        "UNIVERSAL;10;1;1;1;1\n" +

        "UNIVERSAL;10;1;1;1;1\n";

String fileName = "test_level.txt";

try {

    // write to file

    FileWriter fileWriter = new FileWriter(fileName);

    fileWriter.write(level);

    fileWriter.close();

} catch (IOException e) {

    fail("Failed to create test file");

}

// load from file

Field field = Field.loadFromFile(fileName);

AbstractWaterTank start = field.getPipeOnCords(new Point(1, 1));

HashMap<Direction, AbstractWaterTank> expectedConnected = new HashMap<>();

expectedConnected.put(Direction.UP, field.getPipeOnCords(new Point(1, 0)));

expectedConnected.put(Direction.DOWN, field.getPipeOnCords(new Point(1, 2)));

expectedConnected.put(Direction.RIGHT, field.getPipeOnCords(new Point(2, 1)));

HashMap<Direction, AbstractWaterTank> actualConnected = start.getConnectedWaterTanks();

assertEquals(expectedConnected.size(), actualConnected.size());

for (Direction direction : expectedConnected.keySet()) {

    assertTrue(actualConnected.containsKey(direction));

    assertEquals(expectedConnected.get(direction), actualConnected.get(direction));

}

// delete file

File file = new File(fileName);

file.delete();

}

```

```

@Test
void getConnectedFromFourDirections() throws IOException {

    String level = "3 3\n" +

        "sp□\n" +

        "ppp\n" +

        "dp□\n" +

        "UNIVERSAL;10;1;1;1;1\n" +

        "UNIVERSAL;10;1;1;1;1\n" +

        "UNIVERSAL;10;1;1;1;1\n" +

        "UNIVERSAL;10;1;1;1;1\n" +

        "UNIVERSAL;10;1;1;1;1\n" +

        "UNIVERSAL;10;1;1;1;1\n" +

        "UNIVERSAL;10;1;1;1;1\n";

    String fileName = "test_level.txt";

    try {

        // write to file

        FileWriter fileWriter = new FileWriter(fileName);

        fileWriter.write(level);

        fileWriter.close();

    } catch (IOException e) {

        fail("Failed to create test file");

    }

    // load from file

    Field field = Field.loadFromFile(fileName);

    AbstractWaterTank start = field.getPipeOnCords(new Point(1, 1));

    HashMap<Direction, AbstractWaterTank> expectedConnected = new HashMap<>();

    expectedConnected.put(Direction.UP, field.getPipeOnCords(new Point(1, 0)));

    expectedConnected.put(Direction.DOWN, field.getPipeOnCords(new Point(1, 2)));

    expectedConnected.put(Direction.RIGHT, field.getPipeOnCords(new Point(2, 1)));

    expectedConnected.put(Direction.LEFT, field.getPipeOnCords(new Point(0, 1)));

    HashMap<Direction, AbstractWaterTank> actualConnected = start.getConnectedWaterTanks();

    assertEquals(expectedConnected.size(), actualConnected.size());
}

```

```

        for (Direction direction : expectedConnected.keySet()) {

            assertTrue(actualConnected.containsKey(direction));

            assertEquals(expectedConnected.get(direction), actualConnected.get(direction));

        }

        // delete file

        File file = new File(fileName);

        file.delete();

    }

    @Test

    void tryToGetConnectedPipesAtTheCorner() throws IOException {

        String level = "3 3\n" +

            "s□p\n" +

            "□□p\n" +

            "d□□\n" +

            "UNIVERSAL;10;1;1;1;1\n" +

            "UNIVERSAL;10;1;1;1;1\n" +

            "UNIVERSAL;10;1;1;1;1\n" +

            "UNIVERSAL;10;1;1;1;1\n";

        String fileName = "test_level.txt";

        try {

            // write to file

            FileWriter fileWriter = new FileWriter(fileName);

            fileWriter.write(level);

            fileWriter.close();

        } catch (IOException e) {

            fail("Failed to create test file");

        }

        // load from file

        Field field = Field.loadFromFile(fileName);

        AbstractWaterTank start = field.getPipeOnCords(new Point(2, 0));

        HashMap<Direction, AbstractWaterTank> expectedConnected = new HashMap<>();

        expectedConnected.put(Direction.DOWN, field.getPipeOnCords(new Point(2, 1)));
    }

```

```

HashMap<Direction, AbstractWaterTank> actualConnected = start.getConnectedWaterTanks();

assertEquals(expectedConnected.size(), actualConnected.size());

for (Direction direction : expectedConnected.keySet()) {

    assertTrue(actualConnected.containsKey(direction));

    assertEquals(expectedConnected.get(direction), actualConnected.get(direction));

}

// delete file

File file = new File(fileName);

file.delete();

}

@Test

void fillFromOneEnd() throws Exception {

    String level = "3 3\n" +

        "sp\n" +

        "   \n" +

        "d   \n" +

        "UNIVERSAL;10;1;1;1;1\n" +

        "UNIVERSAL;10;1;1;1;1\n" +

        "UNIVERSAL;10;1;1;1;1\n";

    String fileName = "test_level.txt";

    try {

        // write to file

        FileWriter fileWriter = new FileWriter(fileName);

        fileWriter.write(level);

        fileWriter.close();

    } catch (IOException e) {

        fail("Failed to create test file");

    }

    // load from file

    Field field = Field.loadFromFile(fileName);

    field.getSource().createWater().flow();

```

```

        sleep(10);

        Pipe pipe = field.getPipeOnCords(new Point(1, 0));
        assertEquals(field.getSource().getWater(), pipe.getWater());

        // delete file

        File file = new File(fileName);

        file.delete();
    }

    @Test
    void fillFromNotAnEnd() throws Exception {
        String level = "3 3\n" +
            "sp\n" +
            "\n" +
            "d\n" +
            "UNIVERSAL;10;1;1;1;1\n" +
            "UNIVERSAL;10;1;1;1;1\n" +
            "UNIVERSAL;10;1;1;1;0\n";

        String fileName = "test_level.txt";

        try {
            // write to file

            FileWriter fileWriter = new FileWriter(fileName);

            fileWriter.write(level);

            fileWriter.close();
        } catch (IOException e) {
            fail("Failed to create test file");
        }

        // load from file

        Field field = Field.loadFromFile(fileName);

        field.getSource().createWater().flow();

        sleep(10);

        Pipe pipe = field.getPipeOnCords(new Point(1, 0));
        assertEquals(null, pipe.getWater());
    }
}

```

```

        // delete file

        File file = new File(fileName);

        file.delete();
    }

    @Test

    void doubleFilled() {

        AbstractWaterTank source = new AbstractWaterTank();

        Water water1 = new Water(source);

        Water water2 = new Water(source);

        boolean isFilled1 = _pipe.fill(water1);

        boolean isFilled2 = _pipe.fill(water2);

        assertTrue(isFilled1);

        assertFalse(isFilled2);

        assertEquals(water1, _pipe.getWater());
    }

    @Test

    public void testToString() {

        _pipe.configureTankWithOneMaterial("UNIVERSAL;10;0;1;1;1", MaterialNode.getRoot());

        assertEquals("T", _pipe.toString());
    }

    @Test

    public void testRotateClockwise() {

        _pipe.addEnd(new MaterialWaterTankEnd(Direction.UP, MaterialNode.getRoot(), new
Diameter(10)));

        _pipe.addEnd(new MaterialWaterTankEnd(Direction.RIGHT, MaterialNode.getRoot(), new
Diameter(10)));

        _pipe.rotateClockwise();

        assertNull(_pipe.getEnd(Direction.UP));

        assertNotNull(_pipe.getEnd(Direction.DOWN));

        assertNotNull(_pipe.getEnd(Direction.RIGHT));
    }

```



```

    }

    @Test
    void zeroEndsRotationClockwise() {
        _pipe.rotateClockwise();

        assertEquals(0, _pipe.getEnds().size());
    }

    void allEndsPossible() {
        _pipe.addEnd(new MaterialWaterTankEnd(Direction.UP, MaterialNode.getRoot(), new
Diameter(10)));

        _pipe.addEnd(new MaterialWaterTankEnd(Direction.RIGHT, MaterialNode.getRoot(), new
Diameter(10)));

        _pipe.addEnd(new MaterialWaterTankEnd(Direction.DOWN, MaterialNode.getRoot(), new
Diameter(10)));

        _pipe.addEnd(new MaterialWaterTankEnd(Direction.LEFT, MaterialNode.getRoot(), new
Diameter(10)));

        _pipe.rotateClockwise();

        assertEquals(4, _pipe.getEnds().size());
    }

}

// Тест Игрока
public class PlayerTests {

    private Player _player;

    MockPlayerListener _mockPlayerActionListener = new MockPlayerListener();

    @BeforeEach
    void setUp() {
        _player = new Player();

        _player.addListener(_mockPlayerActionListener);
    }

    @Test
    void testRotateWhenNotActive() {

```

```

        _player.rotateClockwise(new Point(1, 5));

        assertFalse(_mockPlayerActionListener.isRotateClockwiseEventReceived);
    }

    @Test
    void testRotateClockwiseWhenActive() {
        _player.setActive(true);

        _player.rotateClockwise(new Point(1, 5));

        assertTrue(_mockPlayerActionListener.isRotateClockwiseEventReceived);

        assertEquals(new Point(1, 5), _mockPlayerActionListener.receivedEvent.cords);
    }

    @Test
    void testStartFlowWhenNotActive() {
        _player.startWaterFlow();

        assertFalse(_mockPlayerActionListener.isStartFlowEventReceived);
    }

    @Test
    void testStartFlowWhenActive() {
        _player.setActive(true);

        _player.startWaterFlow();

        assertTrue(_mockPlayerActionListener.isStartFlowEventReceived);
    }

    @Test
    void testStartLevel() {
        _player.startLevel("levelpath");

        assertTrue(_mockPlayerActionListener.isStartLevelEventReceived);

        assertEquals("levelpath", _mockPlayerActionListener.receivedEvent.levelPath);
    }

    private class MockPlayerListener implements PlayerActionListener {

        public boolean isRotateClockwiseEventReceived = false;

        public boolean isStartFlowEventReceived = false;

        public boolean isStartLevelEventReceived = false;
    }

```

```

        PlayerActionEvent receivedEvent;

        @Override

        public void rotateClockwise(PlayerActionEvent event) {

            isRotateClockwiseEventReceived = true;

            receivedEvent = event;

        }

        @Override

        public void startFlow(PlayerActionEvent event) {

            isStartFlowEventReceived = true;

            receivedEvent = event;

        }

        @Override

        public void startLevel(PlayerActionEvent event) {

            isStartLevelEventReceived = true;

            receivedEvent = event;

        }

    }

}

```

//Тест Источка

```

public class SourceTests {

    @Test

    public void testCreateWater() {

        Source source = new Source();

        Water water = source.createWater();

        assertNotNull(water);

        assertEquals(water, source.getWater());

    }

}

```

```

@Test void testCreateTwoWaters() {

    Source source = new Source();

    Water water = source.createWater();

    try {

        Water water2 = source.createWater();

        fail("Expected an exception");

    } catch (Exception e) {

    }

    assertEquals(water, source.getWater());

}

@Test
public void testToString() {

    Source source = new Source();

    String expected = "s";

    String actual = source.toString();

    assertEquals(expected, actual);

}

}

//Тест воды
public class WaterTests {

    private Field _field;

    private Field prepareField(String level) throws IOException {

        String fileName = "test_level.txt";

        try {

            // write to file

            FileWriter fileWriter = new FileWriter(fileName);

            fileWriter.write(level);

            fileWriter.close();

        } catch (IOException e) {

            fail("Failed to create test file");

        }

    }

}

```

```

    }

    // load from file
    Field field = Field.loadFromFile(fileName);

    // delete file
    File file = new File(fileName);
    file.delete();

    return field;
}

@BeforeEach
void setUp() {
    Water.setWaterDelay(1);
}

@Test
void sourceAndDrainConnected() throws IOException, InterruptedException {
    String level = "3 3\n" +
        "s□□\n" +
        "d□□\n" +
        "□□□\n" +
        "UNIVERSAL;10;1;1;1;1\n" +
        "UNIVERSAL;10;1;1;1;1\n";

    _field = prepareField(level);

    Source src = _field.getSource();
    Water water = src.createWater();
    water.flow();
    sleep(10);
    assertEquals(water, _field.getDrain().getWater());
}

@Test
void sourceAndDrainNotConnected() throws IOException, InterruptedException {

```

```

String level = "3 3\n" +

    "s□□\n" +

    "□□□\n" +

    "□□d\n" +

    "UNIVERSAL;10;1;1;1;1\n" +

    "UNIVERSAL;10;1;1;1;1\n";

_field = prepareField(level);

Source src = _field.getSource();

Water water = src.createWater();

water.flow();

sleep(100);

assertEquals(null, _field.getDrain().getWater());

assertEquals(_field.getSource(), water.getAllSteps().get(0).keySet().toArray()[0]);
}

@Test

void sourceAndDrainConnectedWithPipe() throws IOException, InterruptedException {

    String level = "3 3\n" +

        "s□□\n" +

        "p□□\n" +

        "ppd\n" +

        "UNIVERSAL;10;0;0;1;0\n" +

        "UNIVERSAL;10;0;0;0;1\n" +

        "UNIVERSAL;10;1;0;1;0\n" +

        "UNIVERSAL;10;1;1;0;0\n" +

        "UNIVERSAL;10;0;1;0;1\n";

    _field = prepareField(level);

    Source src = _field.getSource();

    Water water = src.createWater();

    water.flow();

    sleep(10);

    assertEquals(water, _field.getDrain().getWater());

    assertEquals(water, _field.getPipeOnCords(new Point(0, 1)).getWater());
}

```

```

        assertEquals(water, _field.getPipeOnCords(new Point(0, 2)).getWater());

        assertEquals(water, _field.getPipeOnCords(new Point(1, 2)).getWater());
    }

    @Test
    void waterFork() throws IOException, InterruptedException {
        String level = "3 3\n" +

            "s□□\n" +

            "ppp\n" +

            "ppd\n" +

            "UNIVERSAL;10;0;0;1;0\n" +

            "UNIVERSAL;10;0;0;0;1\n" +

            "UNIVERSAL;10;1;1;1;0\n" +

            "UNIVERSAL;10;0;1;0;1\n" +

            "UNIVERSAL;10;0;0;1;1\n" +

            "UNIVERSAL;10;1;1;0;0\n" +

            "UNIVERSAL;10;0;1;0;1\n";

        _field = prepareField(level);

        Source src = _field.getSource();

        Water water = src.createWater();

        water.flow();

        sleep(10);

        assertEquals(water, _field.getDrain().getWater());

        assertEquals(water, _field.getPipeOnCords(new Point(0, 1)).getWater());
        assertEquals(water, _field.getPipeOnCords(new Point(0, 2)).getWater());
        assertEquals(water, _field.getPipeOnCords(new Point(1, 1)).getWater());
        assertEquals(water, _field.getPipeOnCords(new Point(1, 2)).getWater());
        assertEquals(water, _field.getPipeOnCords(new Point(2, 1)).getWater());
    }

    @Test
    void noConnectedPipe() throws IOException, InterruptedException {
        String level = "3 3\n" +

            "s□□\n" +

            "p□p\n" +

```

```

        "\u0000pd\n" +

        "UNIVERSAL;10;0;0;1;0\n" +

        "UNIVERSAL;10;0;0;0;1\n" +

        "UNIVERSAL;10;1;0;0;0\n" +

        "UNIVERSAL;10;0;1;0;1\n" +

        "UNIVERSAL;10;0;1;0;1\n";

    _field = prepareField(level);

    Source src = _field.getSource();

    Water water = src.createWater();

    water.flow();

    sleep(10);

    assertEquals(null, _field.getDrain().getWater());

    assertEquals(water, _field.getPipeOnCords(new Point(0, 1)).getWater());

    assertEquals(null, _field.getPipeOnCords(new Point(2, 1)).getWater());

    assertEquals(null, _field.getPipeOnCords(new Point(1, 2)).getWater());
}

@Test
void connectedWithTheRndOfField() throws IOException, InterruptedException {
    String level = "3 3\n" +

        "s\u0000\n" +

        "p\u0000p\n" +

        "\u0000pd\n" +

        "UNIVERSAL;10;0;0;1;0\n" +

        "UNIVERSAL;10;0;0;0;1\n" +

        "UNIVERSAL;10;1;1;0;1\n" +

        "UNIVERSAL;10;0;1;0;1\n" +

        "UNIVERSAL;10;0;1;0;1\n";

    _field = prepareField(level);

    Source src = _field.getSource();

    Water water = src.createWater();

    water.flow();

    sleep(10);

```



```

        assertEquals(null, _field.getDrain().getWater());

        assertEquals(water, _field.getPipeOnCords(new Point(0, 1)).getWater());

        assertEquals(null, _field.getPipeOnCords(new Point(2, 1)).getWater());

        assertEquals(null, _field.getPipeOnCords(new Point(1, 2)).getWater());
    }

    @Test
    void waterEndFlowEventTest() throws InterruptedException {
        String level = "3 3\n" +

            "s□□\n" +

            "p□□\n" +

            "ppd\n" +

            "UNIVERSAL;10;0;0;1;0\n" +

            "UNIVERSAL;10;0;0;0;1\n" +

            "UNIVERSAL;10;1;0;1;0\n" +

            "UNIVERSAL;10;1;1;0;0\n" +

            "UNIVERSAL;10;0;1;0;1\n";

        try {
            _field = prepareField(level);
        } catch (IOException e) {
            fail("Failed to create field");
        }

        Source src = _field.getSource();

        Water water = src.createWater();

        MockWaterListener listener = new MockWaterListener();

        water.addListener(listener);

        water.flow();

        sleep(10);

        assertEquals(true, listener.isEndFlowEventRecieved);
    }

    @Test
    void stepEndEventWithForkTest() throws IOException, InterruptedException {
        String level = "3 3\n" +

            "s□□\n" +

```

```

        "ppp\n" +

        "ppd\n" +

        "UNIVERSAL;10;0;0;1;0\n" +

        "UNIVERSAL;10;0;0;0;1\n" +

        "UNIVERSAL;10;1;1;1;1\n" +

        "UNIVERSAL;10;0;1;1;1\n" +

        "UNIVERSAL;10;0;1;1;1\n" +

        "UNIVERSAL;10;1;1;0;0\n" +

        "UNIVERSAL;10;0;1;0;1\n";

    _field = prepareField(level);

    Source src = _field.getSource();

    Water water = src.createWater();

    MockWaterListener listener = new MockWaterListener();

    water.addListener(listener);

    water.flow();

    sleep(100);

    assertEquals(5, listener.stepCount);
}

@Test
void stepEndEventTest() throws IOException, InterruptedException {
    String level = "3 3\n" +

        "s□□\n" +

        "p□□\n" +

        "ppd\n" +

        "UNIVERSAL;10;0;0;1;0\n" +

        "UNIVERSAL;10;0;0;0;1\n" +

        "UNIVERSAL;10;1;0;1;0\n" +

        "UNIVERSAL;10;1;1;0;0\n" +

        "UNIVERSAL;10;0;1;0;1\n";

    _field = prepareField(level);

```

```

        Source src = _field.getSource();

        Water water = src.createWater();

        MockWaterListener listener = new MockWaterListener();

        water.addListener(listener);

        water.flow();

        sleep(10);

        assertEquals(5, listener.stepCount);
    }

    private class MockWaterListener implements WaterActionListener {

        public boolean isEndFlowEventRecieved = false;

        public int stepCount = 0;

        @Override

        public void waterEndFlow(WaterActionEvent event) {

            isEndFlowEventRecieved = true;

        }

        @Override

        public void stepEnd(WaterActionEvent event) {

            stepCount++;

        }

    }
}

```

4 Вторая итерация разработки

4.1 Функциональные требования (сценарии)

1) Сценарий «протекание воды через трубы»

1. Пользователь инициирует запуск течения Воды
2. Игрок сообщает игре начать течение воды
3. Игра запрашивает у поля исток.
4. Игра инициирует открытие крана у истока
4. Исток создает внутри Воду и запускает её течение
5. Происходит симуляция течения воды (см. прошлую итерацию)

2) Сценарий «Запрос у трубы соединенных с ней труб

1. Вода запрашивает у трубы соединенных с ней соседей
2. Труба запрашивает у своих концов их соединения
3. Конец трубы запрашивает у Трубы её соседа
4. Конец трубы запрашивает у соседа его Концы
5. Конец трубы запрашивает материал соседа
6. Конец трубы запрашивает диаметр соседа
7. Если Концы соединены, а диаметр и материал совместимы, то возвращает в ответ на запрос Трубы соединенного соседа
8. Труба в ответ на запрос Воды возвращает соединенных соседей
9. Конец сценария

2) Сценарий «Проверка совместимости материалов - материалы идентичны»

1. Первый Материал проверяет является ли он идентичным второму материалу
2. Первый материал возвращает положительный ответ на запрос
3. Конец сценария

2.1) Альтернативный сценарий « Первый материал предок второго материала» Сценарий выполняется с п. 1 сценария 2

1. Первый Материал проверяет является ли он предком второго материала
2. Первый материал возвращает положительный ответ на запрос
3. Конец сценария

2.2) Альтернативный сценарий « Второй материал предок первого материала» Сценарий выполняется с п. 1 сценария 2.1

1. Первый Материал проверяет является ли второй материал его предком
2. Первый материал возвращает положительный ответ на запрос
3. Конец сценария

2.3) Альтернативный сценарий « Материалы несовместимы» Сценарий выполняется с п. 1 сценария 2.2

4. Первый материал возвращает отрицательный ответ на запрос
5. Конец сценария

3) Сценарий «Проверка совместимости диаметров - диаметры идентичны»

1. Первый диаметр проверяет является ли он идентичным второму
2. Первый диаметр возвращает положительный ответ на запрос
3. Конец сценария

3.1) Альтернативный сценарий « Универсальный диаметр» Сценарий выполняется с п. 1 сценария 3

1. Первый диаметр проверяет является ли второй диаметр универсальным
2. Первый диаметр возвращает положительный ответ на запрос
3. Конец сценария

3.2) Альтернативный сценарий «Диаметры несовместимы» Сценарий выполняется с п. 1 сценария 3.1

1. Первый диаметр возвращает отрицательный ответ на запрос
2. Конец сценария

4) Сценарий «Вода перетекает в трубы с несовместимым Диаметром/Материалом - Труба соединена с Фитингом»

1. Вода запрашивает у первой Трубы соединенных с ней соседей
2. В ответ на запрос Воды первая Труба отправляет соединенный Фитинг
3. Вода затекает в Фитинг

4. Вода запрашивает у Фиттинга соединенных с ним соседей
5. В ответ на запрос Воды Фиттинг отправляет соединенную с ним вторую Трубу
6. Вода затекает в Трубу
7. Конец сценария

4.2 Словарь предметной области

Сущность	Знает	Умеет	Предназначена
Материал	о Материале родителя, о Материалах детей	Проверять совместимость других материалов с собой	Для создания Труб с различными характеристика ми
Диаметр		Проверять совместимость других Диаметров с собой	Для создания Труб с различными характеристика ми
Фитинг	о клетке, может знать о Воде	Состоять из нескольких материалов и диаметров	для соединения Труб с несовместимым и Материалами и Диаметрами
Конец трубы	о Трубе	получать соединенного соседа	для соединения труб

4.3 Структура программы на уровне классов

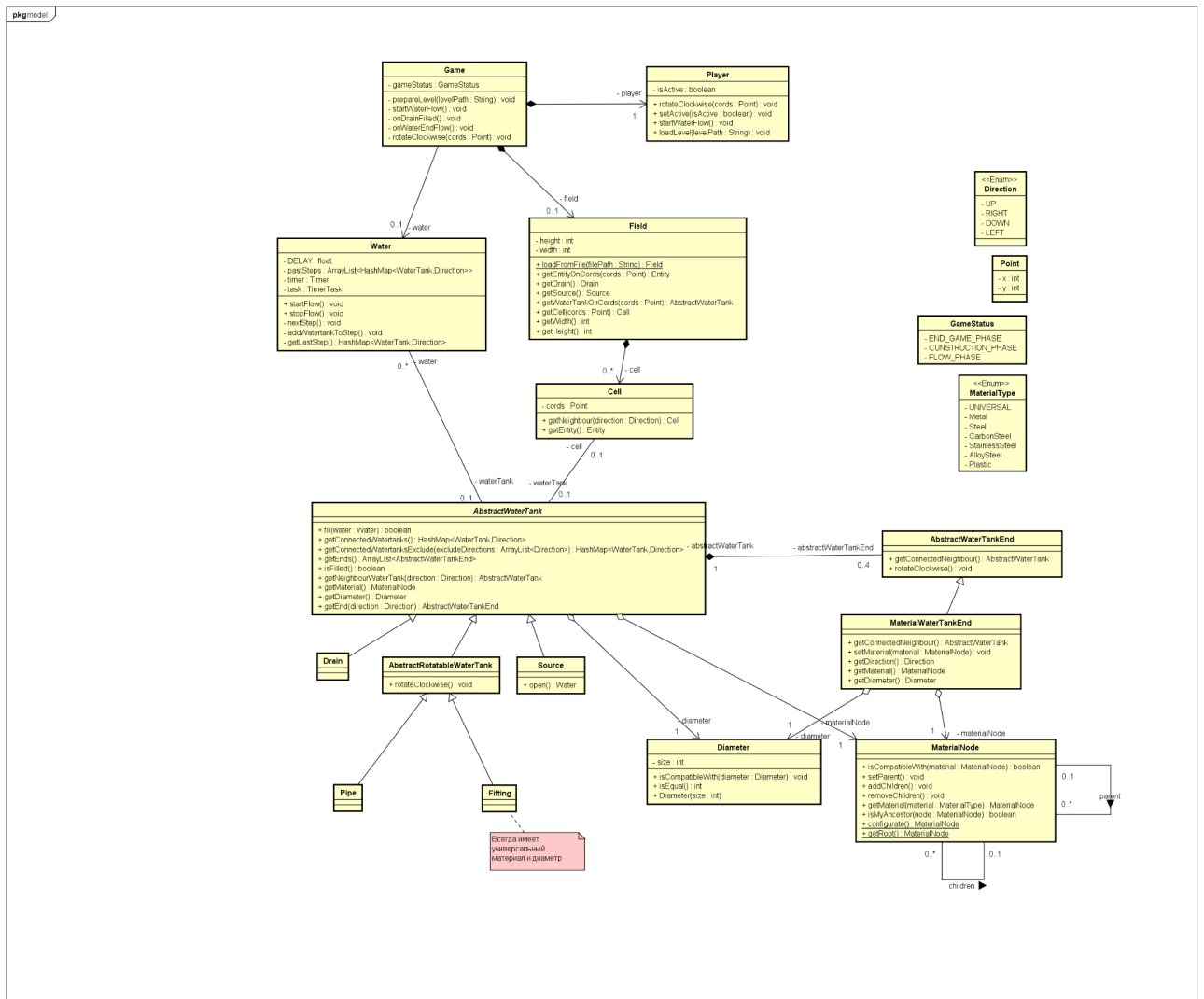
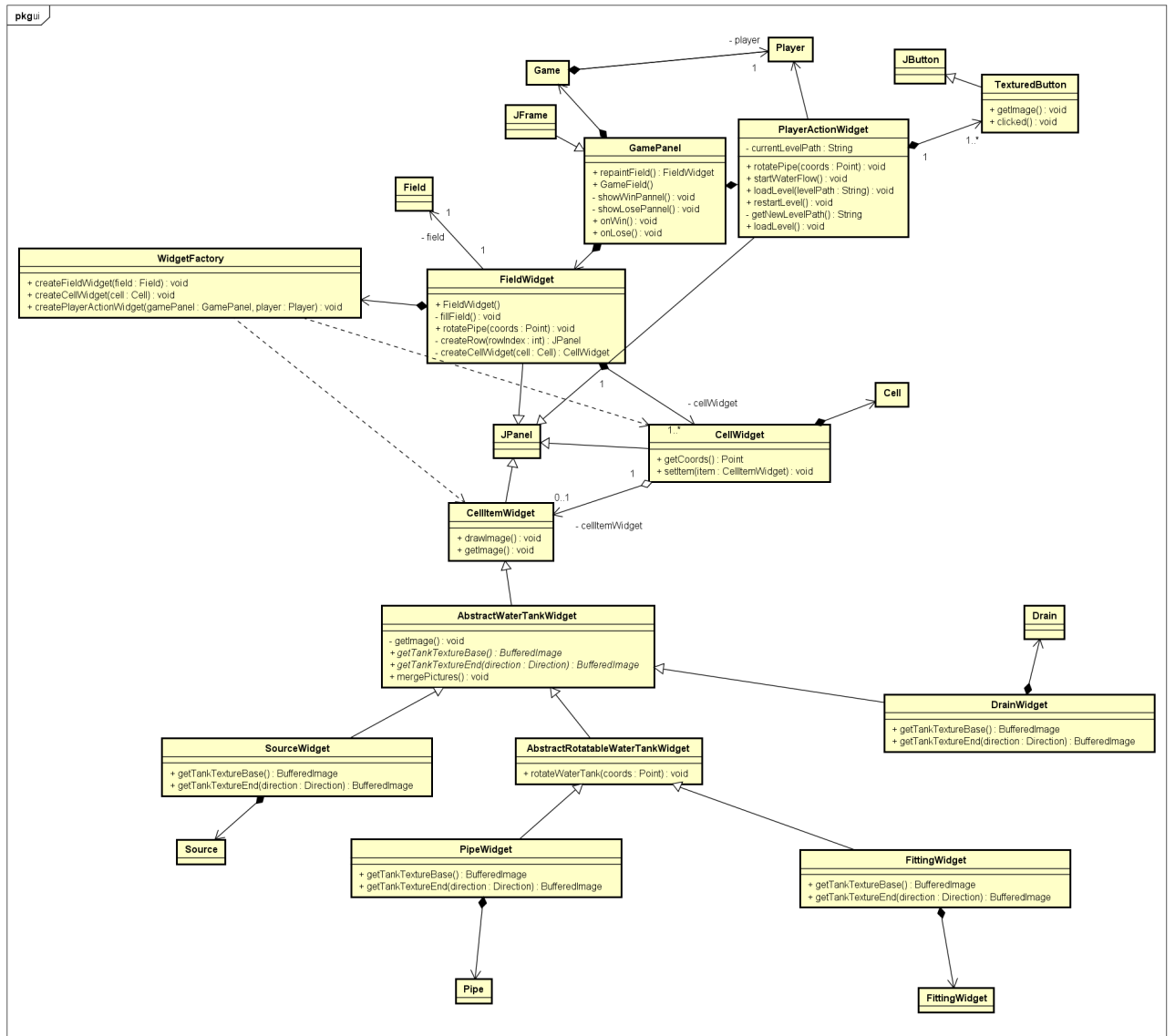


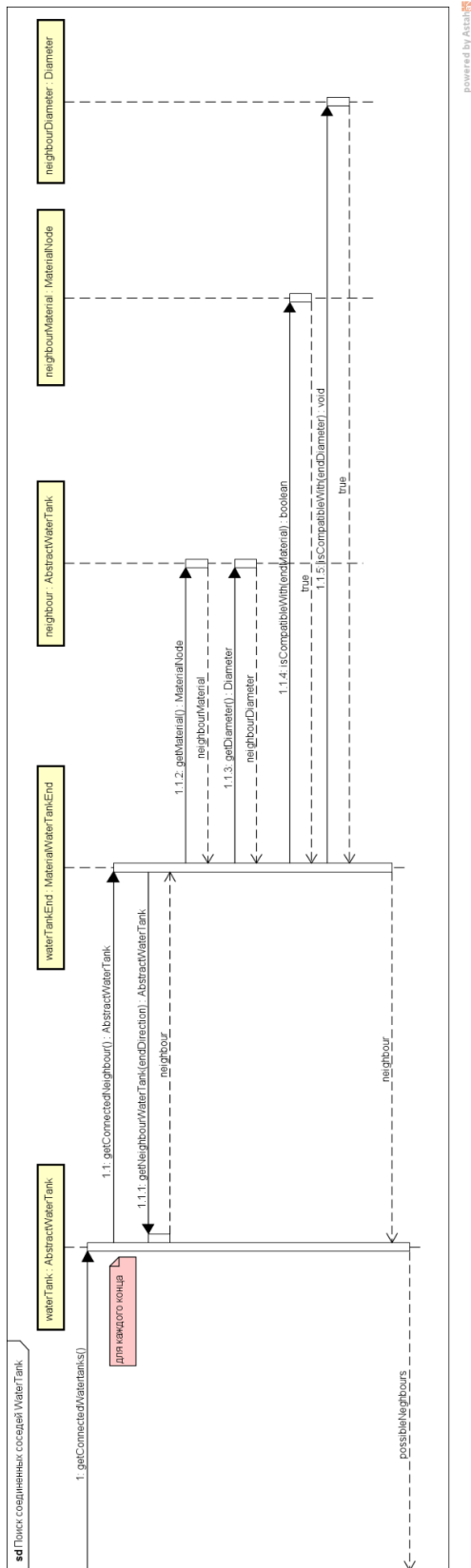
Диаграмма классов модели



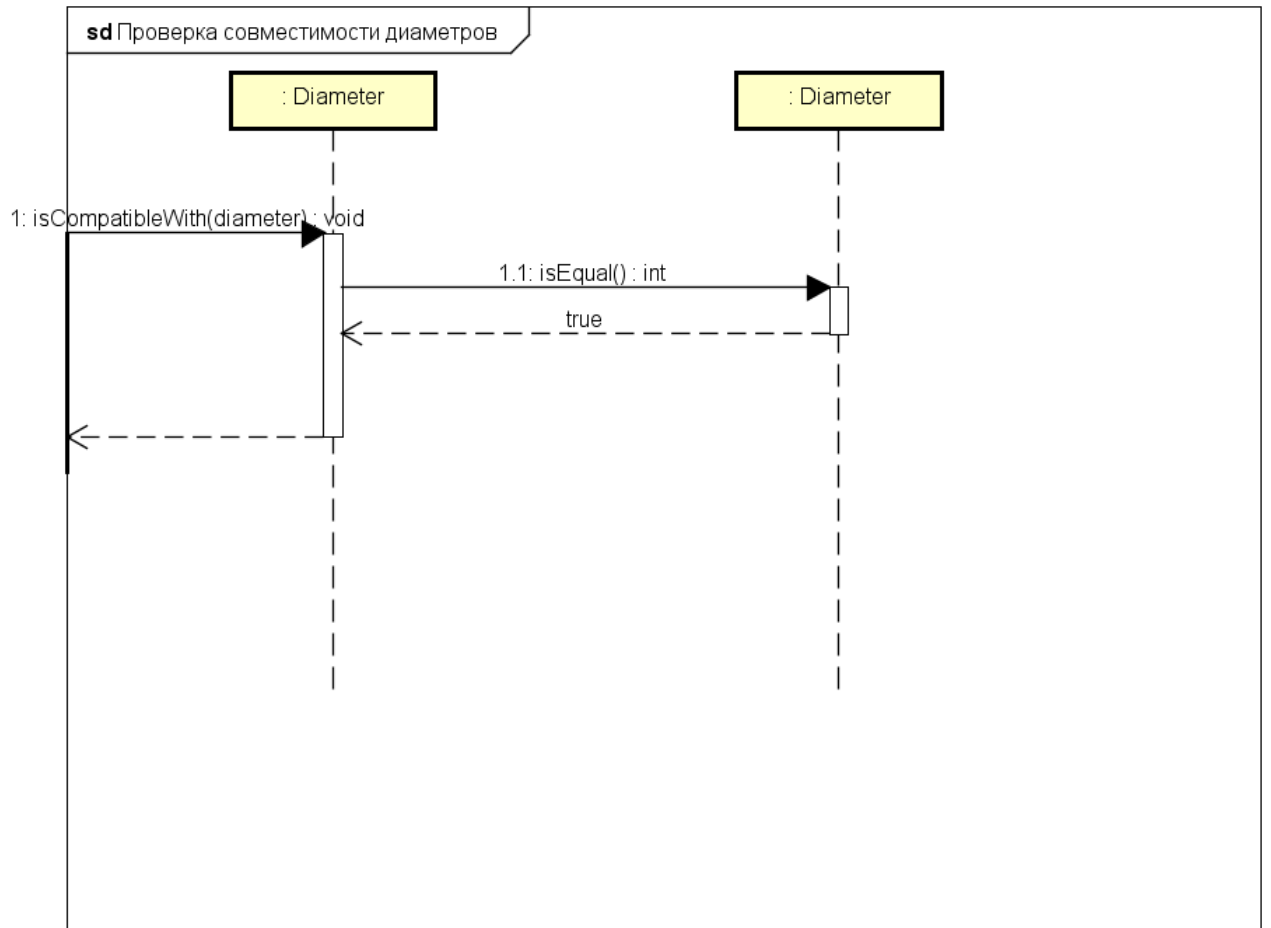
powered by Astah

Диаграмма классов UI

4.4 Типовые процессы в программе

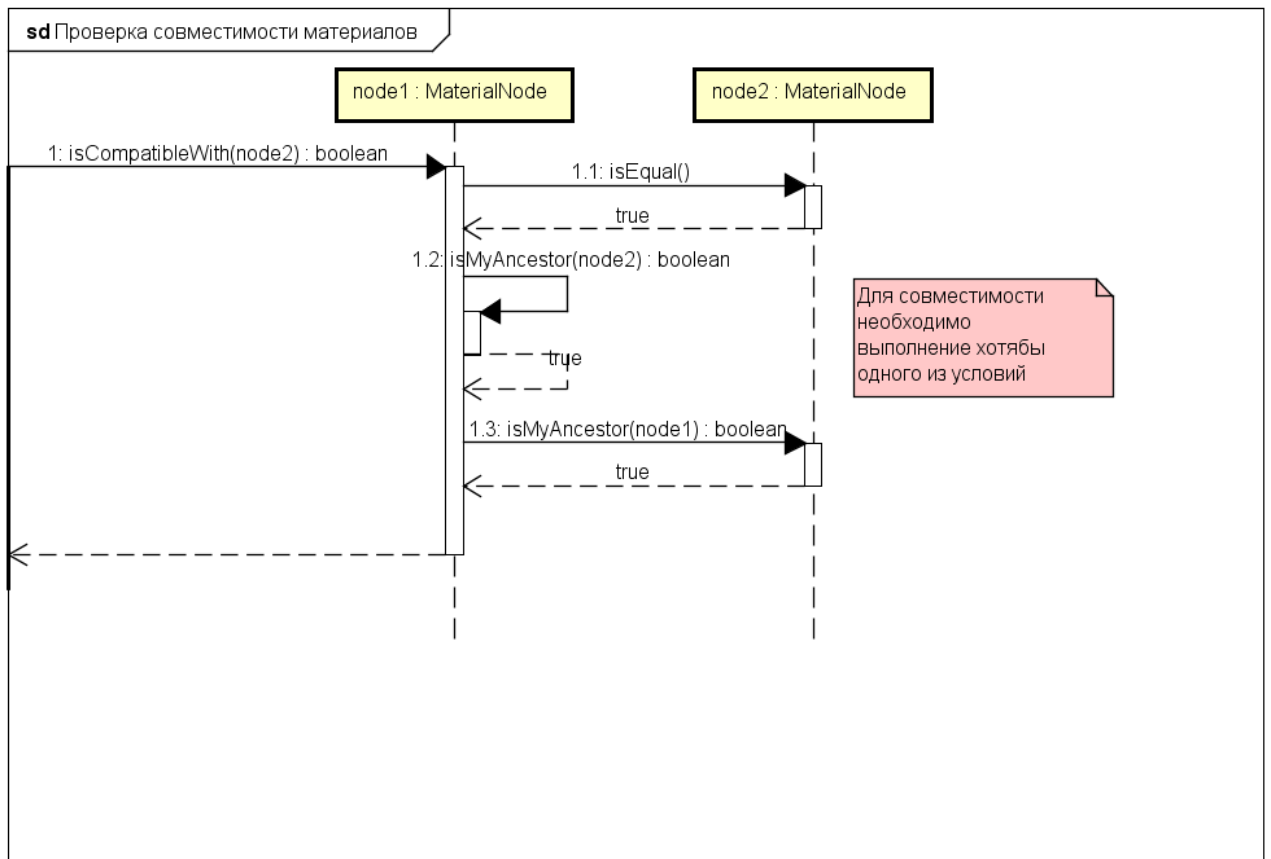


Поиск соединений резервуара



powered by Astah

Проверка совместимости диаметра



powered by Astah

Проверка совместимости материала

4.5 Человеко-машинное взаимодействие

Каждый материал и Диаметр имеет свою текстуру. Фитинги представлены отдельным объектом на поле.

Малые диаметры:

Текстура				
Материал	Универсальн ый	Металл	Пластик	Сталь

Текстура			
Материал	Карбоновая сталь	Легированная сталь	Нержавейка

Большие диаметры:

Текстура				
Материал	Универсальн ый	Металл	Пластик	Сталь


Текстура			
Материал	Карбоновая сталь	Легированная сталь	Нержавейка



Рис. 1 Центральный элемент фитинга



Рис. 2 Кнопка вспомогательной справки



Дерево материалов

Соединяться могут только одинаковые и прямые родственники (один материал является предком другого)

Могут соединяться: Металл с Нержавейкой, Сталь и Легированная сталь.

НЕ могут соединяться: Пластик и Металл, Сталь и Пластик, Карбоновая сталь и Легированная сталь

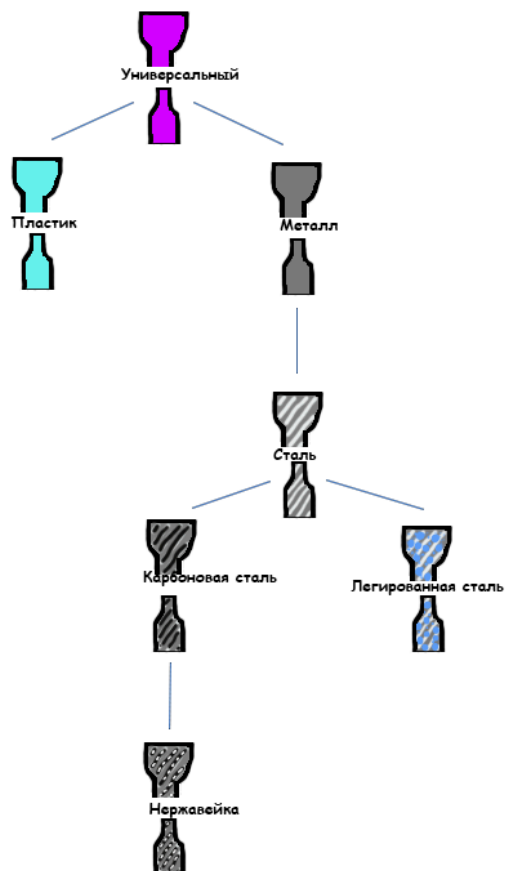


Рис. 3 Вспомогательная справка по диаметрам и материалам

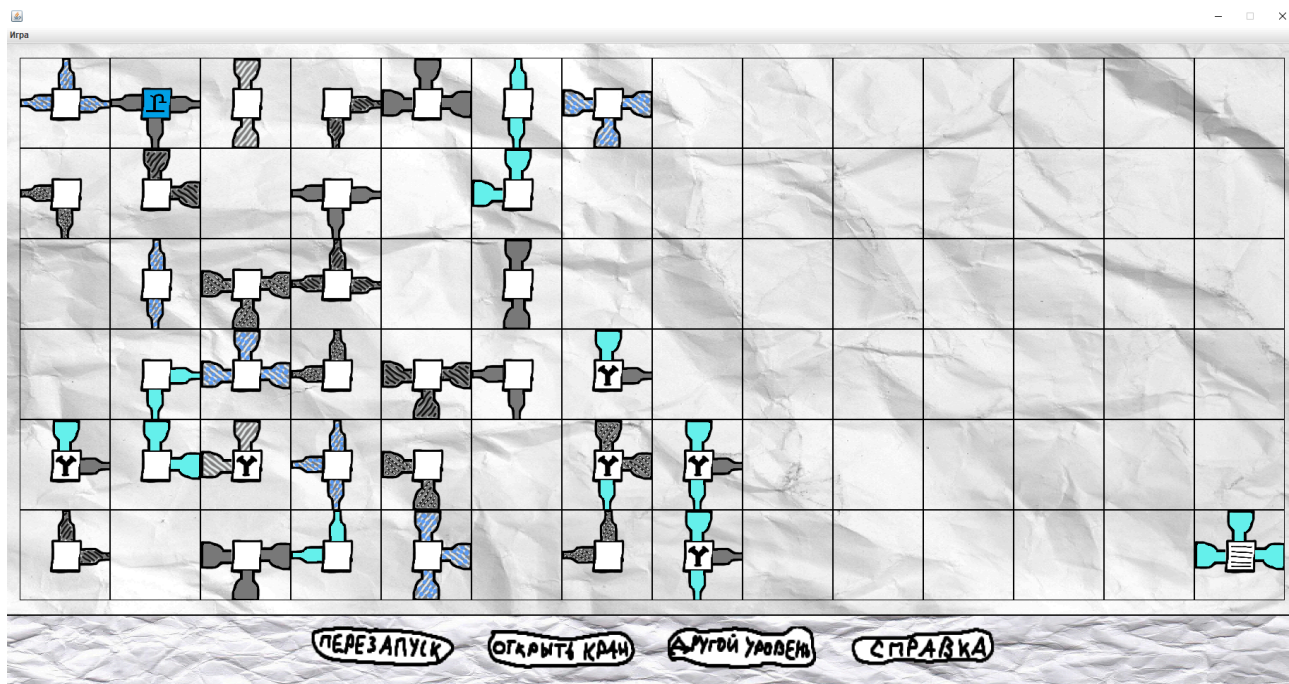


Рис. 4 Полноценный вид поля

4.6 Реализация ключевых классов

//Класс материала

```
public class MaterialNode {

    private MaterialNode _parent;

    private ArrayList<MaterialNode> _childrens = new ArrayList<MaterialNode>();

    private static MaterialNode _root = new MaterialNode(MaterialType.UNIVERSAL);

    public enum MaterialType {

        UNIVERSAL, METAL, PLASTIC, STEEL, CARBON_STEEL, STAINLESS_STEEL, ALLOY_STEEL

    }

    private MaterialType _materialType;

    public MaterialNode(MaterialType materialType) {

        _materialType = materialType;

    }

    public void setParent(MaterialNode parent) {

        _parent = parent;

    }

    public MaterialNode getParent() {

        return _parent;

    }

    public void addChild(MaterialNode child) {

        _childrens.add(child);

        child.setParent(this);

    }

    public void removeChild(MaterialNode child) {

        _childrens.remove(child);

        child.setParent(null);

    }

    public ArrayList<MaterialNode> getChildrens() {

        return _childrens;

    }

}
```

```

public MaterialType getMaterialType() {
    return _materialType;
}

public void setMaterialType(MaterialType materialType) {
    _materialType = materialType;
}

public static MaterialNode getMaterial(MaterialType materialType) {
    return _root.getChildMaterial(materialType);
}

private MaterialNode getChildMaterial(MaterialType materialType) {
    if (_materialType == materialType) {
        return this;
    }

    for (MaterialNode child : _childrens) {
        MaterialNode material = child.getChildMaterial(materialType);
        if (material != null ) {
            return material;
        }
    }
    return null;
}

public boolean isThisMyAncestor(MaterialNode ancestor) {
    if (_parent == null) {
        return false;
    }
    if (_parent == ancestor) {
        return true;
    }
    return _parent.isThisMyAncestor(ancestor);
}

public boolean isCompatible(MaterialNode other) {

```

```

        if (_materialType == other.getMaterialType()) {

            return true;

        }

        if (isThisMyAncestor(other) || other.isThisMyAncestor(this)) {

            return true;

        }

        return false;
    }

    public static MaterialNode configure() {

        MaterialNode metal = new MaterialNode(MaterialType.METAL);

        MaterialNode plastic = new MaterialNode(MaterialType.PLASTIC);

        MaterialNode steel = new MaterialNode(MaterialType.STEEL);

        MaterialNode carbonSteel = new MaterialNode(MaterialType.CARBON_STEEL);

        MaterialNode stainlessSteel = new MaterialNode(MaterialType.STAINLESS_STEEL);

        MaterialNode alloySteel = new MaterialNode(MaterialType.ALLOY_STEEL);

        _root.addChild(metal);

        _root.addChild(plastic);

        metal.addChild(steel);

        steel.addChild(carbonSteel);

        carbonSteel.addChild(stainlessSteel);

        steel.addChild(alloySteel);

        return _root;
    }

    @Override
    public String toString() {

        return _materialType.toString();

    }

    public static MaterialNode getRoot() {

```

```

        return _root;
    }
}

//Класс диаметра
public class Diameter {
    private int _value;

    public Diameter(int value) {
        if (value < 0) {
            throw new IllegalArgumentException("Diameter value cannot be negative");
        }
        _value = value;
    }

    public int getValue() {
        return _value;
    }

    public boolean isCompatible(Diameter diameter) {
        return diameter.getValue() == _value || diameter.getValue() == 0 || _value == 0;
    }

    @Override
    public String toString() {
        return String.valueOf(_value);
    }
}

//Класс конца резервуара с материалами
public class MaterialWaterTankEnd extends AbstractWaterTankEnd {
    MaterialNode _material;
    Diameter _diameter;

    public MaterialWaterTankEnd(Direction direction, MaterialNode material, Diameter diameter) {

```

```

        super(direction);

        _material = material;

        _diameter = diameter;
    }

    @Override

    public AbstractWaterTank getConnectedNeighbour() {

        AbstractWaterTank neighbourAbstractWaterTank =
getParentWaterTank().getNeighbour(getDirection());

        if (neighbourAbstractWaterTank != null) {

            AbstractWaterTankEnd neighbourEnd =
neighbourAbstractWaterTank.getEnd(getDirection().turnAround());

            if (neighbourEnd instanceof MaterialWaterTankEnd) {

                MaterialNode neighbourMaterial = ((MaterialWaterTankEnd)
neighbourEnd).getMaterial();

                Diameter neighbourDiameter = ((MaterialWaterTankEnd) neighbourEnd).getDiameter();

                if (_material.isCompatible(neighbourMaterial) &&
_diameter.isCompatible(neighbourDiameter)){

                    return neighbourAbstractWaterTank;

                }

            }

        }

        return null;
    }

    public MaterialNode getMaterial() {

        return _material;
    }

    @Override

    public boolean setParentWaterTank(AbstractWaterTank parentAbstractWaterTank) {

        if (parentAbstractWaterTank.getMaterial().isCompatible(_material) &&
parentAbstractWaterTank.getDiameter().isCompatible(_diameter)) {

            _parentAbstractWaterTank = parentAbstractWaterTank;

            return true;

        }

        return false;
    }

```

```

@Override

public String toString() {

    String superToString = super.toString();

    return superToString + ": " + _material.toString() + "; diameter: " +
_diameter.toString();

}

public void setMaterial(MaterialNode material) {

    _material = material;

}

public void setDiameter(Diameter diameter) {

    _diameter = diameter;

}

public Diameter getDiameter() {

    return _diameter;

}

}

//Класс фиттинга
public class Fitting extends AbstractRotatableWaterTanks{

    public Fitting() {

        super();

        super.setMaterial(MaterialNode.getMaterial(MaterialNode.MaterialType.UNIVERSAL));

        super.setDiameter(new Diameter(0));

    }

@Override

public void setMaterial(MaterialNode material) {

    return;

}

}

```

```

//Тест диаметра

public class DiameterTests {

    @BeforeEach

    void setUp() {

    }

    @Test

    // Universal diameter

    void zeroValue() {

        Diameter diameter = new Diameter(0);

        assertEquals(0, diameter.getValue());

    }

    @Test

    void positiveValue() {

        Diameter diameter = new Diameter(5);

        assertEquals(5, diameter.getValue());

    }

    @Test

    void negativeValue() {

        try {

            Diameter diameter = new Diameter(-5);

            fail("Expected IllegalArgumentException");

        } catch (IllegalArgumentException e) {

        }

    }

    @Test

    void testToString() {

        Diameter diameter = new Diameter(5);

        assertEquals("5", diameter.toString());

    }

    @Test

```

```

void isCompatibleEqualDiameters() {

    Diameter diameter1 = new Diameter(5);

    Diameter diameter2 = new Diameter(5);

    assertTrue(diameter1.isCompatible(diameter2));

}

@Test

void isCompatibleDifferentDiameters() {

    Diameter diameter1 = new Diameter(5);

    Diameter diameter2 = new Diameter(6);

    assertFalse(diameter1.isCompatible(diameter2));

}

@Test

void isCompatibleUniversalDiameter() {

    Diameter diameter1 = new Diameter(5);

    Diameter diameter2 = new Diameter(0);

    assertTrue(diameter1.isCompatible(diameter2));

}

}

//Тест фиттинга

public class FittingTests {

    Fitting _fitting;

    @BeforeEach

    public void setUp() {

        MaterialNode.configure();

        _fitting = new Fitting();

        Water.setWaterDelay(1);

    }

    @Test

    public void equalMaterialEnds() {

        MaterialWaterTankEnd end1 = new MaterialWaterTankEnd(Direction.UP,
MaterialNode.getMaterial(MaterialType.METAL), new Diameter(10));

```



```

        MaterialWaterTankEnd end2 = new MaterialWaterTankEnd(Direction.DOWN,
MaterialNode.getMaterial(MaterialType.METAL), new Diameter(10));

        _fitting.addEnd(end2);

        _fitting.addEnd(end1);

        assertEquals(end1, _fitting.getEnd(Direction.UP));

        assertEquals(end2, _fitting.getEnd(Direction.DOWN));

    }

    @Test

    public void differentCompatibleMaterialsEnd() {

        MaterialWaterTankEnd end1 = new MaterialWaterTankEnd(Direction.UP,
MaterialNode.getMaterial(MaterialType.METAL), new Diameter(10));

        MaterialWaterTankEnd end2 = new MaterialWaterTankEnd(Direction.DOWN,
MaterialNode.getMaterial(MaterialType.STEEL), new Diameter(10));

        _fitting.addEnd(end2);

        _fitting.addEnd(end1);

        assertEquals(end1, _fitting.getEnd(Direction.UP));

        assertEquals(end2, _fitting.getEnd(Direction.DOWN));

    }

    @Test

    public void differentNotCompatibleMaterialsEnd() {

        MaterialWaterTankEnd end1 = new MaterialWaterTankEnd(Direction.UP,
MaterialNode.getMaterial(MaterialType.METAL), new Diameter(10));

        MaterialWaterTankEnd end2 = new MaterialWaterTankEnd(Direction.DOWN,
MaterialNode.getMaterial(MaterialType.PLASTIC), new Diameter(10));

        _fitting.addEnd(end2);

        _fitting.addEnd(end1);

        assertEquals(end1, _fitting.getEnd(Direction.UP));

        assertEquals(end2, _fitting.getEnd(Direction.DOWN));

    }

    @Test

    public void equalDiameterEnds() {

        MaterialWaterTankEnd end1 = new MaterialWaterTankEnd(Direction.UP,
MaterialNode.getMaterial(MaterialType.METAL), new Diameter(10));

        MaterialWaterTankEnd end2 = new MaterialWaterTankEnd(Direction.DOWN,
MaterialNode.getMaterial(MaterialType.METAL), new Diameter(10));

        _fitting.addEnd(end2);

        _fitting.addEnd(end1);

        assertEquals(end1, _fitting.getEnd(Direction.UP));

```

```

        assertEquals(end2, _fitting.getEnd(Direction.DOWN));
    }

    @Test

    public void differentDiameterEnds() {

        MaterialWaterTankEnd end1 = new MaterialWaterTankEnd(Direction.UP,
MaterialNode.getMaterial(MaterialType.METAL), new Diameter(10));

        MaterialWaterTankEnd end2 = new MaterialWaterTankEnd(Direction.DOWN,
MaterialNode.getMaterial(MaterialType.METAL), new Diameter(20));

        _fitting.addEnd(end2);

        _fitting.addEnd(end1);

        assertEquals(end1, _fitting.getEnd(Direction.UP));

        assertEquals(end2, _fitting.getEnd(Direction.DOWN));
    }

    @Test

    public void testAddEndToFitting() {

        MaterialWaterTankEnd expectedEnd = new MaterialWaterTankEnd(Direction.UP,
MaterialNode.getRoot(), new Diameter(10));

        _fitting.addEnd(expectedEnd);

        assertEquals(1, _fitting.getEnds().size());

        assertEquals(expectedEnd, _fitting.getEnd(Direction.UP));
    }

    @Test

    public void testAddEndsToFitting() {

        ArrayList<AbstractWaterTankEnd> ends = new ArrayList<>();

        MaterialWaterTankEnd expectedEnd1 = new MaterialWaterTankEnd(Direction.UP,
MaterialNode.getRoot(), new Diameter(10));

        MaterialWaterTankEnd expectedEnd2 = new MaterialWaterTankEnd(Direction.DOWN,
MaterialNode.getRoot(), new Diameter(10));

        ends.add(expectedEnd1);

        ends.add(expectedEnd2);

        _fitting.addEnds(ends);

        assertEquals(2, _fitting.getEnds().size());

        assertEquals(expectedEnd1, _fitting.getEnd(Direction.UP));

        assertEquals(expectedEnd2, _fitting.getEnd(Direction.DOWN));
    }

    @Test

```

```

    public void testRemoveEndFromFitting() {

        MaterialWaterTankEnd expectedEnd = new MaterialWaterTankEnd(Direction.UP,
MaterialNode.getRoot(), new Diameter(10));

        _fitting.addEnd(expectedEnd);

        _fitting.removeEnd(expectedEnd);

        assertEquals(0, _fitting.getEnds().size());

        assertEquals(null, _fitting.getEnd(Direction.UP));

    }

    @Test

    public void testRemoveEndsFromFitting(){

        ArrayList<AbstractWaterTankEnd> ends = new ArrayList<>();

        MaterialWaterTankEnd expectedEnd1 = new MaterialWaterTankEnd(Direction.UP,
MaterialNode.getRoot(), new Diameter(10));

        MaterialWaterTankEnd expectedEnd2 = new MaterialWaterTankEnd(Direction.DOWN,
MaterialNode.getRoot(), new Diameter(10));

        ends.add(expectedEnd1);

        ends.add(expectedEnd2);

        _fitting.addEnds(ends);

        _fitting.removeEnds(ends);

        assertEquals(0, _fitting.getEnds().size());

        assertEquals(null, _fitting.getEnd(Direction.UP));

        assertEquals(null, _fitting.getEnd(Direction.DOWN));

    }

    @Test

    public void testClearEndsOfFitting(){

        MaterialWaterTankEnd expectedEnd1 = new MaterialWaterTankEnd(Direction.UP,
MaterialNode.getRoot(), new Diameter(10));

        MaterialWaterTankEnd expectedEnd2 = new MaterialWaterTankEnd(Direction.DOWN,
MaterialNode.getRoot(), new Diameter(10));

        _fitting.addEnd(expectedEnd1);

        _fitting.addEnd(expectedEnd2);

        _fitting.clearEnds();

        assertEquals(0, _fitting.getEnds().size());

        assertEquals(null, _fitting.getEnd(Direction.UP));

        assertEquals(null, _fitting.getEnd(Direction.DOWN));

    }

    @Test

```

```

public void testGetConnectedFromOneDirection() throws IOException {

String level = "3 3\n" +

    "sp\n" +

    "p\n" +

    "d\n" +

    "UNIVERSAL;10;1;1;1;1\n" +

    "UNIVERSAL;10;1;1;1;1\n" +

    "UNIVERSAL;10;1;1;1;1\n" +

    "UNIVERSAL;10;1;1;1;1\n";

String fileName = "test_level.txt";

try {

    // write to file

    FileWriter fileWriter = new FileWriter(fileName);

    fileWriter.write(level);

    fileWriter.close();

} catch (IOException e) {

    fail("Failed to create test file");

}

// load from file

Field field = Field.loadFromFile(fileName);

AbstractWaterTank start = field.getPipeOnCords(new Point(1, 1));

HashMap<Direction, AbstractWaterTank> expectedConnected = new HashMap<>();

expectedConnected.put(Direction.UP, field.getPipeOnCords(new Point(1, 0)));

HashMap<Direction, AbstractWaterTank> actualConnected = start.getConnectedWaterTanks();

assertEquals(expectedConnected.size(), actualConnected.size());

for (Direction direction : expectedConnected.keySet()) {

    assertTrue(actualConnected.containsKey(direction));

    assertEquals(expectedConnected.get(direction), actualConnected.get(direction));

}

// delete file

File file = new File(fileName);

```

```

file.delete();

}

@Test

void testGetConnectedFromTwoDirections() throws IOException {

String level = "3 3\n" +

    "sp\n" +

    "p\n" +

    "dp\n" +

    "UNIVERSAL;10;1;1;1;1\n" +

    "UNIVERSAL;10;1;1;1;1\n" +

    "UNIVERSAL;10;1;1;1;1\n" +

    "UNIVERSAL;10;1;1;1;1\n" +

    "UNIVERSAL;10;1;1;1;1\n";

String fileName = "test_level.txt";

try {

    // write to file

    FileWriter fileWriter = new FileWriter(fileName);

    fileWriter.write(level);

    fileWriter.close();

} catch (IOException e) {

    fail("Failed to create test file");

}

// load from file

Field field = Field.loadFromFile(fileName);

AbstractWaterTank start = field.getPipeOnCords(new Point(1, 1));

HashMap<Direction, AbstractWaterTank> expectedConnected = new HashMap<>();

expectedConnected.put(Direction.UP, field.getPipeOnCords(new Point(1, 0)));

expectedConnected.put(Direction.DOWN, field.getPipeOnCords(new Point(1, 2)));

HashMap<Direction, AbstractWaterTank> actualConnected = start.getConnectedWaterTanks();

assertEquals(expectedConnected.size(), actualConnected.size());

for (Direction direction : expectedConnected.keySet()) {

    assertTrue(actualConnected.containsKey(direction));

```

```

        assertEquals(expectedConnected.get(direction), actualConnected.get(direction));
    }

    // delete file

    File file = new File(fileName);

    file.delete();

}

@Test

void testGetConnectedFromThreeDirections() throws IOException {

String level = "3 3\n" +

    "sp\n" +

    "pp\n" +

    "dp\n" +

    "UNIVERSAL;10;1;1;1;1\n" +

    "UNIVERSAL;10;1;1;1;1\n" +

    "UNIVERSAL;10;1;1;1;1\n" +

    "UNIVERSAL;10;1;1;1;1\n" +

    "UNIVERSAL;10;1;1;1;1\n" +

    "UNIVERSAL;10;1;1;1;1\n";

String fileName = "test_level.txt";

try {

    // write to file

    FileWriter fileWriter = new FileWriter(fileName);

    fileWriter.write(level);

    fileWriter.close();

} catch (IOException e) {

    fail("Failed to create test file");

}

// load from file

Field field = Field.loadFromFile(fileName);

AbstractWaterTank start = field.getPipeOnCords(new Point(1, 1));

HashMap<Direction, AbstractWaterTank> expectedConnected = new HashMap<>();

expectedConnected.put(Direction.UP, field.getPipeOnCords(new Point(1, 0)));

expectedConnected.put(Direction.DOWN, field.getPipeOnCords(new Point(1, 2)));

```

```

expectedConnected.put(Direction.RIGHT, field.getPipeOnCords(new Point(2, 1)));

HashMap<Direction, AbstractWaterTank> actualConnected = start.getConnectedWaterTanks();
assertEquals(expectedConnected.size(), actualConnected.size());

for (Direction direction : expectedConnected.keySet()) {
    assertTrue(actualConnected.containsKey(direction));
    assertEquals(expectedConnected.get(direction), actualConnected.get(direction));
}

// delete file
File file = new File(fileName);
file.delete();
}

@Test
void testGetConnectedFromFourDirections() throws IOException {
String level = "3 3\n" +
    "sp\n" +
    "ppp\n" +
    "dp\n" +
    "UNIVERSAL;10;1;1;1;1\n" +
    "UNIVERSAL;10;1;1;1;1\n" +
    "UNIVERSAL;10;1;1;1;1\n" +
    "UNIVERSAL;10;1;1;1;1\n" +
    "UNIVERSAL;10;1;1;1;1\n" +
    "UNIVERSAL;10;1;1;1;1\n" +
    "UNIVERSAL;10;1;1;1;1\n";
String fileName = "test_level.txt";
try {
    // write to file
    FileWriter fileWriter = new FileWriter(fileName);
    fileWriter.write(level);
    fileWriter.close();
} catch (IOException e) {
    fail("Failed to create test file");
}
}

```

```

// load from file

Field field = Field.loadFromFile(fileName);

AbstractWaterTank start = field.getPipeOnCords(new Point(1, 1));

HashMap<Direction, AbstractWaterTank> expectedConnected = new HashMap<>();

expectedConnected.put(Direction.UP, field.getPipeOnCords(new Point(1, 0)));

expectedConnected.put(Direction.DOWN, field.getPipeOnCords(new Point(1, 2)));

expectedConnected.put(Direction.RIGHT, field.getPipeOnCords(new Point(2, 1)));

expectedConnected.put(Direction.LEFT, field.getPipeOnCords(new Point(0, 1)));

HashMap<Direction, AbstractWaterTank> actualConnected = start.getConnectedWaterTanks();

assertEquals(expectedConnected.size(), actualConnected.size());

for (Direction direction : expectedConnected.keySet()) {

    assertTrue(actualConnected.containsKey(direction));

    assertEquals(expectedConnected.get(direction), actualConnected.get(direction));

}

// delete file

File file = new File(fileName);

file.delete();

}

@Test

void tryToGetConnectedFittingsAtTheCorner() throws IOException {

String level = "3 3\n" +

    "s□p\n" +

    "□□p\n" +

    "d□□\n" +

    "UNIVERSAL;10;1;1;1;1\n" +

    "UNIVERSAL;10;1;1;1;1\n" +

    "UNIVERSAL;10;1;1;1;1\n" +

    "UNIVERSAL;10;1;1;1;1\n";

String fileName = "test_level.txt";

try {

    // write to file

```



```

        FileWriter fileWriter = new FileWriter(fileName);

        fileWriter.write(level);

        fileWriter.close();
    } catch (IOException e) {

        fail("Failed to create test file");
    }

    // load from file

    Field field = Field.loadFromFile(fileName);

    AbstractWaterTank start = field.getPipeOnCords(new Point(2, 0));

    HashMap<Direction, AbstractWaterTank> expectedConnected = new HashMap<>();

    expectedConnected.put(Direction.DOWN, field.getPipeOnCords(new Point(2, 1)));

    HashMap<Direction, AbstractWaterTank> actualConnected = start.getConnectedWaterTanks();

    assertEquals(expectedConnected.size(), actualConnected.size());

    for (Direction direction : expectedConnected.keySet()) {

        assertTrue(actualConnected.containsKey(direction));

        assertEquals(expectedConnected.get(direction), actualConnected.get(direction));
    }

    // delete file

    File file = new File(fileName);

    file.delete();

}

@Test
void fillFromOneEnd() throws Exception {

    String level = "3 3\n" +

        "sp\n" +

        "   \n" +

        "d   \n" +

        "UNIVERSAL;10;1;1;1;1\n" +

        "UNIVERSAL;10;1;1;1;1\n" +

        "UNIVERSAL;10;1;1;1;1\n";

```

```

String fileName = "test_level.txt";

try {

    // write to file

    FileWriter fileWriter = new FileWriter(fileName);

    fileWriter.write(level);

    fileWriter.close();

} catch (IOException e) {

    fail("Failed to create test file");

}

// load from file

Field field = Field.loadFromFile(fileName);

field.getSource().createWater().flow();

sleep(10);

Pipe pipe = field.getPipeOnCords(new Point(1, 0));

assertEquals(field.getSource().getWater(), pipe.getWater());

// delete file

File file = new File(fileName);

file.delete();

}

@Test

void fillFromNotAnEnd() throws Exception {

    String level = "3 3\n" +

        "sp\n" +

        "\n\n" +

        "d\n" +

        "UNIVERSAL;10;1;1;1;1\n" +

        "UNIVERSAL;10;1;1;1;1\n" +

        "UNIVERSAL;10;1;1;1;0\n";

    String fileName = "test_level.txt";

    try {

        // write to file

        FileWriter fileWriter = new FileWriter(fileName);

```

```

        fileWriter.write(level);

        fileWriter.close();

    } catch (IOException e) {

        fail("Failed to create test file");

    }

    // load from file

    Field field = Field.loadFromFile(fileName);

    field.getSource().createWater().flow();

    sleep(10);

    Pipe pipe = field.getPipeOnCords(new Point(1, 0));

    assertEquals(null, pipe.getWater());

    // delete file

    File file = new File(fileName);

    file.delete();

}

@Test

void doubleFilled() {

    AbstractWaterTank source = new AbstractWaterTank();

    Water water1 = new Water(source);

    Water water2 = new Water(source);

    boolean isFilled1 = _fitting.fill(water1);

    boolean isFilled2 = _fitting.fill(water2);

    assertTrue(isFilled1);

    assertFalse(isFilled2);

    assertEquals(water1, _fitting.getWater());

}

@Test

public void testRotateClockwise() {

    _fitting.addEnd(new MaterialWaterTankEnd(Direction.UP, MaterialNode.getRoot(), new
Diameter(10)));

```

```

        _fitting.addEnd(new MaterialWaterTankEnd(Direction.RIGHT, MaterialNode.getRoot(), new
Diameter(10)));

        _fitting.rotateClockwise();

        assertNull(_fitting.getEnd(Direction.UP));

        assertNotNull(_fitting.getEnd(Direction.DOWN));

        assertNotNull(_fitting.getEnd(Direction.RIGHT));

    }

    @Test

    void zeroEndsRotationClockwise() {

        _fitting.rotateClockwise();

        assertEquals(0, _fitting.getEnds().size());

    }

    void allEndsPossible() {

        _fitting.addEnd(new MaterialWaterTankEnd(Direction.UP, MaterialNode.getRoot(), new
Diameter(10)));

        _fitting.addEnd(new MaterialWaterTankEnd(Direction.RIGHT, MaterialNode.getRoot(), new
Diameter(10)));

        _fitting.addEnd(new MaterialWaterTankEnd(Direction.DOWN, MaterialNode.getRoot(), new
Diameter(10)));

        _fitting.addEnd(new MaterialWaterTankEnd(Direction.LEFT, MaterialNode.getRoot(), new
Diameter(10)));

        _fitting.rotateClockwise();

        assertEquals(4, _fitting.getEnds().size());

    }

}

//Тест материала

public class MaterialNodeTests {

    @BeforeEach

    void setUp() {

        MaterialNode.configure();

    }

}

```

```

@Test
void equalsMaterialsCompatible() {

    MaterialNode materialNode1 = MaterialNode.getMaterial(MaterialType.METAL);

    MaterialNode materialNode2 = MaterialNode.getMaterial(MaterialType.METAL);

    assertTrue(materialNode1.isCompatible(materialNode2));

    assertTrue(materialNode2.isCompatible(materialNode1));

}

@Test
void notEqualButCompatibleFirstParent() {

    MaterialNode materialNode1 = MaterialNode.getMaterial(MaterialType.METAL);

    MaterialNode materialNode2 = MaterialNode.getMaterial(MaterialType.STEEL);

    assertTrue(materialNode1.isCompatible(materialNode2));

    assertTrue(materialNode2.isCompatible(materialNode1));

}

@Test
void notEqualButCompatibleSecondParent() {

    MaterialNode materialNode1 = MaterialNode.getRoot();

    MaterialNode materialNode2 = MaterialNode.getMaterial(MaterialType.METAL);

    MaterialNode materialNode3 = MaterialNode.getMaterial(MaterialType.CARBON_STEEL);

    assertTrue(materialNode1.isCompatible(materialNode2));

    assertTrue(materialNode2.isCompatible(materialNode3));

}

@Test
void notCompatible() {

    MaterialNode materialNode1 = MaterialNode.getMaterial(MaterialType.METAL);

    MaterialNode materialNode2 = MaterialNode.getMaterial(MaterialType.PLASTIC);

    assertFalse(materialNode1.isCompatible(materialNode2));

    assertFalse(materialNode2.isCompatible(materialNode1));

}

@Test
void universalMaterial() {

    MaterialNode materialNode1 = MaterialNode.getMaterial(MaterialType.UNIVERSAL);

    for (MaterialType materialType : MaterialType.values()) {

```

```

        MaterialNode materialNode2 = MaterialNode.getMaterial(materialType);

        assertTrue(materialNode1.isCompatible(materialNode2));

        assertTrue(materialNode2.isCompatible(materialNode1));

    }

}

```

//Тест конца с материалом

```

public class MaterialEndTests {

    Field _field;

    @BeforeEach
    void setUp() {

        Water.setWaterDelay(1);

    }

    @Test
    public void equalMaterialEnd() throws IOException, InterruptedException {

        String level = "3 3\n" +

            "s□□\n" +

            "f□□\n" +

            "□□d\n" +

            "METAL;10;0;0;1;0\n" +

            "METAL;10;0;0;0;1\n" +

            "UNIVERSAL;0;METAL;10;null;10;null;10;null;10\n";

        _field = prepareField(level);

        Source src = _field.getSource();

        Water water = src.createWater();

        water.flow();

        sleep(100);

        Fitting fitting = (Fitting) _field.getCell(new Point(0, 1)).getEntity();

        assertNotNull(fitting.getWater());

    }

    @Test

```

```

public void differentCompatibleMaterialEnd() throws IOException, InterruptedException {

    String level = "3 3\n" +

        "s□□\n" +

        "f□□\n" +

        "□□d\n" +

        "METAL;10;0;0;1;0\n" +

        "METAL;10;0;0;0;1\n" +

        "UNIVERSAL;0;STEEL;10;null;10;null;10;null;10\n";

    _field = prepareField(level);

    Source src = _field.getSource();

    Water water = src.createWater();

    water.flow();

    sleep(100);

    Fitting fitting = (Fitting) _field.getCell(new Point(0, 1)).getEntity();

    assertNotNull(fitting.getWater());

}

@Test

public void differentNotCompatibleMaterialEnd() throws IOException, InterruptedException {

    String level = "3 3\n" +

        "s□□\n" +

        "f□□\n" +

        "□□d\n" +

        "METAL;10;0;0;1;0\n" +

        "METAL;10;0;0;0;1\n" +

        "UNIVERSAL;0;PLASTIC;10;null;10;null;10;null;10\n";

    _field = prepareField(level);

    Source src = _field.getSource();

    Water water = src.createWater();

    water.flow();

    sleep(100);

    Fitting fitting = (Fitting) _field.getCell(new Point(0, 1)).getEntity();

    assertNull(fitting.getWater());
}

```

```

}

@Test
public void differentEqualDiameterEnd() throws IOException, InterruptedException {
    String level = "3 3\n" +
        "s□□\n" +
        "f□□\n" +
        "□□d\n" +
        "UNIVERSAL;10;0;0;1;0\n" +
        "UNIVERSAL;10;0;0;0;1\n" +
        "UNIVERSAL;0;UNIVERSAL;10>null;10>null;10>null;10\n";

    _field = prepareField(level);

    Source src = _field.getSource();
    Water water = src.createWater();
    water.flow();

    sleep(100);

    Fitting fitting = (Fitting) _field.getCell(new Point(0, 1)).getEntity();
    assertNotNull(fitting.getWater());
}

@Test
public void differentNotEqualDiameterEnd() throws IOException, InterruptedException {
    String level = "3 3\n" +
        "s□□\n" +
        "f□□\n" +
        "□□d\n" +
        "UNIVERSAL;10;0;0;1;0\n" +
        "UNIVERSAL;10;0;0;0;1\n" +
        "UNIVERSAL;0;UNIVERSAL;20>null;10>null;10>null;10\n";

    _field = prepareField(level);

    Source src = _field.getSource();
    Water water = src.createWater();
    water.flow();

```



```

        sleep(100);

        Fitting fitting = (Fitting) _field.getCell(new Point(0, 1)).getEntity();

        assertNull(fitting.getWater());
    }

    @Test
    public void differentUniversalDiameterEnd() throws IOException, InterruptedException {
        String level = "3 3\n" +
            "s□□\n" +
            "f□□\n" +
            "□□d\n" +
            "UNIVERSAL;10;0;0;1;0\n" +
            "UNIVERSAL;10;0;0;0;1\n" +
            "UNIVERSAL;0;UNIVERSAL;0>null;10>null;10>null;10\n";

        _field = prepareField(level);

        Source src = _field.getSource();
        Water water = src.createWater();
        water.flow();

        sleep(100);

        Fitting fitting = (Fitting) _field.getCell(new Point(0, 1)).getEntity();
        assertNotNull(fitting.getWater());
    }

    private Field prepareField(String level) throws IOException {
        String fileName = "test_level.txt";

        try {
            // write to file

            FileWriter fileWriter = new FileWriter(fileName);

            fileWriter.write(level);

            fileWriter.close();
        } catch (IOException e) {
            fail("Failed to create test file");
        }

        // load from file
    }

```

```
Field field = Field.loadFromFile(fileName);

// delete file
File file = new File(fileName);
file.delete();

return field;
}
}
```

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. <https://stackoverflow.com>
3. <https://habr.com/>