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 №17)

Студент: Ковалева А. А.

Группа: ПрИн-367

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

Руководитель проекта, нормоконтроллер _____ Литовкин Д.В.

Волгоград, 2023 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: Ковалева А. А.
Группа: ПрИн-367
1. Тема: «Проектирование и реализация программы с использованием объектно-ориентированного подхода» (индивидуальное задание – вариант №17)
Утверждена приказом от «___»_____2023г. №
2. Срок представления работы к защите «___»_____2023г.
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 «____»_____2023г.
Руководитель проекта: _____ Литовкин Д.В.
Задание принял к исполнению: _____ Ковалева А. А.

«___»_____2023г.

Содержание

1 Формулировка задания

Игра "Танки".

- Ира ведется пошагово двумя игроками;
- Игрок может пропускать свой ход, т.е. ничего не делать;
- На поле NxM клеток находятся танк игрока, танк противника, штабы и препятствия;
- Танк игрока имеет несколько жизней;
- Каждый танк может стрелять неограниченное количество раз, но не чаще 1 раза за N ходов;
- Цель игры - уничтожить танк противника ИЛИ штаб противника;
- Препятствиями могут быть:
    - кирпичная стена, которая может быть разрушена;
    - вода - это непроходимое препятствие.

Дополнительные требования:

- полет снаряда должен визуализироваться;
- разрушение объекта должно визуализироваться.

Подвариант 2: необходимо предусмотреть в программе **точки расширения**, используя которые можно реализовать вариативную часть программы (в дополнение к базовой функциональности).

Вариативность: предусмотреть возможность создания новых видов снарядов, отличающихся областью поражения, траекторией и дальностью полета.

НЕ изменяя ранее созданные классы, а используя **точки расширения**, **реализовать**: умный снаряд, который летит в заданную точку по кратчайшему маршруту (может облетать препятствия). Дальность полета снаряда ограничена. Область поражения - 4 смежных клетки.

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

Игра "Танки".

- Игра ведется пошагово двумя игроками;
- Игрок может пропускать свой ход, т.е. ничего не делать;
- На поле NxM клеток находятся танк игрока, танк противника, штабы и препятствия;
- Танк игрока имеет несколько жизней;
- Каждый танк может стрелять неограниченное количество раз, но не чаще 1 раза за N ходов;
- Цель игры - уничтожить танк противника ИЛИ штаб противника;
- Препятствиями могут быть:
    - кирпичная стена, которая может быть разрушена;
    - вода - это непроходимое препятствие.

Дополнительные требования:

- полет снаряда должен визуализироваться;
- разрушение объекта должно визуализироваться.

3.2 Функциональные требования (сценарии)

1) Сценарий «Играть»

1. Пользователь инициирует начало игры
2. Игра создает при помощи Генерации поля Поле из Ячеек и размещает на нём два Танка и два Штаба к ним, Стены и Воду
3. Игра запрашивает у Поля Танки, которые находятся на нем
4. Игра случайным образом выбирает активный Танк
5. Делать
    - 5.1. По указанию пользователя Танк перемещается на соседнюю Ячейку, стреляет в заданном направлении или пропускает ход
    - 5.2. Игра запрашивает у Поля танки, которые находятся на нем
    - 5.3. Игра делает активным следующий живой Танк, который располагается на Поле

    Пока на поле есть хотя бы один живой Танк
6. Игра считает победителем единственный Танк, который уничтожил Танк противника или Штаб противника

2) Сценарий «Генерация поля создает поле из ячеек и размещает на нем два танка, два штаба, стены и лужи»

1. Игра инициирует создание поля размером NxM ячеек посредством генерации поля

2. Генерация поля создает и расставляет стены внутри поля

3. Генерация поля создает и расставляет воду внутри поля

4. Генерация поля создает два танка и помещает их на поле

6. Генерация поля создает и помещает два штаба на поле

3) Сценарий «Танк стреляет»

1. Пользователь хочет стрельнуть в заданном направлении

2. Танк разрешает себе стрельнуть (так как стрелять можно 1 раз за 3 хода)

3. Танк стреляет в заданном направлении

4. Танк обнуляет себе возможность стрелять

5. Снаряд летит до первой несвободной ячейки или края поля

6. Снаряд уничтожает препятствие, а потом и себя

7. Препятствием оказывается штаб, поэтому игра заканчивается с определением победителя

3.1) Альтернативный сценарий «Танк не может стрельнуть»

1. Сценарий начинается после пункта 1

2. Танк запрещает себе стрельнуть (так как стрелять можно 1 раз за 3 хода) и сообщает об этом пользователю

3. Сценарий переходит к пункту 5.1 главного сценария

3.2) Альтернативный сценарий «Снаряд попадает в стену периметра»

1. Сценарий начинается после пункта 2
2. Снаряд понимает, что перед ним ячеек больше нет, и саморазрушается
3. Сценарий переходит к пункту 5.2 главного сценария

3.3) Альтернативный сценарий «Снаряд попадает в стену»

1. Сценарий начинается после пункта 7
2. Препятствием оказывается стена, поэтому снаряд ее разрушает
3. Сценарий переходит к пункту 5.2 главного сценария

3.4) Альтернативный сценарий «Снаряд попадает в танк»

1. Сценарий начинается после пункта 7
2. Препятствием оказывается танк, поэтому снаряд отнимает у него одну жизнь
3. Сценарий переходит к пункту 5.2 главного сценария

4) Сценарий «Танк перемещается на свободную ячейку»

1. Танк запрашивает у ячейки, в которой он находится, соседнюю ячейку в направлении своего движения
2. Ячейка сообщает о ячейке, с которой соседствует
3. Танк спрашивает у соседней ячейки, есть ли в ней непроходимое препятствие
4. Танк просит ячейку, в которой он находится, изъять его из нее
5. Ячейка извлекает танк из себя

6. Танк просит соседнюю ячейку поместить себя в нее
7. Ячейка помещает танк в себя, т.к. в ней нет другого танка

4.1) Альтернативный сценарий «В соседней ячейке находится непроходимое препятствие»

1. Сценарий начинается после пункта 3
2. Соседняя ячейка сообщает, что препятствие присутствует
3. Сценарий переходит к пункту 5.1 главного сценария

4.2) Альтернативный сценарий «Пропуск хода»

1. Сценарий выполняется вместо сценария 4
2. Пользователь инициирует пропуск хода
3. Сценарий переходит к пункту 5.2 главного сценария

5) Сценарий «Досрочное завершение игры»

1. Сценарий начинается в любой точке главного сценария, когда пользователь инициирует завершение игры
2. Игра завершается без определения победителя

6) Сценарий «Снаряд летит по свободной ячейке»

1. Снаряд запрашивает у ячейки, в которой он находится, соседнюю ячейку в направлении своего движения
2. Ячейка сообщает о ячейке, с которой соседствует
3. Снаряд спрашивает у соседней ячейки, есть ли в ней непроходимое препятствие

4. Снаряд просит ячейку, в которой он находится, изъять его из нее
5. Ячейка извлекает снаряд из себя
6. Снаряд просит соседнюю ячейку поместить себя в нее
7. Ячейка помещает снаряд в себя, т.к. в ней нет другого непроходимого объекта

3.3 Словарь предметной области

Игра знает о поле. Игра управляет игровым циклом: определяет очередного игрока, определяет окончание, определяет победителя.

Генерация поля создает поле и определяет начальную расстановку игровых элементов (танки, стены, лужи, штабы) на поле.

Поле - прямоугольная область, состоящая из ячеек. Позволяет получить танки, находящиеся на поле.

Ячейка - квадратная область поля. Знает о четырёх соседних ячейках. В ней может находиться по отдельности стена, лужа, штаб, танк. Она понимает, какой объект в ней находится. Ячейка может передать объекту, что его поразили.

Стена – разрушаемый объект одним снарядом, заполняет всю ячейку. Танк не может ее пройти, если не разрушит.

Вода – непроходимый объект, заполняет всю ячейку.

Штаб – разрушаемый объект одним снарядом, заполняет всю ячейку. После разрушения штаба игра заканчивается.

Снаряд – объект, которым стреляет танк в выбранном направлении. Летит до первого препятствия. Сообщает ячейке с препятствием, что он ее поразил.

Танк – объект игры, который может перемещаться в соседнюю ячейку. Также умеет стрелять снарядом 1 раз в 3 хода. Не может переходить стены и лужи. Танк может пропустить ход и остаться в исходной ячейке.

Активный танк – танк, который может совершать действие в текущий ход.

Направление – класс для обозначения направления перемещения танка. Выбирается пользователем с клавиатуры.

3.4 Структура программы на уровне классов

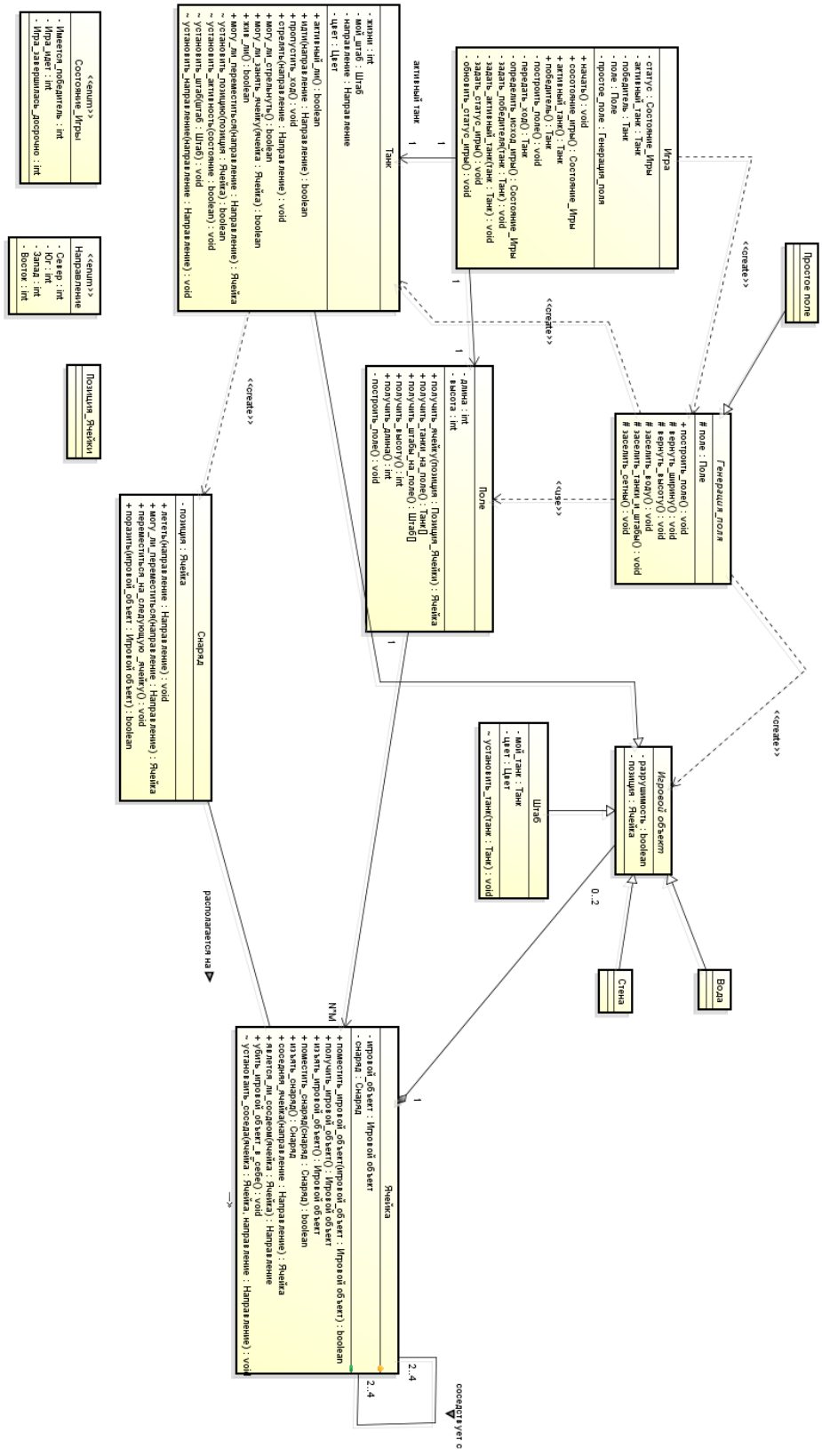


Диаграмма классов вычислительной модели

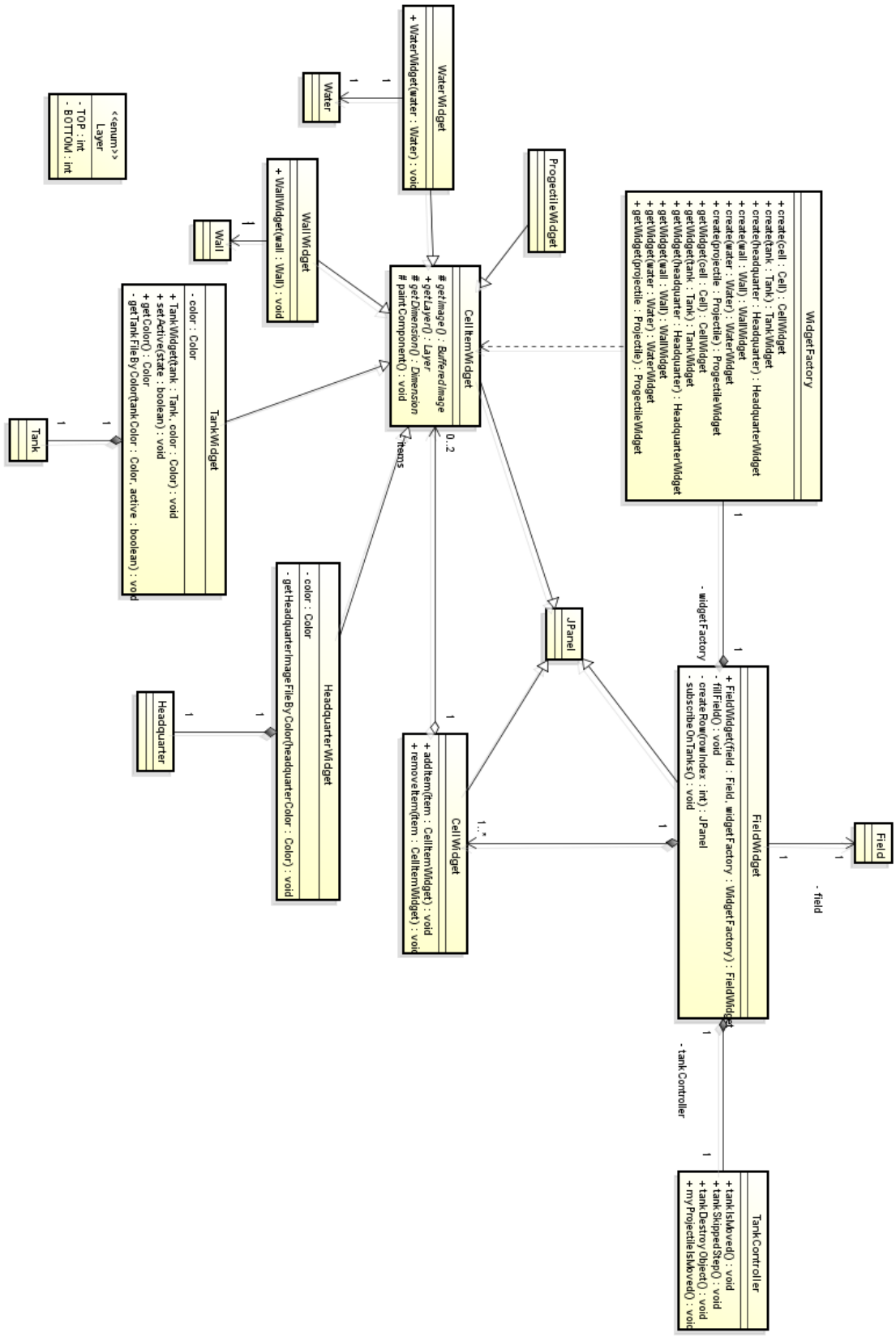
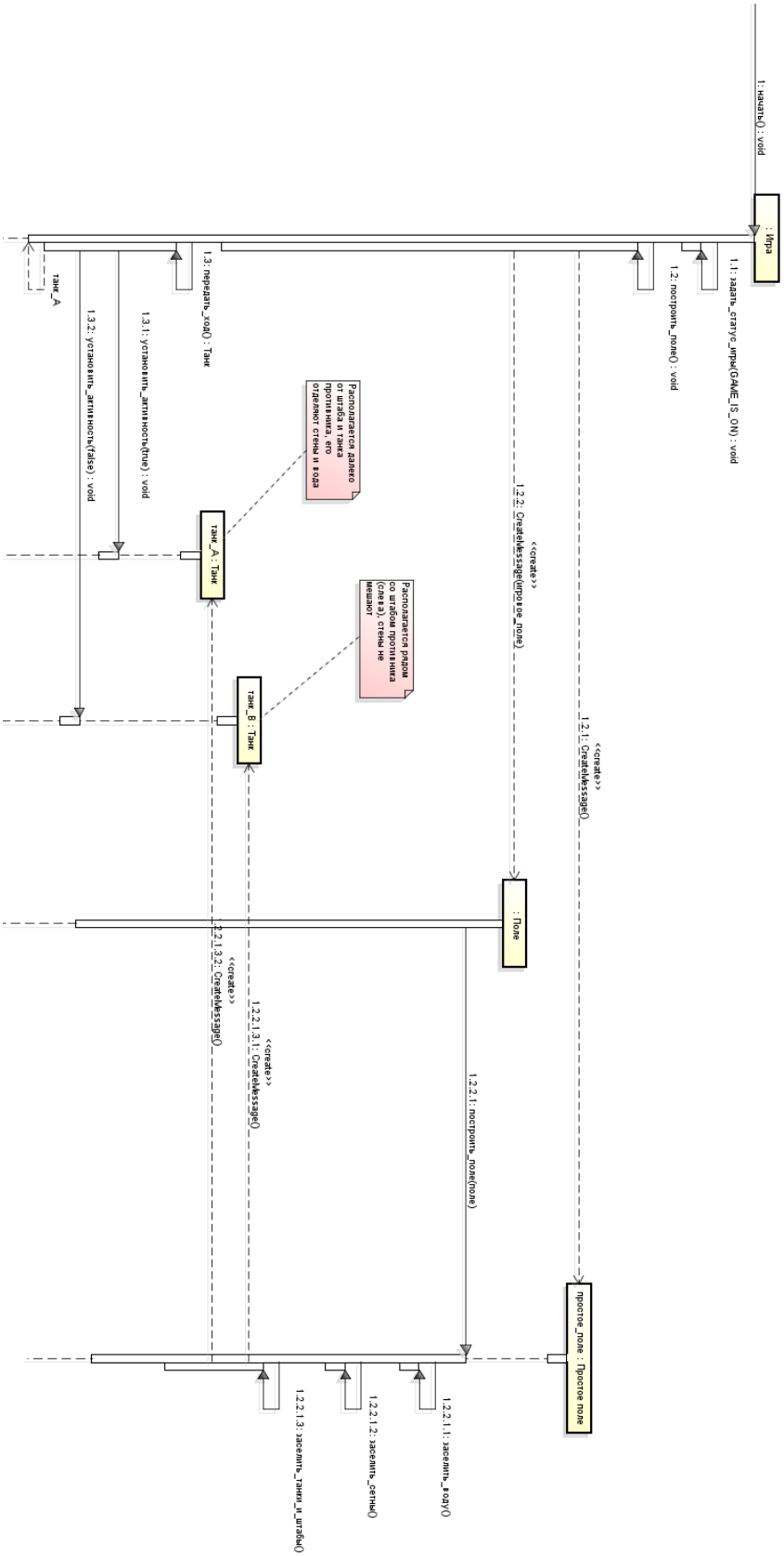
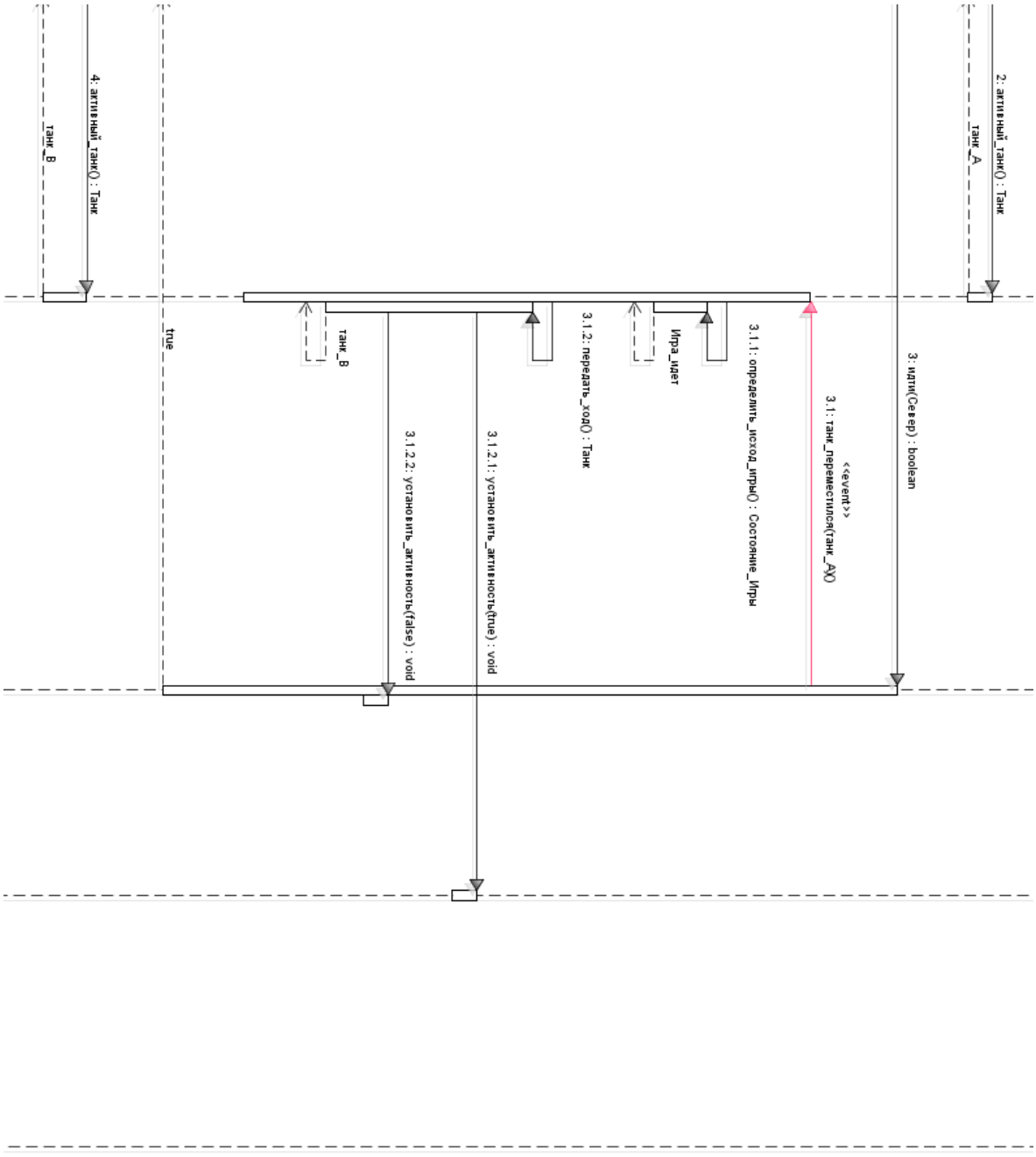
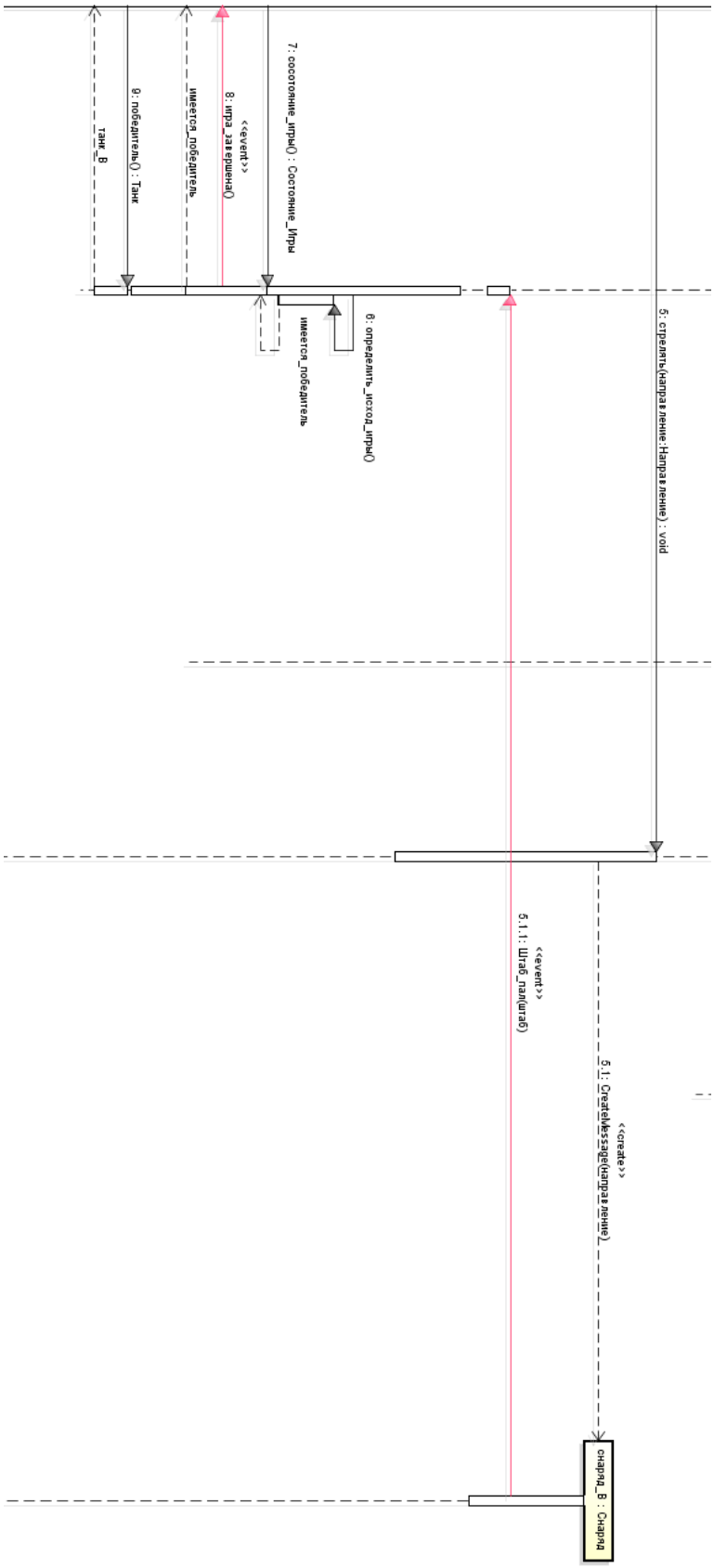


Диаграмма классов представления

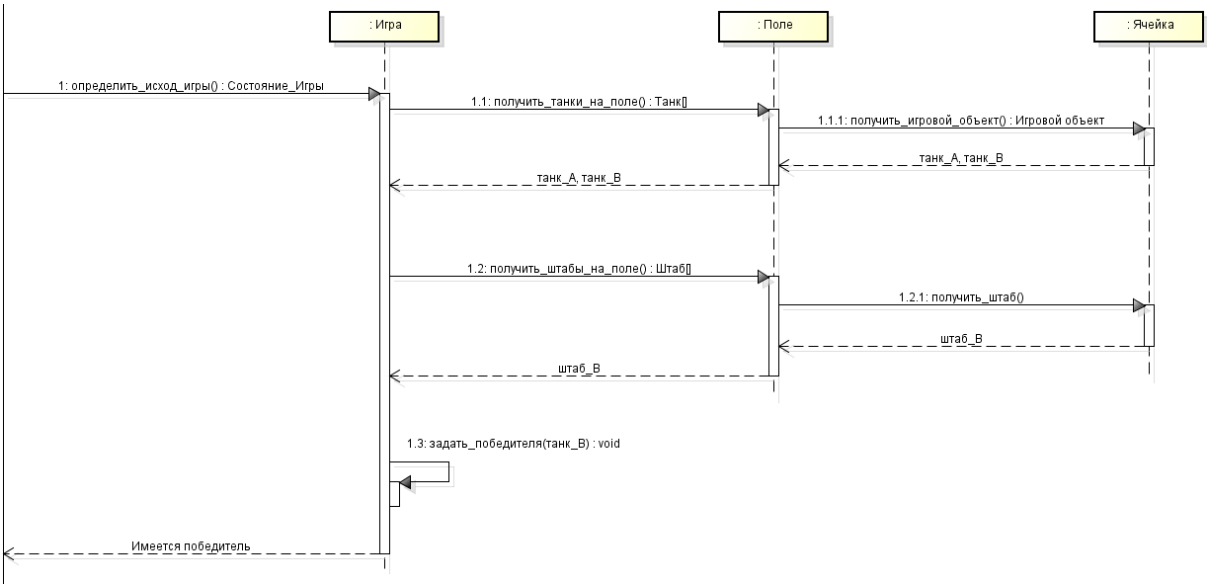
3.5 Типовые процессы в программе



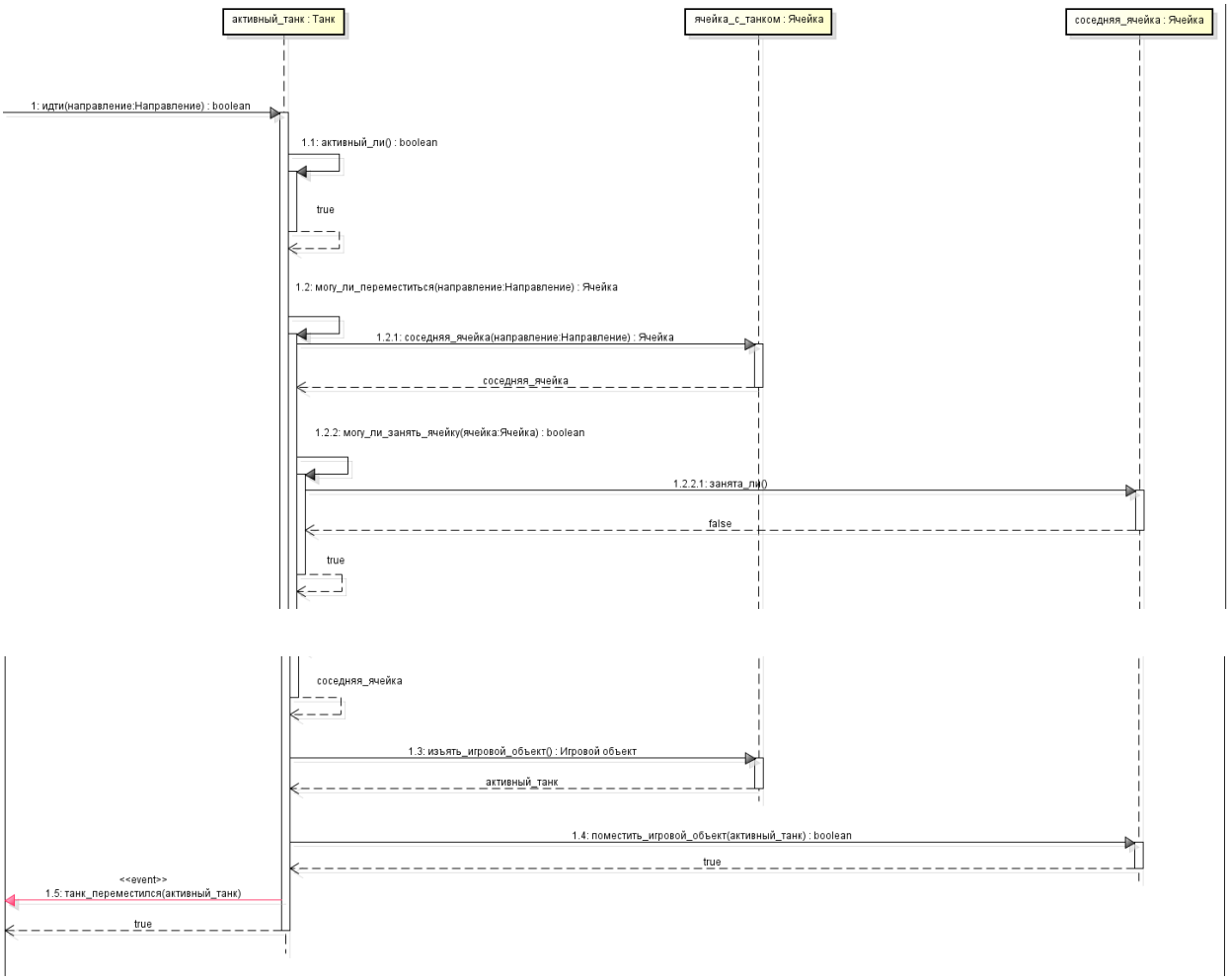

Танк_A
2: активный_танк() : Танк
3: идти(Север) : boolean
<<event>>
3.1: танк_переместился(танк_A)
3.1.1: определить_исход_игры() : Состояние_Игры
Игра_идет
3.1.2: передать_ход() : Танк
3.1.2.1: установить_активность(true) : void
3.1.2.2: установить_активность(false) : void
true
танк_B
Танк_B
4: активный_танк() : Танк



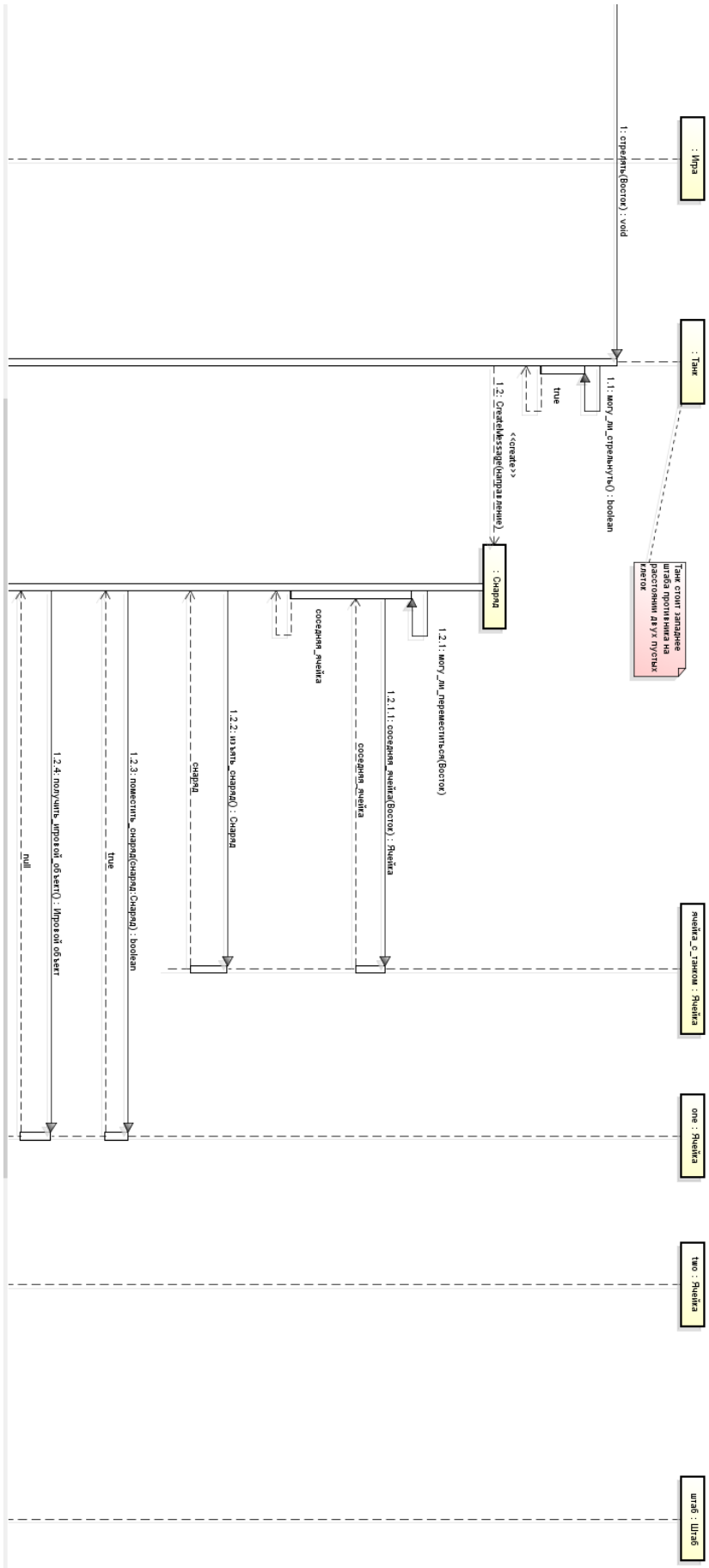
Общий игровой цикл

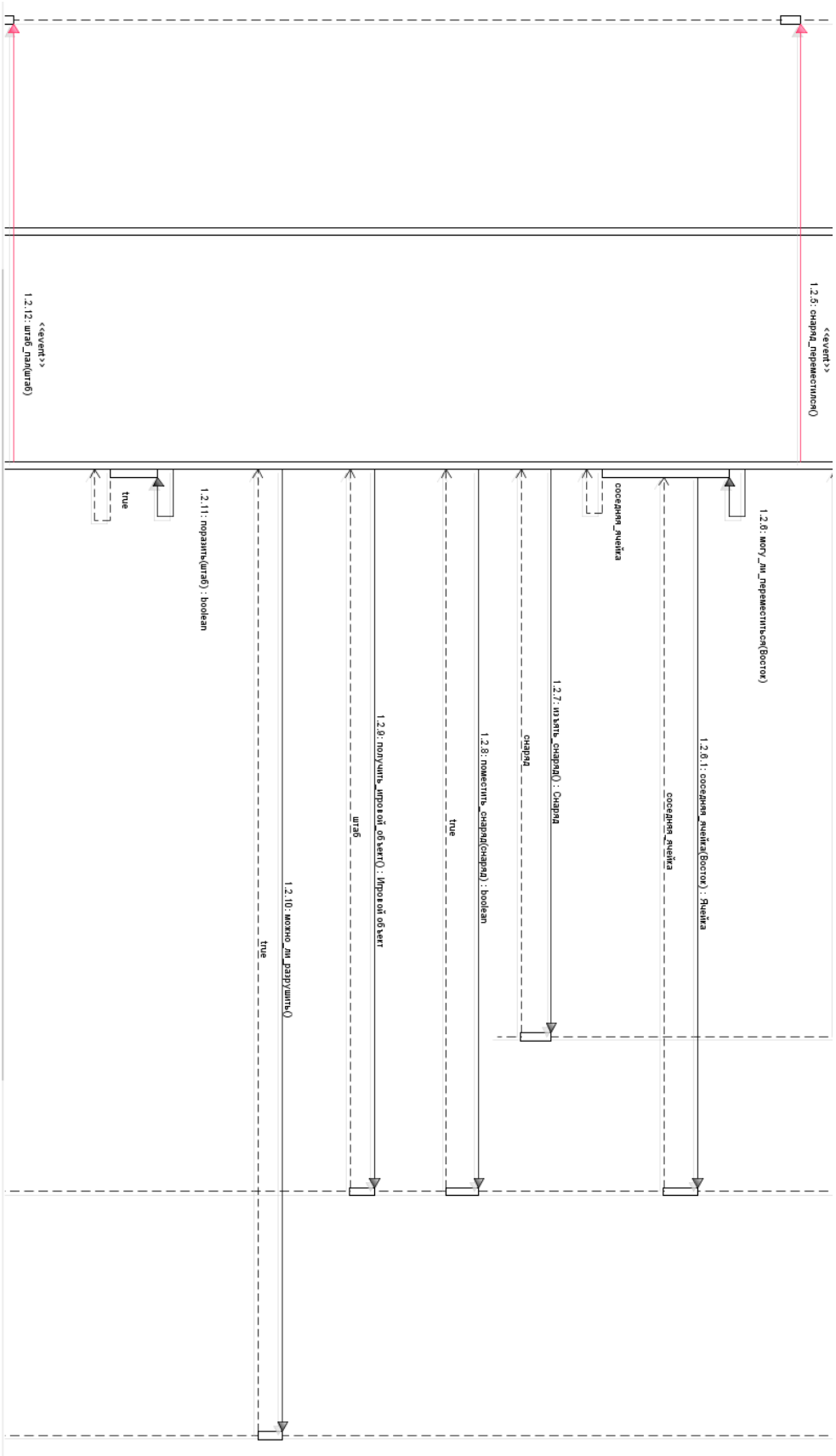


Определить исход игры

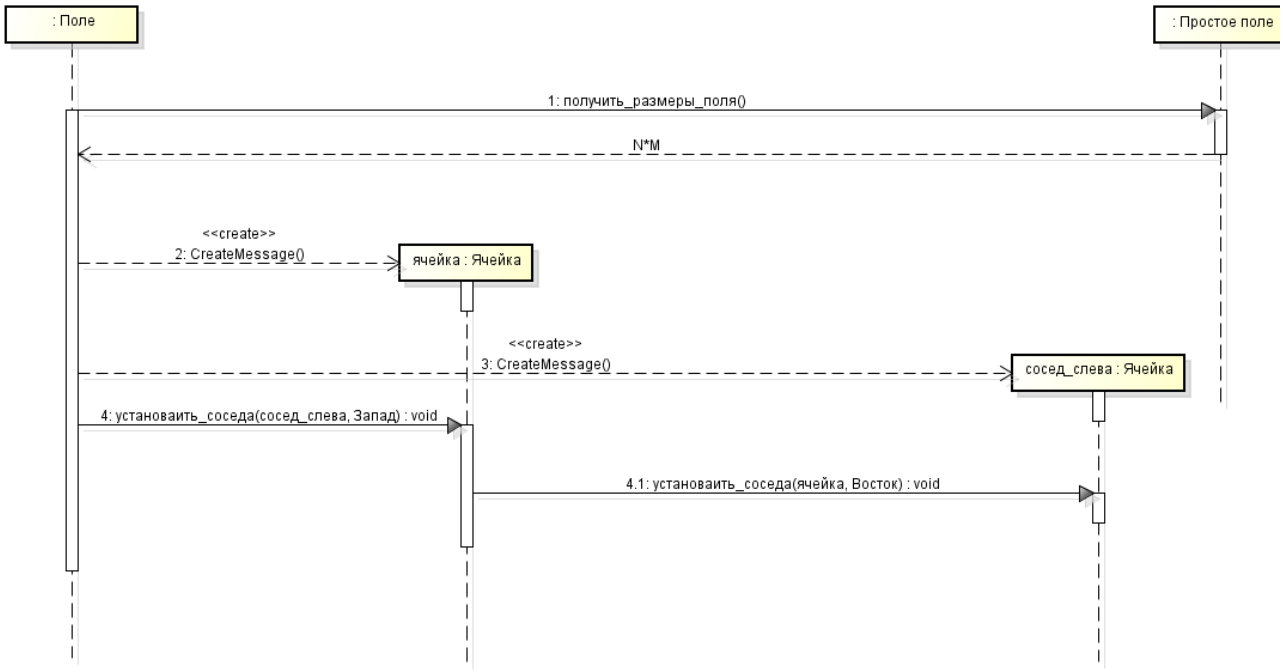


Сделать шаг

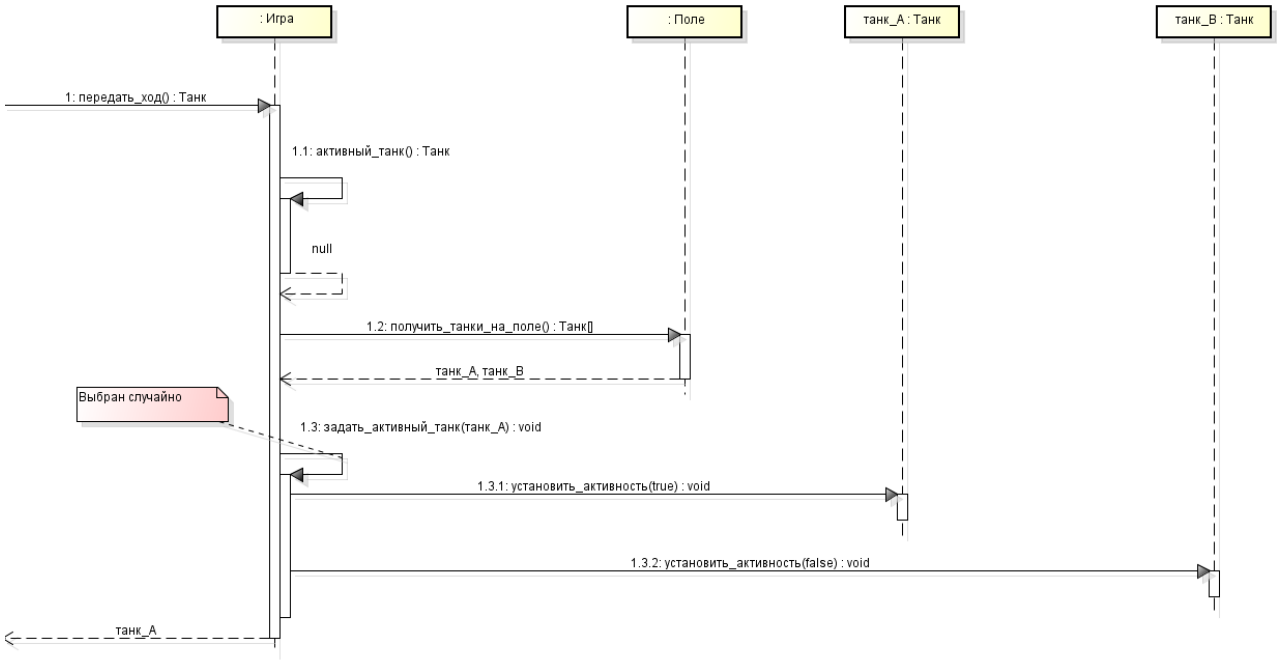

: Игра
: Танк
: Снаряд
ячейка_с_танком : Ячейка
one : Ячейка
two : Ячейка
штаб : Штаб
1: стрельнуть(Восток) : void
1.1: могу_ли_стрельнуть() : boolean
true
1.2: CreateMessage(направление)
<<create>>
Танк стоит западнее штаба противника на расстоянии двух пустых клеток
1.2.1: могу_ли_переместиться(Восток)
1.2.1.1: соседняя_ячейка(Восток) : Ячейка
соседняя_ячейка
соседняя_ячейка
1.2.2: изъять_снаряд() : Снаряд
снаряд
1.2.3: поместить_снаряд(снаряд:Снаряд) : boolean
true
1.2.4: получить_игровой_объект() : Игровой объект
null



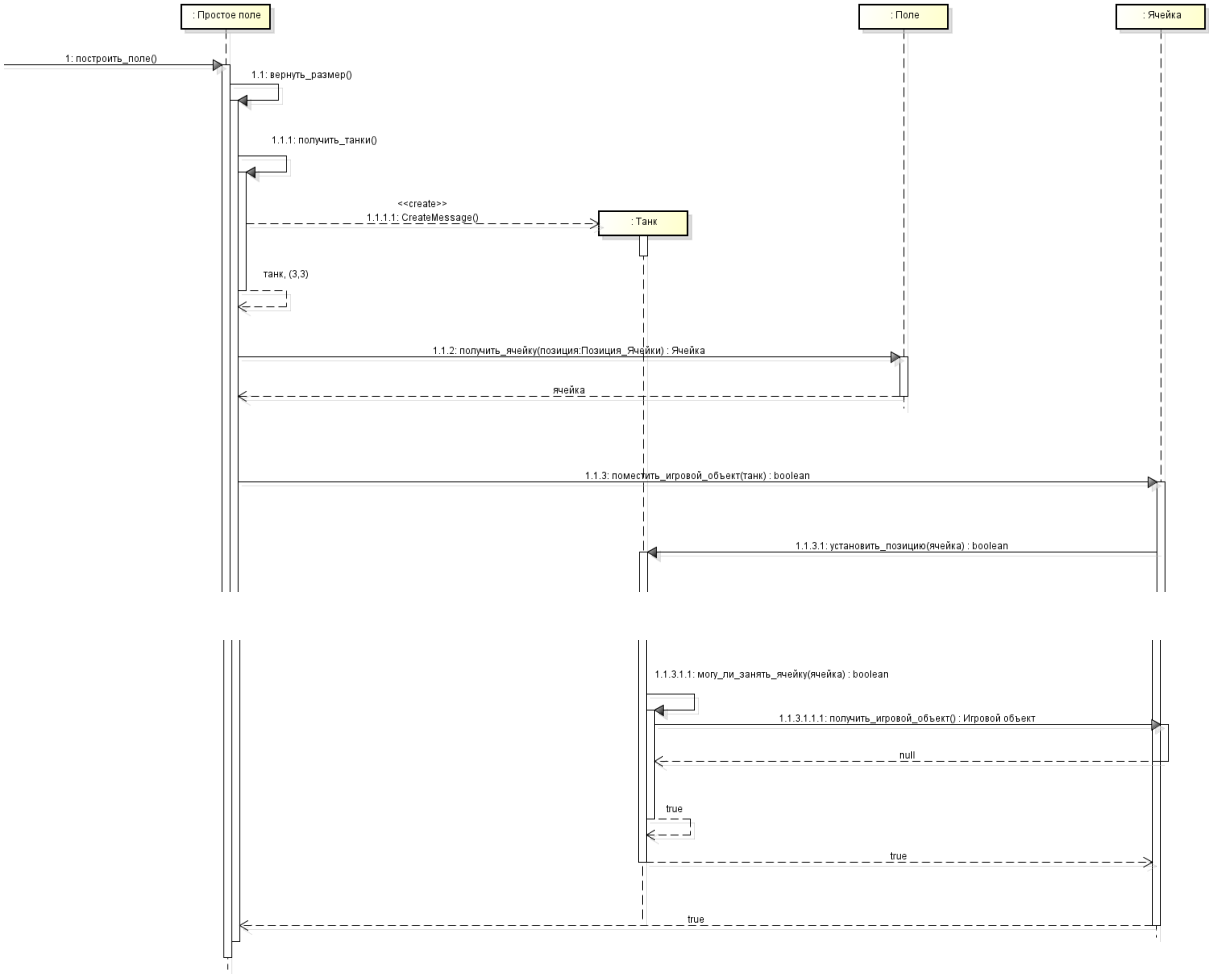
Снаряд поразил разрушаемый объект



Создание поля



Установить активным случайный танк



Установка игрового объекта на поле

3.6 Человеко-машинное взаимодействие

Общий вид главного экрана программы представлен ниже. На нём располагается игровое поле, на котором изображено два танка и два штаба (зеленый и синий), стены, вода. Каждый объект занимает одну ячейку.

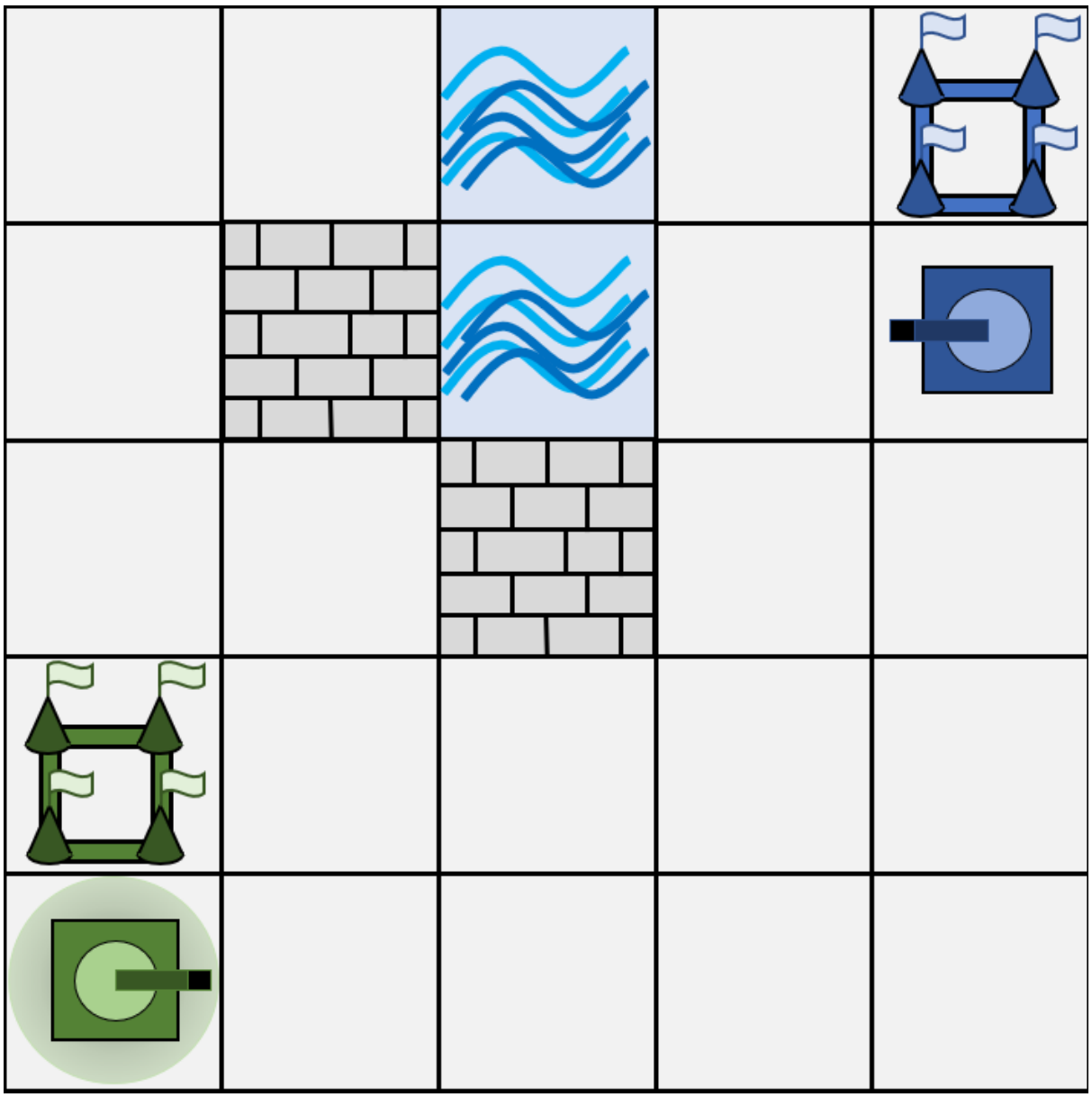


Рис. 1. Общий вид главного экрана программы

Управление активным танком пользователь осуществляет с

помощью клавиатуры.

W – направление вверх.

S – направление вниз.

A – направление влево.

D – направление вправо.

Space - идти.

F – стрелять.

Z – пропустить ход.

Изображение танка представлено на рисунке 2. Его дуло повернуто в ту сторону, которую он стреляет или ходит. Стрелять он может только один раз в 3 своих хода. Танк через все занятые клетки пройти не может.

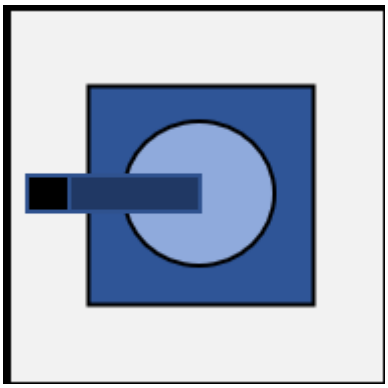


Рис.2. Танк

Активный танк подсвечивается зеленым, чтобы было понятно, чей сейчас ход. Показано на рисунке 3.

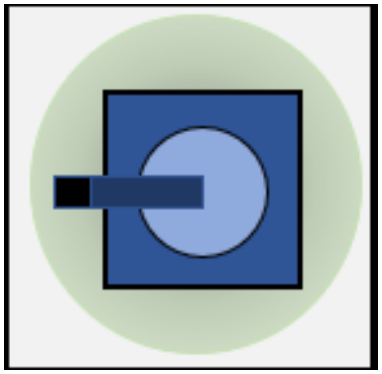


Рис.3. Активный танк

Также у танка есть штаб того же цвета, что и танк. Он представлен на рисунке 4.

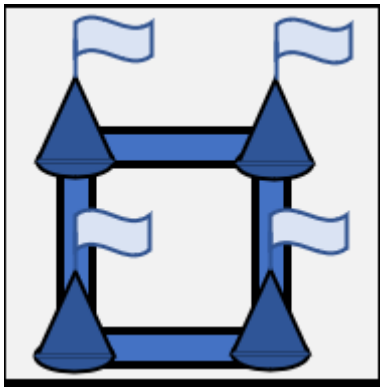


Рис.4. Штаб.

На поле есть препятствия. На рисунке 5 показана стена, которую можно разрушить одним выстрелом, а на рисунке 6 показана вода, ее разрушить нельзя.

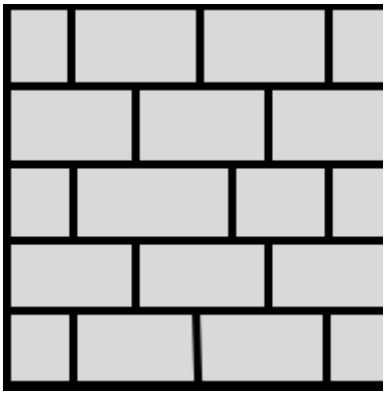


Рис.5. Стена.

Рис.6. Вода.

Танк стреляет снарядом, представленном на рисунке ниже. Он появляется в соседней ячейке, куда направлен танк.

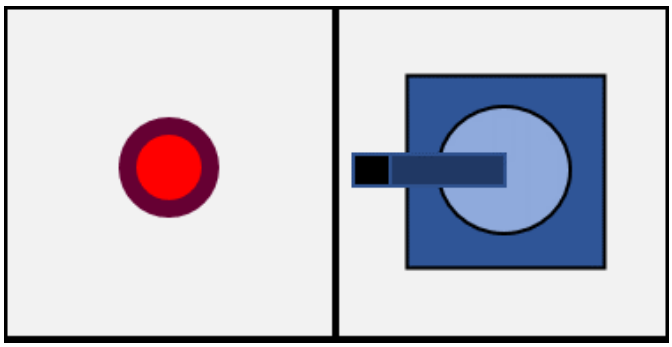
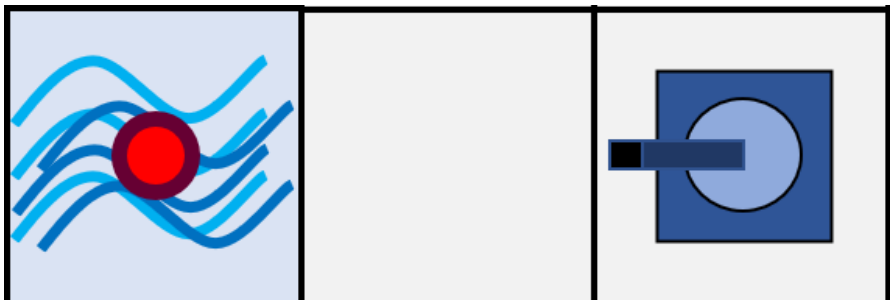
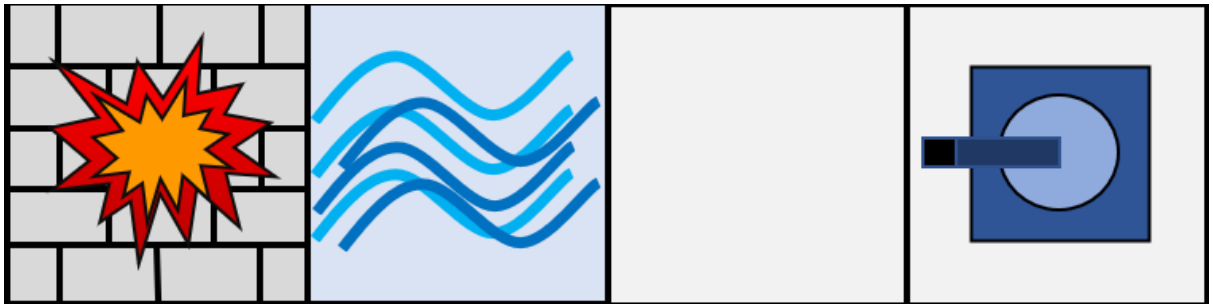


Рис.7. Танк стреляет снарядом.

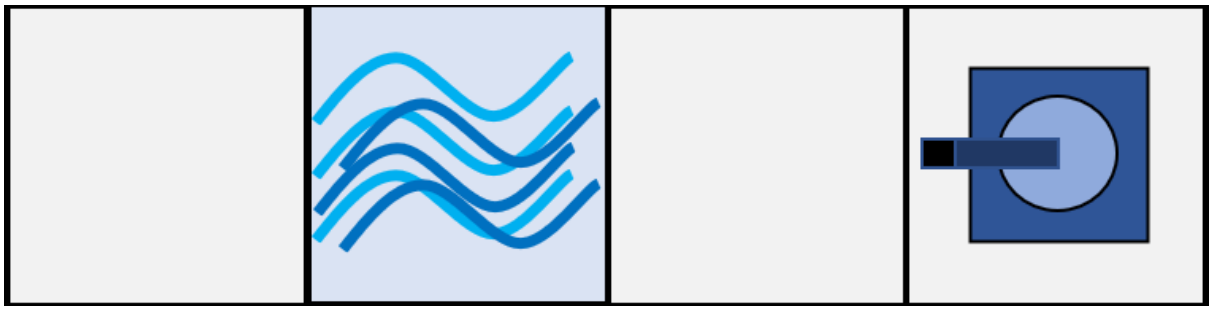
Снаряд пролетает над водой, но разрушает стену и штаб, а также наносит урон в виде отнимания одной жизни у танка. Пример полета снаряда и его взрывы ниже.



а)



б)



в)

Рис.8. Полет снаряда, его взрыв и исчезание объекта.

3.7 Реализация ключевых классов

```
// Класс Игра
public class Game {

    private GameState gameStatus;
    private Tank activeTank;
    private Tank winner;
    private Field gameField;
    private Game_generation generation;

    public Game(Game_generation generation) {
        this.generation = generation;

        initGame();
    }

    private void initGame() {
        setStatus(GameState.GAME_IS_ON);

        buildField();

        for(var i : gameField.getTanksOnField()) {
            i.addTankActionListener(new TankObserver());
        }

        passMoveNextTank();
    }
```

```
public void finish() {
    setStatus(GameState.GAME_FINISHED_AHEAD_OF_SCHEDULE);
    setActiveTank(null);
}

public GameState status() {
    return gameStatus;
}

private void setStatus(GameState status) {
    if(gameStatus != status) {
        gameStatus = status;
        fireGameStatusIsChanged(gameStatus);
    }
}

public GameState getGameStatus() {
    return gameStatus;
}

public Tank winner() {
    return winner;
}

public Tank activeTank() {
    return activeTank;
}

public Field getGameField() {
```

```
    return gameField;
}

private void passMoveNextTank() {
    if(gameStatus != GameState.GAME_IS_ON) {
        setActiveTank(null);
        return;
    }

    List<Tank> tanksOnField = gameField.getTanksOnField();

    if(tanksOnField.size() == 1 ) {
        Tank tank = tanksOnField.get(0);
        if(tank.getLive()) setActiveTank(tank);
    } else if (tanksOnField.size() == 2) {
        Tank firstTank = tanksOnField.get(0);
        Tank secondTank = tanksOnField.get(1);
        if(!firstTank.isActive() && firstTank.getLive()) {
            setActiveTank(firstTank);
        } else if (!secondTank.isActive() && secondTank.getLive()){
            setActiveTank(secondTank);
        }
    }
}


private GameState determineOutcomeGame() {
    GameState result = GameState.GAME_IS_ON;
```

```
        List<Tank> getTanksOnField = gameField.getTanksOnField();
        List<Headquarter> getHeadquartersOnField =
gameField.getHeadquartersOnField();

        if(getTanksOnField.size()==1) {
            setWinner(getTanksOnField.get(0));
            result = GameState.WINNER_FOUND;
        }

        if(getHeadquartersOnField.size()==1) {
            setWinner(getHeadquartersOnField.get(0).getMy_tank());
            result = GameState.WINNER_FOUND;
        }

        return result;
    }

    private void updateGameState() {
        GameState status = determineOutcomeGame();
        setStatus(status);
        if(status == GameState.GAME_IS_ON) {
            passMoveNextTank();
        } else {
            setActiveTank(null);
        }
    }

    private void buildField() {
        gameField = generation.buildField();
```

```
}

private void setWinner(@NotNull Tank tank) {

    winner = tank;

    setActiveTank(null);
}

private void setActiveTank(Tank tank) {

    if (activeTank != null) activeTank.setActive(false);

    activeTank = tank;

    if(tank != null ) tank.setActive(true);
}

// events
private class TankObserver implements TankActionListener {

    @Override
    public void tankIsMoved(@NotNull TankActionEvent event) {
        fireTankIsMoved(event.getTank());
        if(event.getTank().getLive()){
            updateGameState();
        }
    }
```

```
    @Override
    public void tankIsSkipStep(@NotNull TankActionEvent event) {
        fireTankIsSkipStep(event.getTank());
        updateGameState();
    }

    @Override
    public void tankFired(@NotNull TankActionEvent event) {
        fireTankFired(event.getTank());
        updateGameState();
    }

    @Override
    public void tankFiredSmart(@NotNull TankActionEvent event) {

    }

    @Override
    public void tankDestroyObject(@NotNull TankActionEvent event) {

    }

    @Override
    public void myProjectileIsMoved(@NotNull TankActionEvent event) {

    }
}
```

```
    private ArrayList<GameActionListener> gameActionListeners = new
ArrayList<>();

    public void addGameActionListener(@NotNull GameActionListener
listener) {
        gameActionListeners.add(listener);
    }

    public void removeGameActionListener(@NotNull GameActionListener
listener) {
        gameActionListeners.remove(listener);
    }

    private void fireTankIsMoved(@NotNull Tank tank) {
        for(GameActionListener listener: gameActionListeners) {
            GameActionEvent event = new GameActionEvent(listener);
            event.setTank(tank);
            listener.tankIsMoved(event);
        }
    }

    private void fireTankIsSkipStep(@NotNull Tank tank) {
        for(GameActionListener listener: gameActionListeners) {
            GameActionEvent event = new GameActionEvent(listener);
            event.setTank(tank);
            listener.tankIsSkipStep(event);
        }
    }
```

```
    private void fireTankFired(@NotNull Tank tank) {
        for(GameActionListener listener: gameActionListeners) {
            GameActionEvent event = new GameActionEvent(listener);
            event.setTank(tank);
            listener.tankFired(event);
        }
    }

    private void fireGameStatusIsChanged(@NotNull GameState status) {
        for(GameActionListener listener: gameActionListeners) {
            GameActionEvent event = new GameActionEvent(listener);
            event.setStatus(status);
            listener.gameStatusChanged(event);
        }
    }
}

// Класс Ячейка
public class Cell {

    // Игровой объект
    private GameObject game_object;

    public GameObject getGame_object() { return game_object; }

    public GameObject takeGame_object(){
        game_object.setPosition(null);
        var tmp = game_object;
        game_object = null;
```

```
        return tmp;
    }

    // Поле
    private Field field;

    public Field getField() {
        return field;
    }

    public void setField(Field field) {
        this.field = field;
    }

    public void kill() {
        if(this.game_object.getDestructibility()) {
            this.game_object = null;
        }
    }

    public void setGame_object(GameObject game_object) {
        if(getGame_object() != null) throw new IllegalArgumentException("In cell
already set game_object");

        game_object.setPosition(this);
        this.game_object = game_object;
    }

    // Снаряд
```

```
    private Projectile projectile;

    public Projectile getProjectile() {
        return projectile;
    }

    public Projectile takeProjectile() {
        projectile.setPosition(null);
        var tmp = projectile;
        projectile = null;
        return tmp;
    }

    public void setProjectile(Projectile projectile) {
        if(getProjectile() != null) throw new IllegalArgumentException("In cell
already set projectile");

        projectile.setPosition(this);
        this.projectile = projectile;
    }

    // neighbors cell
    private Map<Direction, Cell> neighborCells = new
EnumMap<>(Direction.class);

    public Cell neighborCell(@NotNull Direction direction) {
        return neighborCells.get(direction);
    }
```

```
    public void setNeighbor(@NotNull Cell cell, @NotNull Direction direction)
{
        if (neighborCells.containsKey(direction) &&
neighborCells.containsValue(cell)) return;
        if (neighborCells.containsKey(direction)) throw new
IllegalArgumentException();
        neighborCells.put(direction, cell);
        if (cell.neighborCell(direction.getOppositeDirection()) == null) {
            cell.setNeighbor(this, direction.getOppositeDirection());
        }
    }

    public Direction isNeighbor(Cell cell) {
        if(cell != null) {
            for (var i : neighborCells.entrySet()) {
                if (i.getValue().equals(cell)) return i.getKey();
            }
        }
        return null;
    }

    public ArrayList<Direction> neighbors(){

        ArrayList<Direction> array = new ArrayList<>();
        Direction d1 = isNeighbor(neighborCell(Direction.NORTH));
        if(d1 != null){
            array.add(d1);
        }
```

```
        Direction d2 = isNeighbor(neighborCell(Direction.SOUTH));
        if(d2 != null){
            array.add(d2);
        }
        Direction d3 = isNeighbor(neighborCell(Direction.WEST));
        if(d3 != null){
            array.add(d3);
        }
        Direction d4 = isNeighbor(neighborCell(Direction.EAST));
        if(d4 != null){
            array.add(d4);
        }

        return array;
    }
}

// Класс Поле
public class Field{

    private Map<Point, Cell> cells = new HashMap<>();

    public ArrayList<Cell> cells_ = new ArrayList<>();

    private int width;
    private int height;

    public int getWidth() {
        return width;
```

```
    }

    public int getHeight() {
        return height;
    }

    public Field(int width, int height) {
        if(width <= 0) throw new IllegalArgumentException("Field width must be more than 0");
        if(height <= 0) throw new IllegalArgumentException("Field height must be more than 0");

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

        setupField();
    }

    private void setupField() {
        for(int y = 0; y < height; ++y) {
            for(int x = 0; x < width; ++x) {
                Point p = new Point(x, y);
                Cell cell = new Cell();
                cell.setField(this);
                if(x > 0) getCell(new Point((p.getX() - 1), p.getY())).setNeighbor(cell, Direction.EAST);
                if(y > 0) getCell(new Point( p.getX(), (p.getY() - 1))).setNeighbor(cell, Direction.SOUTH);
                cells.put(p, cell);
```

```
            cells_.add(cell);
        }
    }
}

public Cell getCell(@NotNull Point point) {
    return cells.get(point);
}

public List<Tank> getTanksOnField() {
    List<Tank> tanks = new ArrayList<>();
    for(var i : cells.entrySet()) {
        GameObject tank = i.getValue().getGame_object();
        if(tank instanceof Tank tank_) {
            tanks.add(tank_);
        }
    }
    return tanks;
}

public List<Headquarter> getHeadquartersOnField() {
    List<Headquarter> headquarters = new ArrayList<>();
    for(var i : cells.entrySet()) {
        GameObject headquarter = i.getValue().getGame_object();
        if(headquarter instanceof Headquarter headquarter_) {
            headquarters.add(headquarter_);
        }
    }
    return headquarters;
```

```
}

public void removeProjectile(){
    for(var i : cells.entrySet()) {
        Projectile projectile = i.getValue().getProjectile();
        if(projectile != null) {
            projectile.getPosition().takeProjectile();
        }
    }
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Field field = (Field) o;
    return width == field.width &&
            height == field.height &&
            Objects.equals(cells, field.cells);
}

@Override
public int hashCode() {
    return Objects.hash(cells, width, height);
}

@Override
public String toString() {
    return "Field{" +
```

```
            "cells=" + cells +
            ", width=" + width +
            ", height=" + height +
            '}';
    }

    //events

    private ArrayList<FieldActionListener> fieldListListener = new
ArrayList<>();

    public void addFieldActionListener(FieldActionListener listener) {
        fieldListListener.add(listener);
    }

    public void removeFieldCellActionListener(FieldActionListener listener) {
        fieldListListener.remove(listener);
    }

    private void fireTankDestroyObject(GameObject game_object) {
        for(FieldActionListener listener: fieldListListener) {
            FieldActionEvent event = new FieldActionEvent(listener);
            event.setGameObject(game_object);
            listener.tankFired(event);
        }
    }
}

// Класс Танк
```

```
public class Tank extends GameObject {

    // Жизни танка
    private int lives = 3;

    public boolean getLive() {
        return lives > 0;
    }

    // Ячейка для выстрела
    Cell projectileCell;

    public void setProjectileCell(Cell projectileCell) {
        this.projectileCell = projectileCell;
    }

    public Cell getProjectileCell() {
        return projectileCell;
    }

    // Порядковый номер заряженного снаряда
    private int projectile = 0;

    public void setProjectile() {
        this.projectile++;
        if(this.projectile>1)
        {
            this.projectile=0;
        }
```

```
}

public int getProjectile() {
    return projectile;
}

// Возможность стрельнуть
private int numberShoot = 3;

// Цвет команды
private Color color;

public void setColor(Color color) {
    this.color = color;
}

public Color getColor() {
    return color;
}

// Направление дула
private Direction direction = Direction.NORTH;

public void setDirection(Direction direction) {
    this.direction = direction;
}

public Direction getDirection() {
    return direction;
```

```
}

// Разрушимость
@Override
public void setDestructibility(boolean destructibility) {
    super.setDestructibility(destructibility);
}

@Override
public boolean getDestructibility() {
    return super.getDestructibility();
}

// Позиция
@Override
public Cell getPosition() {
    return super.getPosition();
}

@Override
public void setPosition(Cell position) {
    super.setPosition(position);
}

// Мой штаб
private Headquarter myHeadquarter;

public Headquarter getMyHeadquarter() {
    return myHeadquarter;
```

```
    }

    public void setMyHeadquarter(Headquarter myHeadquarter) {
        if(this.myHeadquarter ==null) {
            this.myHeadquarter = myHeadquarter;
            myHeadquarter.setColor(this.color);
            myHeadquarter.setMy_tank(this);
        }
    }

    // Активность танка
    private boolean isActive;

    public void setActive(boolean value) {
        isActive = value;
    }

    public boolean isActive() {
        return isActive;
    }

    // Перемещение на одну клетку
    public void move(@NotNull Direction direction) throws
InterruptedException {
        if(isActive) {
            Cell oldPosition = this.getPosition();
            Cell newPosition = canMove(direction);
            if (newPosition != null) {
                getPosition().takeGame_object();
```

```
            newPosition.setGame_object(this);
            fireTankIsMoved(oldPosition, newPosition);
            numberShoot++;
        }
    }
}

// Стрелять по направлению
public void shootDir() {
    if(isActive && numberShoot >=3) {
        SimpleProjectile projectile = new SimpleProjectile(this.getPosition(),
100);
        //Smart_projectile projectile = new Smart_projectile(this.getPosition());
        projectile.addProjectileActionListener(new ProjectileObserver());
        getPosition().setProjectile(projectile);
        //projectile.fly(getPosition().getField().getCell(new Point(2,2)));
        projectile.fly(this.direction);
    }
}

// Стрелять в ячейку
public void shootCell() throws InterruptedException {
    if(isActive && numberShoot >=3 && projectileCell!=null) {
        //Simple_projectile projectile = new
Simple_projectile(this.getPosition());
        SmartProjectile projectile = new SmartProjectile(this.getPosition(), 10);
        projectile.addProjectileActionListener(new ProjectileObserver());
        getPosition().setProjectile(projectile);
        //projectile.fly(getPosition().getField().getCell(new Point(2,2)));
```

```
            projectile.fly(this.projectileCell);
            //projectile.fly(direction);
            getPosition().getField().removeProjectile();
            fireTankFiredSmart();
            fireTankFired();
            numberShoot = 0;
        }
    }

    // Пропустить ход
    public void skipStep() {
        if(isActive) {
            fireTankIsSkipStep();
        }
    }

    // Могу ли переместиться в ячейку по направлению
    private Cell canMove(@NotNull Direction direction) {
        Cell result = null;

        Cell neighborCell = getPosition().neighborCell(direction);
        if(neighborCell != null && canStayAtPosition(neighborCell)) {
            result = neighborCell;
        }

        return result;
    }

    // Есть ли в ячейке объект
```

```
    public static boolean canStayAtPosition(@NotNull Cell position) {
        return position.getGame_object() == null;
    }

    // events
    private class ProjectileObserver implements ProjectileActionListener{

        @Override
        public void projectileDestroyObject(@NotNull ProjectileActionEvent
event) {
            GameObject game_object = event.getGame_object();
            fireTankDestroy(game_object);
        }

        @Override
        public void projectileIsMoved(@NotNull ProjectileActionEvent event)
throws InterruptedException {
            fireMyProjectileIsMoved(event.getFromCell(), event.getToCell(),
event.getProjectile());
        }

        @Override
        public void projectile(@NotNull ProjectileActionEvent event) {
            getPosition().getField().removeProjectile();
            fireTankFired();
            numberShoot = 0;
        }
    }
```

```
    private ArrayList<TankActionListener> tankListListener = new
ArrayList<>();

    public void addTankActionListener(TankActionListener listener) {
        tankListListener.add(listener);
    }

    public void removeTankActionListener(TankActionListener listener) {
        tankListListener.remove(listener);
    }

    private void fireTankIsMoved(@NotNull Cell oldPosition, @NotNull Cell
newPosition) {
        for(TankActionListener listener: tankListListener) {
            TankActionEvent event = new TankActionEvent(listener);
            event.setTank(this);
            event.setFromCell(oldPosition);
            event.setToCell(newPosition);
            listener.tankIsMoved(event);
        }
    }

    private void fireTankIsSkipStep() {
        for(TankActionListener listener: tankListListener) {
            TankActionEvent event = new TankActionEvent(listener);
            event.setTank(this);
            listener.tankIsSkipStep(event);
        }
    }
```

```
    private void fireTankFired(){
        for(TankActionListener listener: tankListListener) {
            TankActionEvent event = new TankActionEvent(listener);
            event.setTank(this);
            listener.tankFired(event);
        }
    }

    private void fireTankFiredSmart(){
        for(TankActionListener listener: tankListListener) {
            TankActionEvent event = new TankActionEvent(listener);
            event.setTank(this);
            listener.tankFiredSmart(event);
        }
    }

    private void fireTankDestroy(GameObject game_object){
        for(TankActionListener listener: tankListListener) {
            TankActionEvent event = new TankActionEvent(listener);
            event.setTank(this);
            event.setGame_object(game_object);
            listener.tankDestroyObject(event);
        }
    }

    private void fireMyProjectileIsMoved(Cell oldPosition, Cell newPosition,
Projectile projectile) throws InterruptedException {
        for(TankActionListener listener: tankListListener) {
```

```
            TankActionEvent event = new TankActionEvent(listener);
            event.setMyProjectile(projectile);
            event.setFromCell(oldPosition);
            event.setToCell(newPosition);
            listener.myProjectileIsMoved(event);
        }
    }
}

// Класс Штаб
public class Headquarter extends GameObject {

    private static int LIVE = 1;

    private Color color;

    public Color getColor() {
        return color;
    }

    public void setColor(Color color) {
        this.color = color;
    }

    @Override
    public void setDestructibility(boolean destructibility) {
        super.setDestructibility(destructibility);
    }
```

```
    @Override
    public boolean getDestructibility() {
        return super.getDestructibility();
    }

    @Override
    public Cell getPosition() {
        return super.getPosition();
    }

    @Override
    public void setPosition(Cell position) {
        super.setPosition(position);
    }

    private Tank my_tank;

    public Tank getMy_tank() {
        return my_tank;
    }

    public void setMy_tank(Tank my_tank) {
        if(this.my_tank==null) {
            this.my_tank = my_tank;
            my_tank.setMyHeadquarter(this);
        }
    }
}
// Простой снаряд
```

```
public class SimpleProjectile extends Projectile{

    private Cell position;

    public Cell getPosition() {
        return position;
    }

    public void setPosition(Cell position) {
        this.position = position;
    }

    public SimpleProjectile(Cell position, int range_of_flight) {
        super(position, range_of_flight);
        setPosition(position);
        setRange_of_flight(range_of_flight);
    }

    private int range_of_flight;

    public int getRange_of_flight() {
        return range_of_flight;
    }

    public void setRange_of_flight(int range_of_flight) {
        this.range_of_flight = range_of_flight;
    }

    Timer t;
```

```
    public void fly(Direction direction) {

        final Cell[] newPosition = {canMove(direction)};
        final GameObject[] game_object = {null};

        ActionListener performer = new ActionListener() {
            final int a = 0;
            @Override
            public void actionPerformed(ActionEvent e) {

                if (newPosition[0] != null && t.isRunning()) {
                    try {
                        move(getPosition(), newPosition[0]);
                    } catch (InterruptedException ex) {
                        throw new RuntimeException(ex);
                    }
                    game_object[0] = newPosition[0].getGame_object();
                    if (game_object[0] != null && game_object[0].getDestructibility())
{
                        newPosition[0].kill();
                        fireProjectileDestroyObject(game_object[0]);
                        t.stop();
                        try {
                            fireProjectile();
                        } catch (InterruptedException ex) {
                            throw new RuntimeException(ex);
                        }
                    }
```

```
                newPosition[0] = canMove(direction);
                System.out.println("a");
            }
            else {
                t.stop();
                try {
                    fireProjectile();
                } catch (InterruptedException ex) {
                    throw new RuntimeException(ex);
                }
            }
        }
    };

    t = new Timer(2000, performer);
    t.addActionListener(performer);
    t.start();
  }

  public void move(Cell oldPosition, Cell newPosition) throws
InterruptedException {

    newPosition.setProjectile(this);
    oldPosition.takeProjectile();
    this.setPosition(newPosition);
    fireProjectileIsMoved(oldPosition,newPosition);
  }

  private Cell canMove(@NotNull Direction direction) {
```

```
        Cell result = null;
        if(getPosition()!=null){
            Cell neighborCell = getPosition().neighborCell(direction);

            if(neighborCell != null) {
                result = neighborCell;
            }
        }

        return result;
    }

    // events
    private ArrayList<ProjectileActionListener> projectileListListener = new
ArrayList<>();

    public void addProjectileActionListener(ProjectileActionListener listener) {
        projectileListListener.add(listener);
    }

    public void removeProjectileActionListener(ProjectileActionListener
listener) {
        projectileListListener.remove(listener);
    }

    private void fireProjectileDestroyObject(GameObject game_object) {
        for(ProjectileActionListener listener: projectileListListener) {
            ProjectileActionEvent event = new ProjectileActionEvent(listener);
            event.setProjectile(this);
```

```
            event.setGame_object(game_object);
            listener.projectileDestroyObject(event);
        }
    }

    private void fireProjectileIsMoved(Cell oldPosition, Cell newPosition)
throws InterruptedException {
        for(ProjectileActionListener listener: projectileListListener) {
            ProjectileActionEvent event = new ProjectileActionEvent(listener);
            event.setProjectile(this);
            event.setFromCell(oldPosition);
            event.setToCell(newPosition);
            listener.projectileIsMoved(event);
        }
    }

    private void fireProjectile() throws InterruptedException {
        for(ProjectileActionListener listener: projectileListListener) {
            ProjectileActionEvent event = new ProjectileActionEvent(listener);
            event.setProjectile(this);
            listener.projectile(event);
        }
    }
}

// Фабрика виджетов
public class WidgetFactory {

    private final Map<Cell, CellWidget> cells = new HashMap<>();
```

```
    private final Map<Tank, TankWidget> tanks = new HashMap<>();
    private final Map<Headquarter, HeadquarterWidget> headquarters = new
HashMap<>();
    private final Map<Water, WaterWidget> waters = new HashMap<>();
    private final Map<Wall, WallWidget> walls = new HashMap<>();
    private final Map<Projectile, ProjectileWidget> projectiles = new
HashMap<>();
    private final List<Color> usedColors = new ArrayList<>();

    public CellWidget create(@NotNull Cell cell) {
        if(cells.containsKey(cell)) return cells.get(cell);

        CellWidget item = new CellWidget();

        GameObject game_object = cell.getGame_object();

        Projectile projectile = cell.getProjectile();

        if(game_object instanceof Tank _tank) {
            TankWidget tankWidget = create(_tank);
            item.addItem(tankWidget);
        }

        if(game_object instanceof Headquarter _headquarter) {
            HeadquarterWidget headquarterWidget = create(_headquarter);
            item.addItem(headquarterWidget);
        }

        if(game_object instanceof Water _water) {
```

```
        WaterWidget waterWidget = create(_water);
        item.addItem(waterWidget);
    }

    if(game_object instanceof Wall _wall) {
        WallWidget wallWidget = create(_wall);
        item.addItem(wallWidget);
    }

    cells.put(cell, item);
    return item;
}

public CellWidget getWidget(@NotNull Cell cell) {
    return cells.get(cell);
}

public TankWidget create(@NotNull Tank tank) {
    if(tanks.containsKey(tank)) return tanks.get(tank);

    Color color = tank.getColor();
    usedColors.add(color);

    TankWidget item = new TankWidget(tank, color);
    tanks.put(tank, item);
    return item;
}

public TankWidget getWidget(@NotNull Tank tank) {
```

```
        return tanks.get(tank);
    }

    public HeadquarterWidget create(@NotNull Headquarter headquarter) {
        if(headquarters.containsKey(headquarter)) return
headquarters.get(headquarter);

        Color color = headquarter.getColor();
        usedColors.add(color);

        HeadquarterWidget item = new HeadquarterWidget(headquarter, color);
        headquarters.put(headquarter, item);
        return item;
    }

    public HeadquarterWidget getWidget(@NotNull Headquarter headquarter) {
        return headquarters.get(headquarter);
    }

    public WallWidget create(@NotNull Wall wall) {
        if(walls.containsKey(wall)) return walls.get(wall);

        WallWidget item = new WallWidget(wall);
        walls.put(wall, item);
        return item;
    }

    public WallWidget getWidget(@NotNull Wall wall) {
        return walls.get(wall);
```

```
    }

    public WaterWidget create(@NotNull Water water) {
        if(waters.containsKey(water)) return waters.get(water);

        WaterWidget item = new WaterWidget(water);
        waters.put(water, item);
        return item;
    }

    public WaterWidget getWidget(@NotNull Water water) {
        return waters.get(water);
    }

    public ProjectileWidget create(@NotNull Projectile projectile) {
        if(projectiles.containsKey(projectile)) return projectiles.get(projectile);

        if(projectile instanceof SmartProjectile smartProjectile) {
            SmartProjectileWidget item = new
SmartProjectileWidget(smartProjectile);
            projectiles.put(projectile, item);
            return item;
        }

        if(projectile instanceof SimpleProjectile simpleProjectile) {
            SimpleProjectileWidget item = new
SimpleProjectileWidget(simpleProjectile);
            projectiles.put(projectile, item);
            return item;
```

```
        }

        return null;
    }

    public ProjectileWidget getWidget(@NotNull Projectile projectile) {
        return projectiles.get(projectile);
    }
}

// Виджет Поля
public class FieldWidget extends JPanel {

    private final Field field;
    private final WidgetFactory widgetFactory;

    public FieldWidget(@NotNull Field field, @NotNull  WidgetFactory
widgetFactory) {
        this.field = field;
        this.widgetFactory = widgetFactory;
        setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
        fillField();
        subscribeOnTanks();
    }

    private JTextField textFieldX = new
JTextField("",SwingConstants.CENTER);
```

```
    private JTextField textFieldY = new JTextField("",
SwingConstants.CENTER);

    private JButton button = new JButton("Сохранить");

    public void labelEnabled(){
        button.setEnabled(true);
        textFieldX.setEnabled(true);
        textFieldY.setEnabled(true);
    }

    private void fillField() {

        JLabel jlabel = new JLabel("Приятной игры!");
        jlabel.setFont(new Font("Verdana",1,30));
        add(jlabel);
        setBorder(new LineBorder(Color.BLACK));
        setBackground(Color.LIGHT_GRAY);

        for (int i = 0; i < field.getHeight(); ++i) {
            JPanel row = createRow(i);
            add(row);
        }

        JLabel label_about = new JLabel("Введите координаты ячейки:",
SwingConstants.CENTER);
        label_about.setFont(new Font("Verdana", 1, 20));
        setBorder(new LineBorder(Color.BLACK));
        add(label_about);
```

```
textFieldX.setCaretColor(Color.GREEN);
textFieldX.setBackground(Color.LIGHT_GRAY);
textFieldX.setFont(new Font("Verdana", 1, 20));
add(textFieldX);

textFieldY.setCaretColor(Color.GREEN);
textFieldY.setBackground(Color.LIGHT_GRAY);
textFieldY.setFont(new Font("Verdana", 1, 20));
add(textFieldY);

button.setBackground(Color.PINK);
button.setFont(new Font("Verdana", 1, 20));
add(button);

JLabel label = new JLabel("", SwingConstants.CENTER);
label.setFont(new Font("Verdana", 1, 20));
setBorder(new LineBorder(Color.BLACK));
add(label);

button.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent e) {
      try {
         int x = Integer.parseInt(textFieldX.getText());
         int y = Integer.parseInt(textFieldY.getText());
         Cell cell = field.getCell(new Point(x,y));
         field.getTanksOnField().get(0).setProjectileCell(cell);
         field.getTanksOnField().get(1).setProjectileCell(cell);
```

```
                if(x>=field.getWidth() || y>=field.getHeight() || x<-1 || y<-1){
                    label.setText("Введите первое число до " + field.getWidth() +
", а второе до " + field.getHeight());
                    repaint();
                }
                else {
                    label.setText("Вы стреляете в ячейку: " + textFieldX.getText()
+ ", " + textFieldY.getText());
                    button.setEnabled(false);
                    textFieldX.setEnabled(false);
                    textFieldY.setEnabled(false);
                    repaint();
                }
            } catch (NumberFormatException a) {
                label.setText("Введите первое число до " + field.getWidth() + ", а
второе до " + field.getHeight());
                repaint();
            }
        }
    });
  }

  private JPanel createRow(int rowIndex) {
    JPanel row = new JPanel();
    row.setLayout(new BoxLayout(row, BoxLayout.X_AXIS));

    for(int i = 0; i < field.getWidth(); ++i) {
      Point point = new Point(i, rowIndex);
      Cell cell = field.getCell(point);
```

```
            CellWidget cellWidget = widgetFactory.create(cell);
            cellWidget.setBorder(new LineBorder(Color.black));

            row.add(cellWidget);
        }
        return row;
    }

    private void subscribeOnTanks() {
        List<Tank> tanks = field.getTanksOnField();
        for(Tank tank : tanks) {
            tank.addTankActionListener(new TankController());
        }
    }

    private class TankController implements TankActionListener {

        @Override
        public void tankIsMoved(@NotNull TankActionEvent event) {
            TankWidget tankWidget = widgetFactory.getWidget(event.getTank());
            CellWidget from = widgetFactory.getWidget(event.getFromCell());
            CellWidget to = widgetFactory.getWidget(event.getToCell());
            from.removeItem(tankWidget);
            to.addItem(tankWidget);
            repaint();
        }

        @Override
        public void tankIsSkipStep(@NotNull TankActionEvent event) {
```

```
            Tank tank = event.getTank();
            TankWidget tankWidget = widgetFactory.getWidget(tank);
            CellWidget from =
widgetFactory.getWidget(event.getTank().getPosition());
            from.removeItem(tankWidget);
            from.addItem(tankWidget);
            repaint();
        }

        @Override
        public void tankFired(@NotNull TankActionEvent event) {
            Tank tank = event.getTank();
            TankWidget tankWidget = widgetFactory.getWidget(tank);
            CellWidget from =
widgetFactory.getWidget(event.getTank().getPosition());
            from.removeItem(tankWidget);
            from.addItem(tankWidget);

            repaint();
        }

        @Override
        public void tankFiredSmart(@NotNull TankActionEvent event) {
            labelEnabled();
        }

        @Override
        public void tankDestroyObject(@NotNull TankActionEvent event) {
            GameObject game_object = event.getGame_object();
```

```
        if(game_object instanceof Tank tank){
            TankWidget tankWidget = widgetFactory.getWidget(tank);
            CellWidget cell =
widgetFactory.getWidget(game_object.getPosition());
            cell.removeItem(tankWidget);
            tankWidget.repaint();
        }

        if(game_object instanceof Headquarter headquarter){
            HeadquarterWidget headquarterWidget =
widgetFactory.getWidget(headquarter);
            CellWidget cell =
widgetFactory.getWidget(game_object.getPosition());
            cell.removeItem(headquarterWidget);
            headquarterWidget.repaint();
        }

        if(game_object instanceof Wall wall){
            WallWidget wallWidget = widgetFactory.getWidget(wall);
            CellWidget cell =
widgetFactory.getWidget(game_object.getPosition());
            cell.removeItem(wallWidget);
            wallWidget.repaint();
        }

        if(game_object instanceof Water water){
            WaterWidget waterWidget = widgetFactory.getWidget(water);
```

```
            CellWidget cell =
widgetFactory.getWidget(game_object.getPosition());
            cell.removeItem(waterWidget);
            waterWidget.repaint();
        }
    }

    @Override
    public void myProjectileIsMoved(@NotNull TankActionEvent event)
throws InterruptedException {
        ProjectileWidget projectileWidget =
widgetFactory.create(event.getMyProjectile());
        if (event.getFromCell() != null) {
            CellWidget from = widgetFactory.getWidget(event.getFromCell());
            from.removeItem(projectileWidget);
            repaint();

        }
        if (event.getToCell() != null){
            CellWidget to = widgetFactory.getWidget(event.getToCell());
            to.addItem(projectileWidget);
            repaint();
        }
    }
  }
}

// Виджет Ячейки
public class CellWidget extends JPanel {
```

```
public enum Layer {
   TOP,
   BOTTOM
}

private Map<Layer, CellItemWidget> items = new HashMap();

private static final int CELL_SIZE = 120;

public CellWidget() {
   setPreferredSize(new Dimension(CELL_SIZE, CELL_SIZE));
   setBackground(ImageUtils.BACKGROUND_COLOR);
   setSize(new Dimension(120,120));
}

public void addItem(CellItemWidget item) {
   if(items.size() > 2) throw new IllegalArgumentException();
   int index = -1;

   if (items.containsKey(Layer.TOP)) {
      index = 0;
   }

   items.put(item.getLayer(), item);
   add(item, index);
}

public void removeItem(CellItemWidget item) {
```

```
        if (items.containsValue(item)) {
            int index = 0;

            if(item.getLayer() == Layer.TOP) {
                if(items.containsKey(Layer.BOTTOM)) {
                    index = 1;
                }
            }

            remove(index);
            items.remove(item.getLayer());
            repaint();
        }
    }
}

// Виджет Танка
public class TankWidget extends CellItemWidget {

    private final Tank tank;
    private final Color color;

    public TankWidget(Tank tank, Color color) {
        super();
        this.tank = tank;
        this.color = color;
        setFocusable(true);
        setPreferredSize(new Dimension(120,120));
        addKeyListener(new KeyController());
```

```
}

@Override
protected BufferedImage getImage() {
    BufferedImage image = null;
    try {
        image = ImageIO.read(getTankFileByColor(color, tank.isActive()));
        image = switch (tank.getDirection()) {
            case SOUTH -> rotate(image, 1.5708);
            case WEST -> rotate(image, 3.14159);
            case NORTH -> rotate(image, 4.71239);
            default -> image;
        };
        image = ImageUtils.resizeImage(image, 120, 120);

    } catch (IOException e) {
        e.printStackTrace();
    }
    return image;
}

@Override
public CellWidget.Layer getLayer() {
    return CellWidget.Layer.BOTTOM;
}

@Override
protected Dimension getDimension() {
    return new Dimension(120, 120);
```

```
}

public Color getColor() {
   return color;
}

private static File getTankFileByColor(Color tankColor, boolean active) {
   File file = null;
   if (tankColor == Color.BLUE)  {
      if(active) {
         file = new File("tank_blue_a.png");
      }
      else {
         file = new File("tank_blue.png");
      }
   }
   if (tankColor == Color.GREEN) {
      if(active){
         file = new File("tank_green_a.png");
      }
      else {
         file = new File("tank_green.png");
      }
   }
   return file;
}

public static BufferedImage rotate(BufferedImage image, double angle) {
   BufferedImage bufImg = toBufferedImage(image);
```

```
        double sin = Math.abs(Math.sin(angle)), cos = Math.abs(Math.cos(angle));
        int w = bufImg.getWidth(), h = bufImg.getHeight();
        int neww = (int) Math.floor(w * cos + h * sin), newh = (int) Math.floor(h *
cos + w * sin);
        BufferedImage result = new BufferedImage(neww, newh,
Transparency.TRANSLUCENT);
        Graphics2D g = result.createGraphics();
        g.translate((neww - w) / 2, (newh - h) / 2);
        g.rotate(angle, w / 2, h / 2);
        g.drawRenderedImage(bufImg, null);
        g.dispose();
        return result;
    }

    public static BufferedImage toBufferedImage(Image image) {
        if (image instanceof BufferedImage) {
            return (BufferedImage) image;
        }

        BufferedImage buff = new BufferedImage(image.getWidth(null),
image.getHeight(null),
                BufferedImage.TYPE_INT_ARGB);
        Graphics2D g = buff.createGraphics();
        g.drawImage(image, 0, 0, null);
        g.dispose();

        return buff;
    }
```

```
private class KeyController implements KeyListener {

    @Override
    public void keyTyped(KeyEvent arg0) {
    }

    @Override
    public void keyPressed(KeyEvent ke) {
        int keyCode = ke.getKeyCode();

        directionByKeyCode(keyCode);
        try {
            moveAction(keyCode);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
        skipStepAction(keyCode);
        try {
            fired(keyCode);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }

        skipProjectile(keyCode);
        repaint();
    }

    @Override
    public void keyReleased(KeyEvent arg0) {
```

```
    }

    private void skipStepAction(@NotNull int keyCode) {
        if(keyCode == KeyEvent.VK_Z) {
            tank.skipStep();
            System.out.println(color + " skip step");
        }
    }

    private void moveAction(@NotNull int keyCode) throws
InterruptedException {
        if(keyCode == KeyEvent.VK_SPACE) {
            Direction direction = tank.getDirection();
            System.out.println(color + " go to " + direction);
            if (direction != null && tank.isActive()) {
                tank.move(direction);
            }
        }
    }

    private void fired(@NotNull int keyCode) throws InterruptedException {
        if(keyCode == KeyEvent.VK_F){
            Direction direction = tank.getDirection();
            System.out.println(color + " fire to " + direction);
            if (direction != null && tank.isActive()) {
                if(tank.getProjectile()==0){
                    tank.shootDir();
                }
                if(tank.getProjectile()==1){
```

```
                tank.shootCell();

            }
        }
    }
}

private void directionByKeyCode(@NotNull int keyCode) {
    switch (keyCode) {
        case KeyEvent.VK_W:
            tank.setDirection(Direction.NORTH);
            break;
        case KeyEvent.VK_S:
            tank.setDirection(Direction.SOUTH);
            break;
        case KeyEvent.VK_A:
            tank.setDirection(Direction.WEST);
            break;
        case KeyEvent.VK_D:
            tank.setDirection(Direction.EAST);
            break;
    }
}

private int a = 0;
private JLabel label = new JLabel("Снаряд: простой");

private void skipProjectile(@NotNull int keyCode){
```

```
            if(a==0) {
                label.setFont(new Font("Verdana", 1, 10));
                add(label);
                a++;
            }

            if(keyCode == KeyEvent.VK_X){
                tank.setProjectile();

                if(tank.getProjectile()==1) {
                    label.setText("Снаряд: умный");
                }
                if(tank.getProjectile()==0) {
                    label.setText("Снаряд: простой");
                }
            }
        }
    }
}

// Виджет Штаба
public class HeadquarterWidget extends CellItemWidget{


    private final Headquarter headquarter;

    private final Color color;

    public HeadquarterWidget(Headquarter headquarter, Color color) {
```

```
    this.headquarter = headquarter;
    this.color = color;
    setPreferredSize(new Dimension(120,120));
}

@Override
protected BufferedImage getImage() {
    BufferedImage image = null;
    try {
        image = ImageIO.read(getHeadquarterImageFileByColor(color));
        image = ImageUtils.resizeImage(image, 120, 120);

    } catch (IOException e) {
        e.printStackTrace();
    }
    return image;
}

@Override
public CellWidget.Layer getLayer() {
    return CellWidget.Layer.BOTTOM;
}

@Override
protected Dimension getDimension() {
    return new Dimension(120, 120);
}
```

```
    private static File getHeadquarterImageFileByColor(Color headquarterColor)
{
        File file = null;
        if (headquarterColor == Color.BLUE)  {
            file =  new File("head_blue.png");
        }
        if (headquarterColor == Color.GREEN) {
            file = new File("head_green.png");
        }
        return file;
    }
}

// Виджет Снаряда
public class SimpleProjectileWidget extends ProjectileWidget{

    private final SimpleProjectile projectile;

    public SimpleProjectileWidget(SimpleProjectile projectile) {
        this.projectile = projectile;
        setPreferredSize(new Dimension(120,120));
    }

    @Override
    protected BufferedImage getImage() {
        BufferedImage image = null;
        try {
            image = ImageIO.read(new File("simple_projectile.png"));
            image = ImageUtils.resizeImage(image, 60, 60);
```

```
        } catch (IOException e) {
            e.printStackTrace();
        }
        return image;
    }

    @Override
    public CellWidget.Layer getLayer() {
        return CellWidget.Layer.TOP;
    }

    @Override
    protected Dimension getDimension() {
        return new Dimension(60, 60);
    }
}
```

3.8 Реализация ключевых тестовых случаев

```
// Тесты Ячейки
class CellTest {

    private Cell cell;

    public CellTest() {
    }



    @BeforeEach
    public void testSetup() {

        cell = new Cell();
    }

    @Test
    public void test_setTank_InEmptyCell() {
        Tank tank = new Tank();

        cell.setGame_object(tank);

        assertEquals(tank, cell.getGame_object());
        assertEquals(cell, tank.getPosition());
    }

    @Test
    public void test_takeTank_FromCellWithTank() {
```

```
        Tank tank = new Tank();

        cell.setGame_object(tank);

        assertEquals(tank, cell.takeGame_object());
        assertNull(tank.getPosition());
        assertNull(cell.getGame_object());
    }


    @Test
    public void test_setTank_ToCellWithTank() {
        Tank tank = new Tank();
        Tank newTank = new Tank();

        cell.setGame_object(tank);

        assertThrows(IllegalArgumentException.class, () ->
cell.setGame_object(newTank));
        assertEquals(tank, cell.getGame_object());
        assertEquals(cell, tank.getPosition());
        assertNull(newTank.getPosition());
    }

    @Test
    public void test_setNeighborCell() {
        Cell neighborCell = new Cell();
        Direction direction = Direction.NORTH;
```

```
        cell.setNeighbor(neighborCell, direction);

        assertEquals(neighborCell, cell.neighborCell(direction));
        assertEquals(cell,
neighborCell.neighborCell(direction.getOppositeDirection()));
    }


    @Test
    public void test_setNeighborCell_doubleSided() {
        Cell neighborCell = new Cell();
        Direction direction = Direction.NORTH;

        cell.setNeighbor(neighborCell, direction);
        neighborCell.setNeighbor(cell, direction.getOppositeDirection());
        assertEquals(neighborCell, cell.neighborCell(direction));
        assertEquals(cell,
neighborCell.neighborCell(direction.getOppositeDirection()));
    }

    @Test
    public void test_setNeighborCell_twoTimesInOneDirection() {
        Cell neighborCell = new Cell();
        Cell anotherCell = new Cell();
        Direction direction = Direction.NORTH;

        cell.setNeighbor(neighborCell, direction);
        assertThrows(IllegalArgumentException.class, () ->
cell.setNeighbor(anotherCell, direction));
```

```
        assertEquals(neighborCell, cell.neighborCell(direction));
        assertEquals(cell,
neighborCell.neighborCell(direction.getOppositeDirection()));
    }

    @Test
    public void test_isNeighbor_WhenNeighborCellExists() {
        Cell neighborCell = new Cell();
        Direction direction = Direction.NORTH;

        cell.setNeighbor(neighborCell, direction);
        assertEquals(direction, cell.isNeighbor(neighborCell));
    }

    @Test
    public void test_isNeighbor_WhenNeighborCellNotExists() {
        Cell neighborCell = new Cell();

        assertNull(cell.isNeighbor(neighborCell));
    }
}

// Тесты Поля
class FieldTest {

    private int eventCount = 0;

    class FieldObserver implements FieldActionListener {
```

```
    @Override
    public void tankDestroyObject(@NotNull FieldActionEvent event) {
        eventCount += 1;
    }

    @Override
    public void tankFired(@NotNull FieldActionEvent event) {

    }
}

private Field field;

@BeforeEach
public void testSetup() {
    eventCount = 0;
    field = new Field(2, 2);
    field.addFieldActionListener(new FieldObserver());
}

@Test
public void test_create_withCorrectParams() {
    Cell cell_0_0 = field.getCell(new Point(0, 0));
    Cell cell_0_1 = field.getCell(new Point(1, 0));
    Cell cell_1_0 = field.getCell(new Point(0, 1));
    Cell cell_1_1 = field.getCell(new Point(1, 1));

    assertEquals(Direction.SOUTH, cell_0_0.isNeighbor(cell_1_0));
    assertEquals(Direction.SOUTH, cell_0_1.isNeighbor(cell_1_1));
```

```
    assertEquals(Direction.NORTH, cell_1_1.isNeighbor(cell_0_1));
    assertEquals(Direction.NORTH, cell_1_0.isNeighbor(cell_0_0));
    assertEquals(Direction.EAST, cell_0_0.isNeighbor(cell_0_1));
    assertEquals(Direction.EAST, cell_1_0.isNeighbor(cell_1_1));
    assertEquals(Direction.WEST, cell_0_1.isNeighbor(cell_0_0));
    assertEquals(Direction.WEST, cell_1_1.isNeighbor(cell_1_0));
}

@Test
public void test_create_withNegativeWidth() {
    assertThrows(IllegalArgumentException.class, () -> new Field(-1, 1));
}

@Test
public void test_create_withZeroWidth() {
    assertThrows(IllegalArgumentException.class, () -> new Field(0, 1));
}

@Test
public void test_create_withNegativeHeight() {
    assertThrows(IllegalArgumentException.class, () -> new Field(1, -1));
}

@Test
public void test_create_withZeroHeight() {
    assertThrows(IllegalArgumentException.class, () -> new Field(1, 0));
}

@Test
```

```
    public void test_getTanksOnField_empty() {
        assertTrue(field.getTanksOnField().isEmpty());
    }

    @Test
    public void test_getTanksOnField_oneTank() {
        Tank tank = new Tank();
        field.getCell(new Point(0, 0)).setGame_object(tank);

        assertTrue(field.getTanksOnField().contains(tank));
        assertEquals(1, field.getTanksOnField().size());
    }

    @Test
    public void test_getTanksOnField_severalTanks() {
        Tank robot = new Tank();
        Tank anotherTank = new Tank();
        field.getCell(new Point(0, 0)).setGame_object(robot);
        field.getCell(new Point(1, 0)).setGame_object(anotherTank);

        assertTrue(field.getTanksOnField().containsAll(Arrays.asList(robot,
anotherTank)));
        assertEquals(2, field.getTanksOnField().size());
    }
}

// Тесты Игры
public class GameTest {
    private Game game;
```

```
    private enum Event {TANK_MOVED, TANK_SKIP_STEP,
TANK_FIRED}

    private List<Pare<Event, Tank>> events = new ArrayList<>();
    private List<Pare<Event, Tank>> expectedEvents = new ArrayList<>();

    class EventListener implements GameActionListener {

        @Override
        public void tankIsMoved(@NotNull GameActionEvent event) {
            events.add(new Pare<>(Event.TANK_MOVED, event.getTank()));
        }

        @Override
        public void tankIsSkipStep(@NotNull GameActionEvent event) {
            events.add(new Pare<>(Event.TANK_SKIP_STEP, event.getTank()));
        }

        @Override
        public void tankFired(@NotNull GameActionEvent event) {
            events.add(new Pare<>(Event.TANK_FIRED, event.getTank()));
        }

        @Override
        public void gameStatusChanged(@NotNull GameActionEvent event) {

        }
    }
```

```
    @BeforeEach
    public void testSetup() {
        events.clear();
        expectedEvents.clear();

        game = new Game(new Game_generationTest());
        game.addGameActionListener(new EventListener());
    }

    @Test
    public void test_finishGame() {
        game.finish();

        assertEquals(GameState.GAME_FINISHED_AHEAD_OF_SCHEDULE,
game.status());
    }

    @Test
    public void test_tankMoved_success() throws InterruptedException {
        Tank tank = game.activeTank();
        expectedEvents.add(new Pare<>(Event.TANK_MOVED, tank));

        game.activeTank().move(Direction.EAST);

        assertNotEquals(tank, game.activeTank());
        assertFalse(tank.isActive());
        assertEquals(expectedEvents, events);
        assertEquals(GameState.GAME_IS_ON, game.status());
```

```
    }

    @Test
    public void test_tankMoved_incorrectDirection() throws
InterruptedException {
        Tank tank = game.activeTank();
        game.activeTank().move(Direction.WEST);

        assertEquals(tank, game.activeTank());
        assertTrue(tank.isActive());
        assertEquals(expectedEvents, events);
        assertEquals(GameState.GAME_IS_ON, game.status());
    }

    @Test
    public void test_tankSkipStep() {
        Tank tank = game.activeTank();
        expectedEvents.add(new Pare<>(Event.TANK_SKIP_STEP, tank));

        game.activeTank().skipStep();

        assertNotEquals(tank, game.activeTank());
        assertFalse(tank.isActive());
        assertEquals(expectedEvents, events);
        assertEquals(GameState.GAME_IS_ON, game.status());
    }

    @Test
    public void test_winnerFound() throws InterruptedException {
```

```
        Tank tank = game.activeTank();

        game.activeTank().move(Direction.EAST);
        expectedEvents.add(new Pare<>(Event.TANK_MOVED, tank));

        tank = game.activeTank();

        game.activeTank().move(Direction.WEST);

        tank = game.activeTank();

        game.activeTank().setDirection(Direction.WEST);
        game.activeTank().shootDir();

        assertEquals(expectedEvents, events);
        assertTrue(tank.isActive());
        assertNotEquals(tank, game.winner());
    }
}

// Тесты Снаряда
public class ProjectileTest {

    @Test
    public void test_shoot_in_wall_back_to_back() {
        Game game;
        Tank tank;

        game = new Game(new Game_generationTest());
```

```
    tank = game.activeTank();
    tank.setDirection(Direction.SOUTH);
    tank.shootDir();

    assertEquals(GameState.GAME_IS_ON, game.status());
}

@Test
public void test_shoot_in_wall_with_distance() {
    Game game;
    Tank tank;

    game = new Game(new Game_generationTest());
    tank = game.activeTank();
    tank.setDirection(Direction.EAST);
    tank.shootDir();

    assertEquals(GameState.GAME_IS_ON, game.status());
}


//Этот тест почему-то надо запускать отдельно, и он проходит
@Test
public void test_kill_headquarter() {
    Game game;
    Tank tank;

    game = new Game(new Game_generationTest());
    tank = game.activeTank();
```

```
    tank.setDirection(Direction.NORTH);
    tank.shootDir();

    assertEquals(GameState.GAME_IS_ON, game.status());
}

@Test
public void test_kill_tank () throws InterruptedException {
    Game game;
    Tank tank;

    game = new Game(new Game_generationTest());
    tank = game.activeTank();

    game.activeTank().move(Direction.EAST);
    game.activeTank().move(Direction.SOUTH);
    game.activeTank().move(Direction.EAST);
    game.activeTank().move(Direction.WEST);
    game.activeTank().move(Direction.EAST);
    tank.setDirection(Direction.NORTH);
    tank.shootDir();

    assertEquals(GameState.GAME_IS_ON, game.status());
}

@Test
public void test_kill_wall () throws InterruptedException {
    Game game;
    Tank tank;
```

```
    game = new Game(new Game_generationTest());
    tank = game.activeTank();

    game.activeTank().move(Direction.EAST);
    game.activeTank().move(Direction.SOUTH);
    tank.setDirection(Direction.NORTH);
    tank.shootDir();

    assertEquals(GameState.GAME_IS_ON, game.status());
}

@Test
public void test_shoot_water () throws InterruptedException {
    Game game;
    Tank tank;

    game = new Game(new Game_generationTest());
    tank = game.activeTank();

    game.activeTank().move(Direction.EAST);
    game.activeTank().move(Direction.SOUTH);
    game.activeTank().move(Direction.EAST);
    game.activeTank().move(Direction.SOUTH);
    tank.setDirection(Direction.NORTH);
    tank.shootDir();

    assertEquals(GameState.GAME_IS_ON, game.status());
}
```

```
}

// Тесты Танка
public class TankTest {

    enum EVENT {TANK_MOVED, TANK_SKIP_STEP}

    private List<EVENT> events = new ArrayList<>();
    private List<EVENT> expectedEvents = new ArrayList<>();

    private class EventsListener implements TankActionListener {

        @Override
        public void tankIsMoved(@NotNull TankActionEvent event) {
            events.add(EVENT.TANK_MOVED);
        }

        @Override
        public void tankIsSkipStep(@NotNull TankActionEvent event) {
            events.add(EVENT.TANK_SKIP_STEP);
        }

        @Override
        public void tankFired(@NotNull TankActionEvent event) {

        }

        @Override
        public void tankFiredSmart(@NotNull TankActionEvent event) {
```

```
    }

    @Override
    public void tankDestroyObject(@NotNull TankActionEvent event) {

    }

    @Override
    public void myProjectileIsMoved(@NotNull TankActionEvent event) {

    }
}

private Cell cell;
private Cell neighborCell;
private final Direction direction = Direction.NORTH;

private Tank tank;

@BeforeEach
public void testSetup() {
    // clean events
    events.clear();
    expectedEvents.clear();

    // create tank
    tank = new Tank();
    tank.setActive(true);
```

```
    tank.addTankActionListener(new EventsListener());

    // create field
    cell = new Cell();
    neighborCell = new Cell();
    cell.setNeighbor(neighborCell, direction);
}

@Test
public void test_setActiveAndIsActive() {
    tank.setActive(true);

    assertTrue(tank.isActive());
    assertTrue(events.isEmpty());
}

@Test
public void test_canStayAtPosition_emptyCell() {
    assertTrue(Tank.canStayAtPosition(cell));
    assertTrue(events.isEmpty());
}

@Test
public void test_canStayAtPosition_cellWithTank() {
    cell.setGame_object(tank);

    assertFalse(Tank.canStayAtPosition(cell));
    assertTrue(events.isEmpty());
}
```

```
    @Test
    public void test_move_emptyCellInDirectionAndTankActive() throws
InterruptedException {
        cell.setGame_object(tank);

        tank.move(direction);

        expectedEvents.add(EVENT.TANK_MOVED);

        assertEquals(tank, neighborCell.getGame_object());
        assertEquals(neighborCell, tank.getPosition());
        assertNull(cell.getGame_object());
        assertEquals(expectedEvents, events);
    }

    @Test
    public void test_move_noCellInDirectionAndTankActive() throws
InterruptedException {
        neighborCell.setGame_object(tank);

        tank.move(Direction.NORTH);

        assertEquals(neighborCell, tank.getPosition());
        assertEquals(tank, neighborCell.getGame_object());
        assertTrue(events.isEmpty());
    }

    @Test
```

```
    public void test_move_emptyCellInDirectionAndTankNotActive() throws
InterruptedException {
        cell.setGame_object(tank);

        tank.setActive(false);
        tank.move(direction);

        assertEquals(tank, cell.getGame_object());
        assertEquals(cell, tank.getPosition());
        assertNull(neighborCell.getGame_object());
        assertTrue(events.isEmpty());
    }

    @Test
    public void test_skipStep_tankActive() {
        cell.setGame_object(tank);

        tank.skipStep();

        expectedEvents.add(EVENT.TANK_SKIP_STEP);

        assertEquals(cell, tank.getPosition());
        assertEquals(tank, cell.getGame_object());
        assertEquals(expectedEvents, events);
    }

    @Test
    public void test_skipStep_tankNotActive() {
        cell.setGame_object(tank);
```

```
        tank.setActive(false);

        tank.skipStep();

        assertEquals(cell, tank.getPosition());
        assertEquals(tank, cell.getGame_object());
        assertTrue(events.isEmpty());
    }
}
```

4 Вторая итерация разработки

4.1 Функциональные требования (сценарии)

1) Сценарий «Играть»

1. Пользователь инициирует начало Игры.
2. Игра создает при помощи Генерации поля Поле из Ячеек и размещает на нём два Танка и два Штаба к ним, Стены и Воду.
3. Игра запрашивает у Поля Танки, которые находятся на нем.
4. Игра случайным образом выбирает активный Танк.
5. Делать
    - 5.1. По указанию пользователя Танк перемещается на соседнюю Ячейку, стреляет в заданном направлении (или в заданную ячейку) или пропускает ход.
    - 5.2. Игра запрашивает у Поля Танки и Штабы, которые находятся на нем.
    - 5.3. Игра делает активным следующий живой Танк, который располагается на Поле.

    Пока на поле есть хотя бы один живой Танк или Штаб.
6. Игра считает победителем единственный Танк, который уничтожил Танк противника или Штаб противника.

2) Сценарий «Генерация поля создает Поле из Ячеек и размещает на нем два Танка, два Штаба, Стены и Воду»

1. Игра инициирует создание Поля размером NxM Ячеек посредством Генерации поля.

2. Генерация поля создает и расставляет стены внутри Поля.

3. Генерация поля создает и расставляет воду внутри Поля.

4. Генерация поля создает два Танка и помещает их на Поле.

6. Генерация поля создает, связывает с Танками и помещает два Штаба на Поле.

3) Сценарий «Танк стреляет Снарядом»

1. Пользователь хочет стрельнуть определенным Снарядом.

2. Танк разрешает себе стрельнуть (так как стрелять можно 1 раз за 3 хода).

3. Танк стреляет.

4. Танк обнуляет себе возможность стрелять.

5. Снаряд летит по определенной траектории в заданном Направлении или в заданную Ячейку.

6. Снаряд уничтожает определенную область Ячеек и в нее попадает препятствие.

7. Танк уничтожает Снаряд после окончания его полёта.

8. Препятствием оказывается Штаб, поэтому Игра заканчивается с определением победителя.

3.1) Альтернативный сценарий «Танк не может стрельнуть»

1. Сценарий начинается после пункта 1.

2. Танк запрещает себе стрельнуть (так как стрелять можно 1 раз за 3 хода).

3. Сценарий переходит к пункту 5.1 главного сценария.

3.2) Альтернативный сценарий «Снаряд попадает в Стену»

1. Сценарий начинается после пункта 7.

2. Препятствием оказывается Стена, поэтому Снаряд ее разрушает, а Танк уничтожает Снаряд.

3. Сценарий переходит к пункту 5.2 главного сценария.

3.3) Альтернативный сценарий «Снаряд попадает в Танк»

1. Сценарий начинается после пункта 7.

2. Препятствием оказывается Танк, поэтому Снаряд отнимает у него одну жизнь.

3. Сценарий переходит к пункту 5.2 главного сценария.

4) Сценарий «Танк перемещается на свободную Ячейку»

1. Танк запрашивает у Ячейки, в которой он находится, соседнюю Ячейку в Направлении своего движения.

2. Ячейка сообщает о Ячейке, с которой соседствует.

3. Танк спрашивает у соседней Ячейки, есть ли в ней непроходимое препятствие.

4. Танк просит Ячейку, в которой он находится, изъять его из нее.

5. Ячейка извлекает Танк из себя.
6. Танк просит соседнюю Ячейку поместить себя в нее.
7. Ячейка помещает Танк в себя.

4.1) Альтернативный сценарий «В соседней Ячейке находится непроходимое препятствие»

1. Сценарий начинается после пункта 3.
2. Соседняя Ячейка сообщает, что препятствие присутствует.
3. Сценарий переходит к пункту 5.1 главного сценария.

4.2) Альтернативный сценарий «Пропуск хода»

1. Сценарий выполняется вместо сценария 4.
2. Пользователь инициирует пропуск хода.
3. Сценарий переходит к пункту 5.2 главного сценария.

5) Сценарий «Досрочное завершение игры»

1. Сценарий начинается в любой точке главного сценария, когда пользователь инициирует завершение игры.
2. Игра завершается без определения победителя.

6) Сценарий «Снаряд летит по свободной Ячейке»

1. Снаряд запрашивает у Ячейки, в которой он находится, соседнюю Ячейку в Направлении своего движения.
2. Ячейка сообщает о Ячейке, с которой соседствует.

3. Снаряд спрашивает у соседней Ячейки, есть ли в ней непроходимое препятствие.
4. Снаряд просит Ячейку, в которой он находится, изъять его из нее.
5. Ячейка извлекает Снаряд из себя.
6. Снаряд просит соседнюю Ячейку поместить себя в нее.
7. Ячейка помещает Снаряд в себя, т.к. в ней нет другого непроходимого объекта.
8. Снаряд отнимает у себя единицу от дальности полёта.

4.2 Словарь предметной области

Игра знает о Поле. Игра управляет игровым циклом: определяет очередного игрока, определяет окончание, определяет победителя.

Генерация поля создает Поле и определяет начальную расстановку игровых элементов (танки, стены, лужи, штабы).

Поле - прямоугольная область, состоящая из Ячеек. Позволяет получить Танки и Штабы, находящиеся на поле.

Ячейка - квадратная область Поля. Знает о четырёх соседних Ячейках. В ней может находиться по отдельности Стена, Вода, Штаб, Танк. Она понимает, какой объект в ней находится. Ячейка может передать объекту, что его поразили.

Стена – разрушаемый объект одним Снарядом, заполняет всю Ячейку. Танк не может ее пройти, если не разрушит.

Вода – непроходимый объект, заполняет всю Ячейку.

Штаб – разрушаемый объект одним Снарядом, заполняет всю Ячейку. После разрушения Штаба игра заканчивается.

Снаряд – объект, которым стреляет Танк в выбранном Направлении или в выбранную Ячейку. Летит по определенной траектории. У него есть определенная дальность полёта. Снаряд имеет определенную область поражения. Сообщает Ячейке с препятствием, что он её поразил. Снаряд создает Танк, который его потом и убивает. Выбирается Снаряд с клавиатуры.

Танк – объект Игры, который может перемещаться в соседнюю Ячейку. Также умеет стрелять Снарядом 1 раз в 3 хода. Не может переходить

Стены, Воду и Штабы. Танк может пропустить ход и остаться в исходной Ячейке.

Активный танк – Танк, который может совершать действие в текущий ход.

Направление – класс для обозначения Направления перемещения Танка. Выбирается пользователем с клавиатуры.

4.3 Структура программы на уровне классов

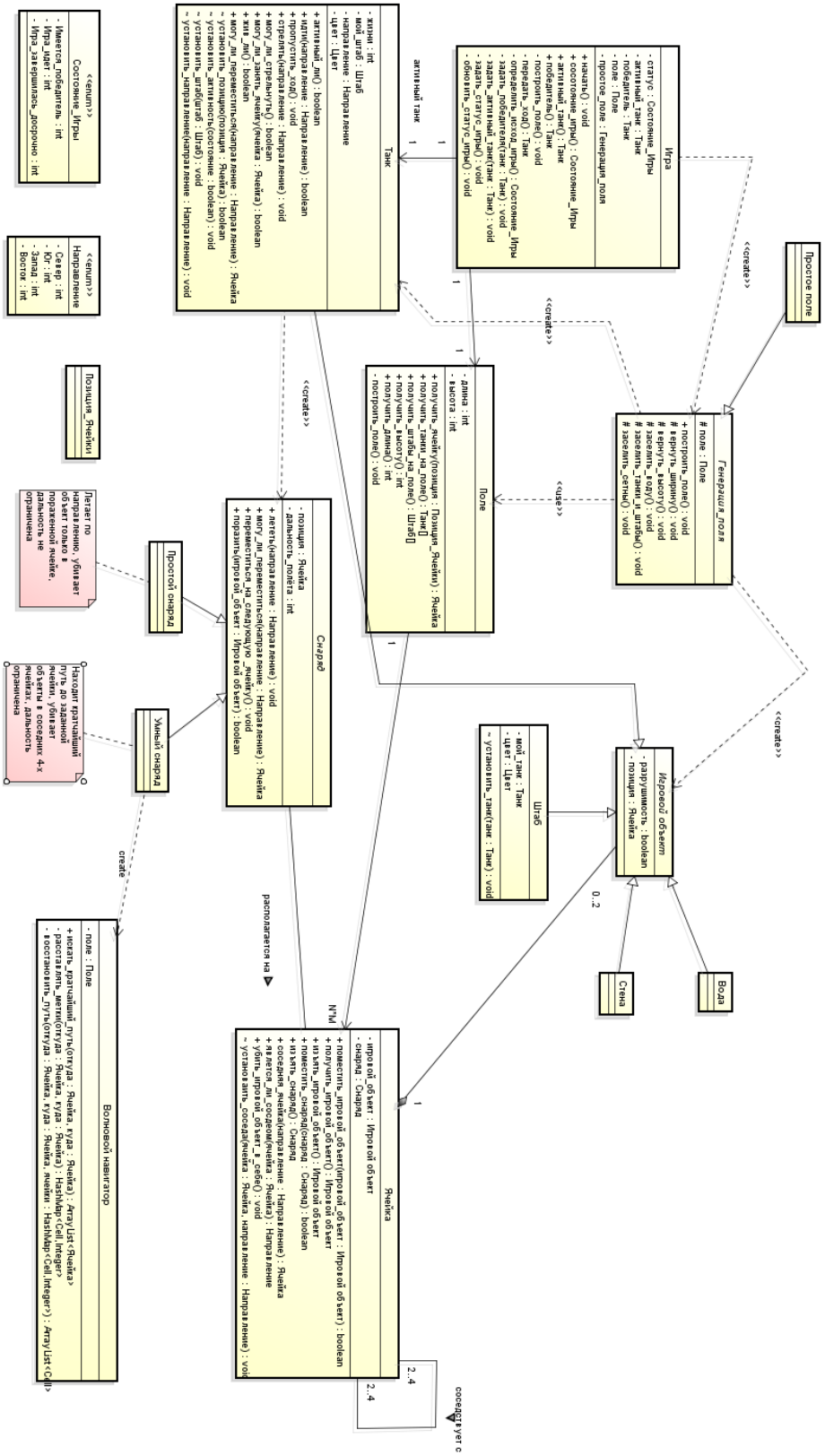


Диаграмма классов вычислительной модели с точкой расширения

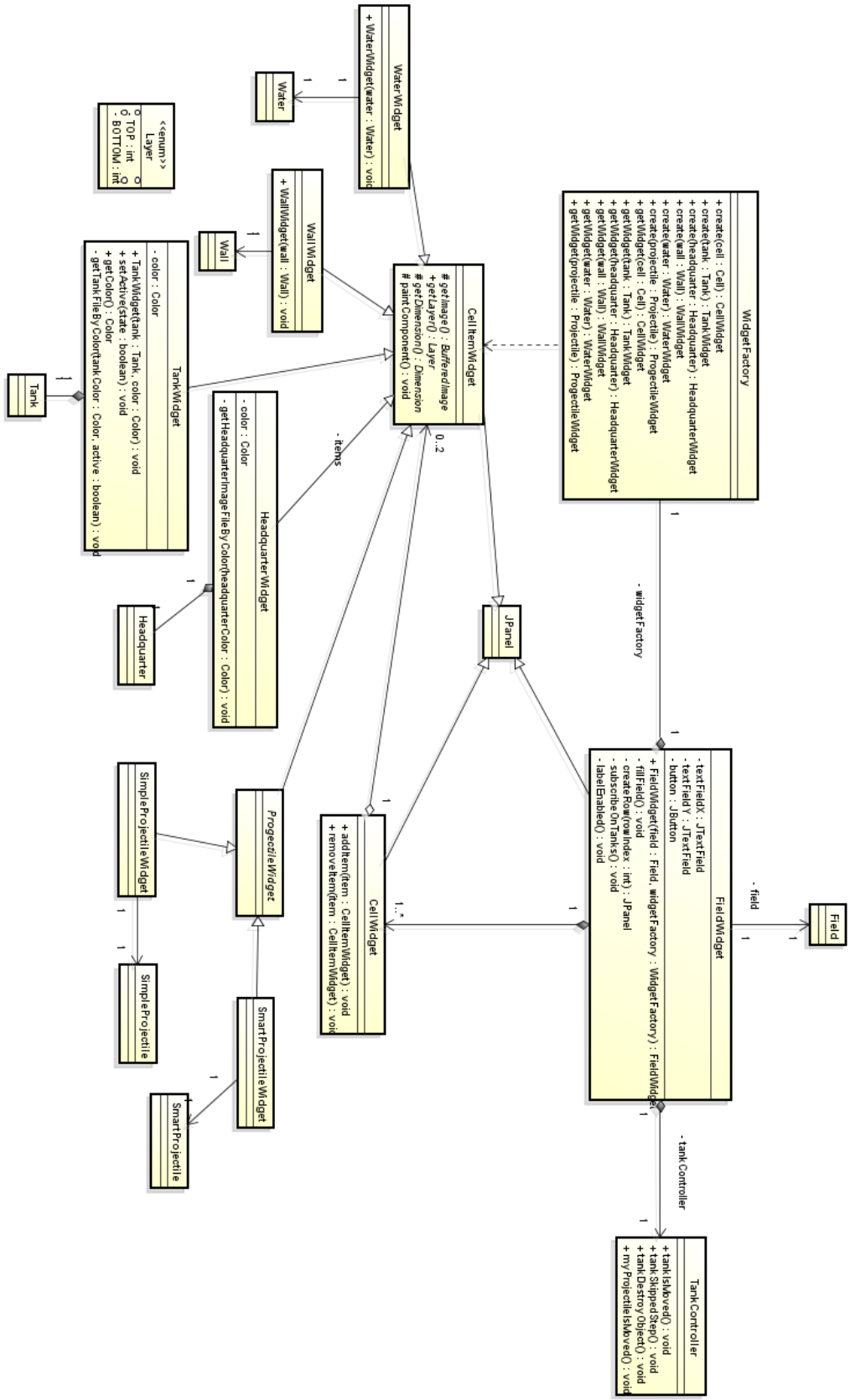
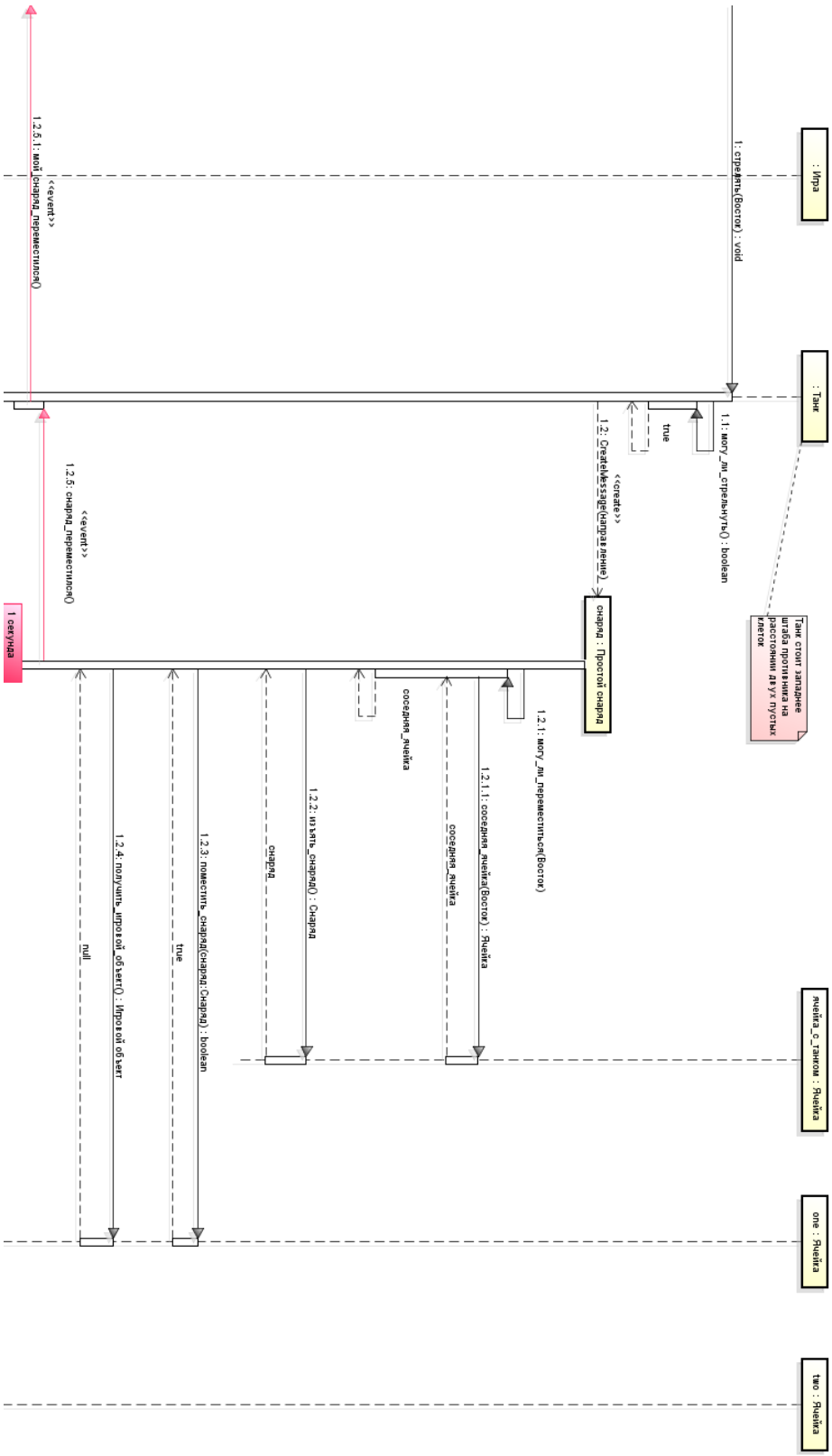
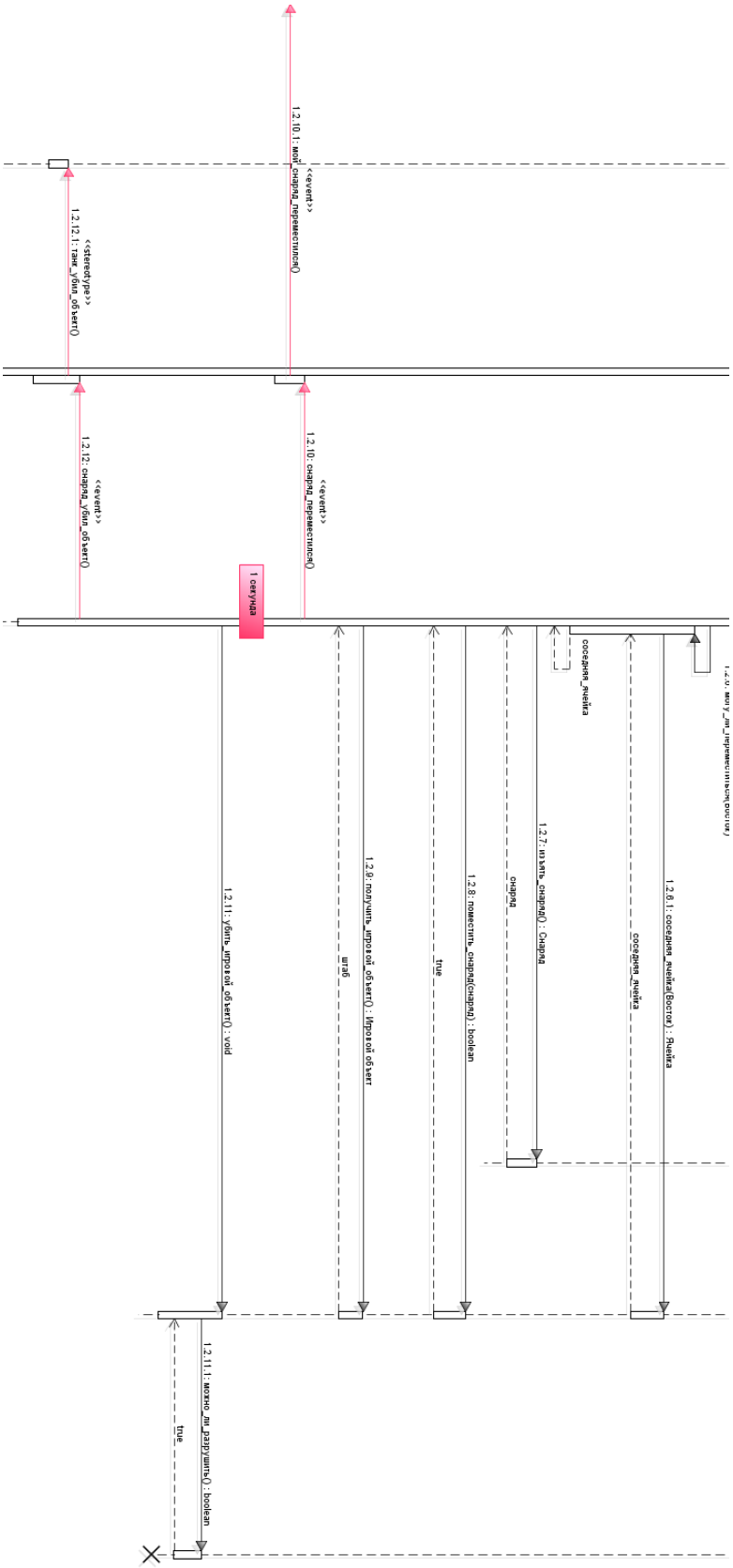


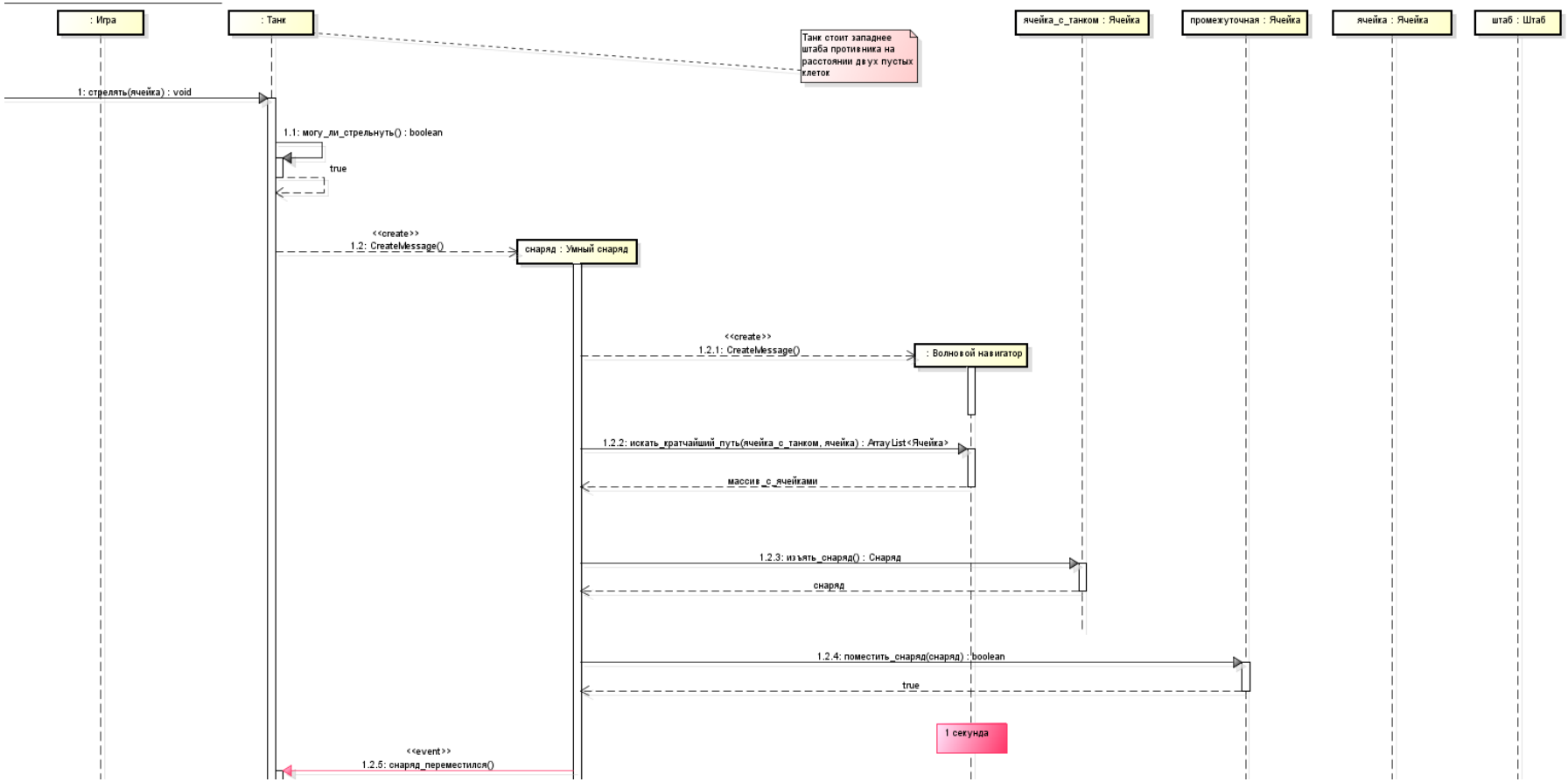
Диаграмма представления с точкой расширения

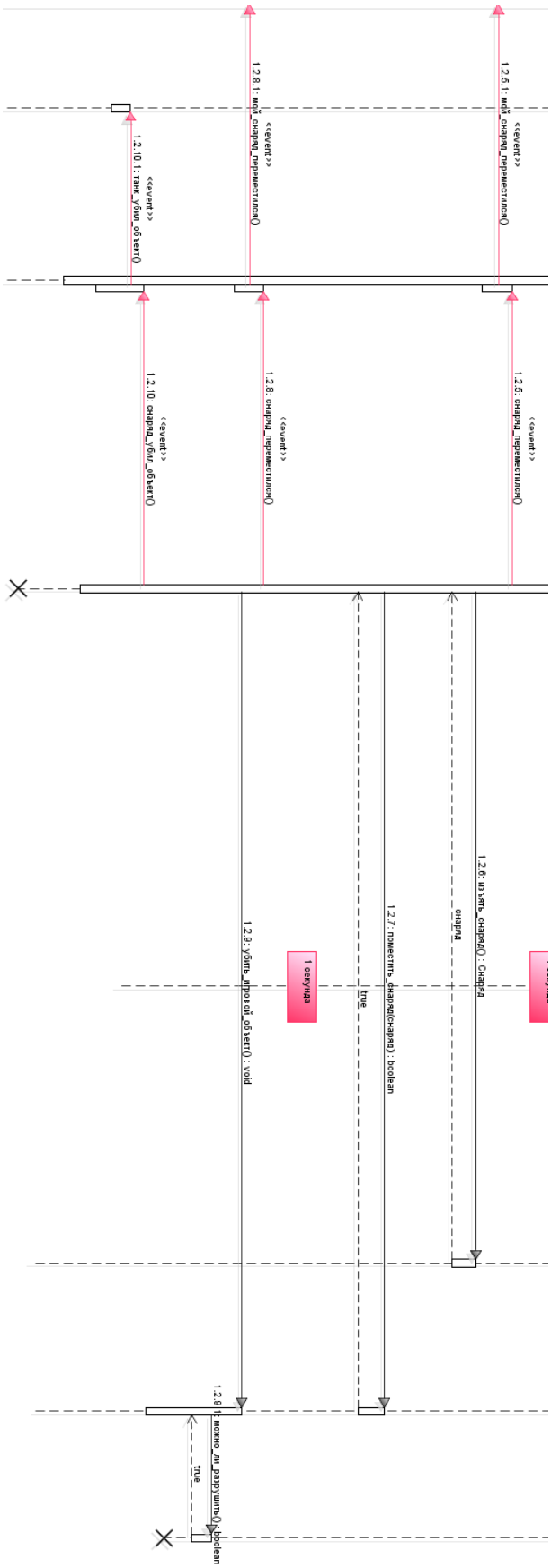
4.4 Типовые процессы в программе





Простой снаряд поразил разрушаемый игровой объект


: Игра
: Танк
Танк стоит западнее штаба противника на расстоянии двух пустых клеток
ячейка_с_танком : Ячейка
промежуточная : Ячейка
ячейка : Ячейка
штаб : Штаб
1: стрелять(ячейка) : void
1.1: могу_ли_стрельнуть() : boolean
true
<<create>>
1.2: CreateIwessage()
снаряд : Умный снаряд
<<create>>
1.2.1: CreateIwessage()
: Волновой навигатор
1.2.2: искать_кратчайший_путь(ячейка_с_танком, ячейка) : ArrayList<Ячейка>
массив_с_ячейками
1.2.3: изъять_снаряд() : Снаряд
снаряд
1.2.4: поместить_снаряд(снаряд) : boolean
true
1 секунда
<<event>>
1.2.5: снаряд_переместился()



Умный снаряд поразил разрушаемый объект

4.5 Человеко-машинное взаимодействие

Общий вид главного экрана программы представлен ниже. На нём располагается игровое поле, на котором изображено два танка и два штаба (зеленый и синий), стены, вода. Каждый объект занимает одну ячейку.

Рис.9. Общий вид главного экрана программы

Управление активным танком пользователь осуществляет с помощью клавиатуры.

W – направление вверх.
S – направление вниз.
A – направление влево.
D – направление вправо.
Space - идти.
F – стрелять.
Z – пропустить ход.
X – поменять снаряд.

Также при желании выстрелить снарядом не по направлению, а в определенную ячейку, нужно ввести её координаты в поле. Координату x в первую строку, координату y – во вторую. Вводить строго число без прочих символов. При неверном вводе покажется сообщение, представленное на рисунке 10.

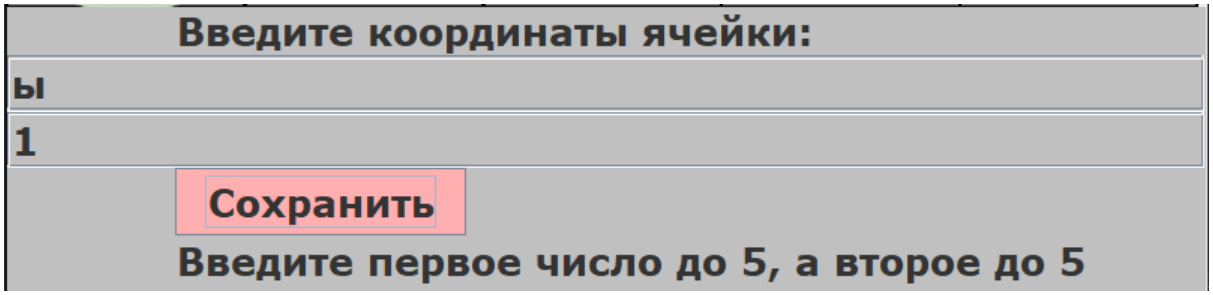


Рис.10. Предупреждение при неверном вводе координат

При верном вводе также покажется письмо, в какую ячейку вы стреляете. При нажатии на F, то есть при выстреле, это поле снова станет активным.

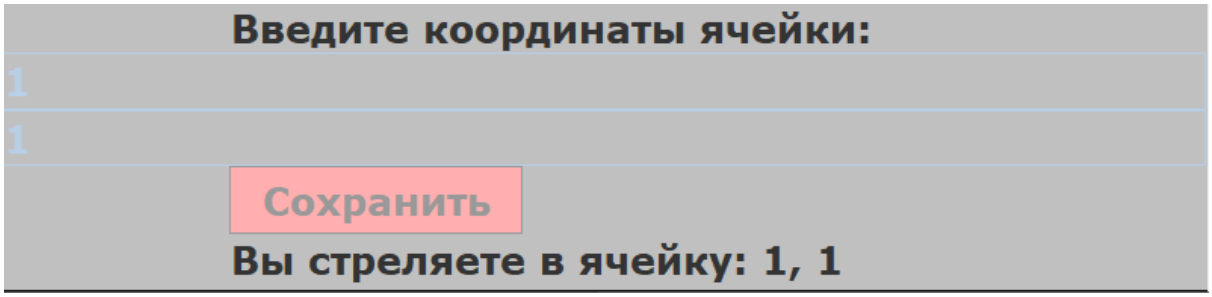


Рис.11. Сообщение о координатах

Изображение танка представлено на рисунке 12. Его дуло повернуто в ту сторону, которую он стреляет или ходит. Стрелять он может только один раз в 3 своих хода. Танк через все занятые клетки пройти не может. Над танком показано, каким снарядом он может стрельнуть.

Рис.12. Танк

Активный танк подсвечивается зеленым, чтобы было понятно, чей сейчас ход. Показано на рисунке 13.

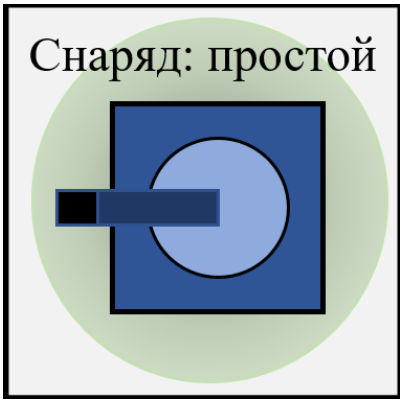


Рис.13. Активный танк

Также у танка есть штаб того же цвета, что и танк. Он представлен на рисунке 14.

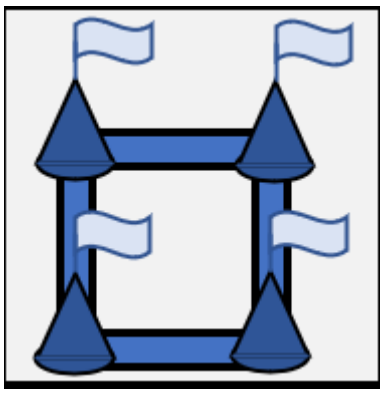


Рис.14. Штаб.

На поле есть препятствия. На рисунке 15 показана стена, которую можно разрушить одним выстрелом, а на рисунке 6 показана вода, ее разрушить нельзя.

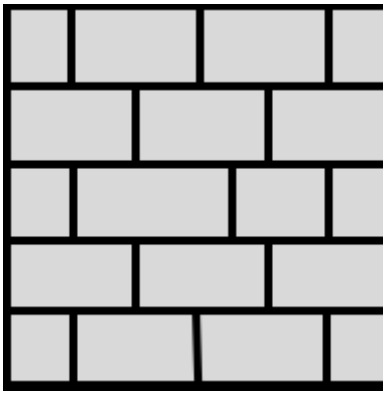


Рис.15. Стена.

Рис.16. Вода.

Танк стреляет снарядом, представленном на рисунке ниже. Он появляется в соседней ячейке, куда направлен танк.

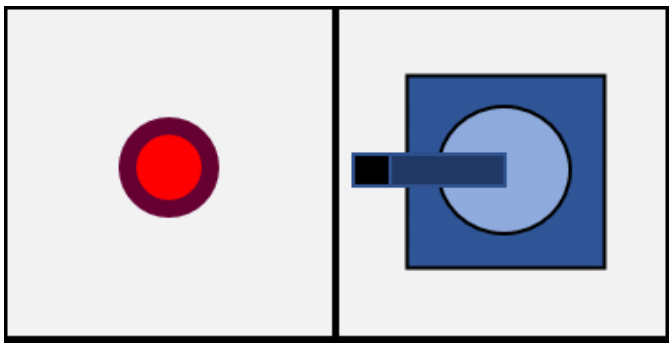
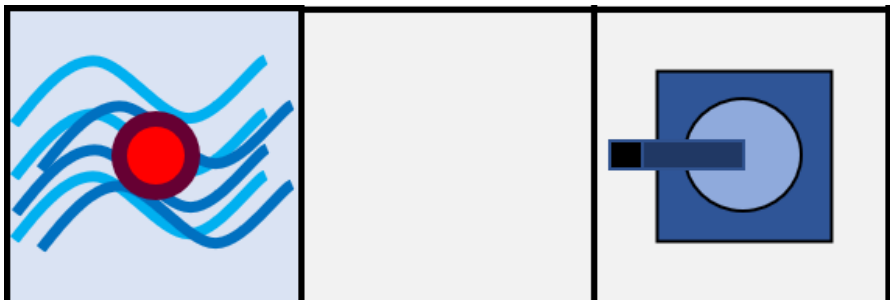
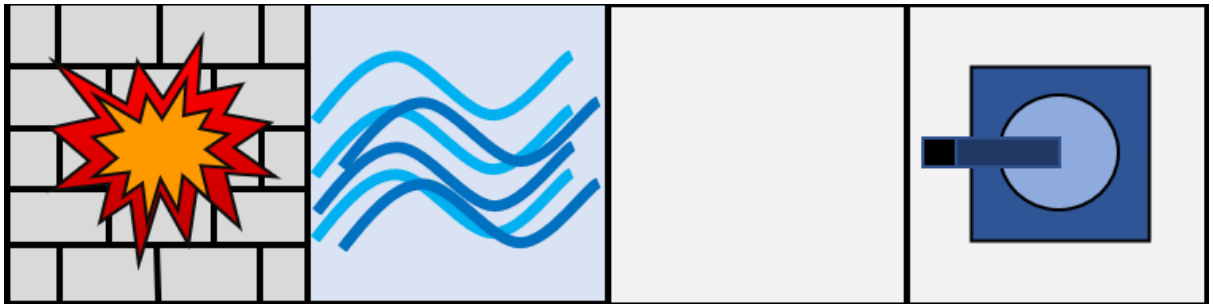


Рис.17. Танк стреляет Простым Снарядом.

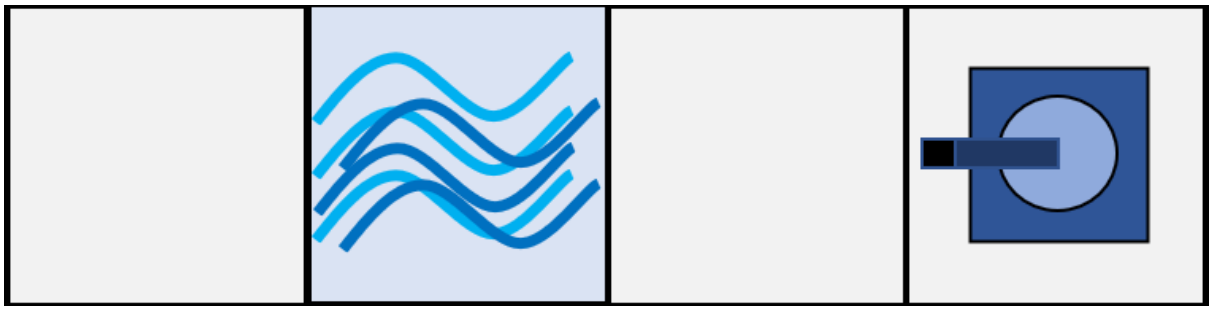
Снаряд пролетает над водой, но разрушает стену и штаб, а также наносит урон в виде отнимания одной жизни у танка. Пример полета снаряда и его взрывы ниже.



а)



б)



в)

Рис.18. Полет снаряда, его взрыв и исчезание объекта.

Также есть Умный снаряд. Он ведет себя также, как и Простой в плане рисовки. Он представлен на рисунке ниже.

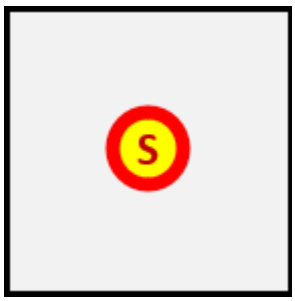


Рис.19. Умный снаряд

4.6 Реализация ключевых классов

```
// Класс Снаряд
public abstract class Projectile {

    // Позиция
    private Cell position;

    public Cell getPosition() {
        return position;
    }

    public void setPosition(Cell position) {
        this.position = position;
    }

    public Projectile(Cell position, int range_of_flight) {
        setPosition(position);
        setRange_of_flight(range_of_flight);
    }

    // Дальность полёта
    private int range_of_flight;

    public int getRange_of_flight() {
        return range_of_flight;
    }

    public void setRange_of_flight(int range_of_flight) {
```

```
        this.range_of_flight = range_of_flight;
    }

    // Снаряд летит с указанием направления или ячейки
    public void fly(Direction direction){};

    public void fly(Cell cell) throws InterruptedException {};

    // Переместиться
    public void move(Cell oldPosition, Cell newPosition) throws
InterruptedException {

    }
}

// Умный снаряд
public class SmartProjectile extends Projectile{

    private Cell position;

    public Cell getPosition() {
        return position;
    }

    public void setPosition(Cell position) {
        this.position = position;
    }

    private int range_of_flight;
```

```
public int getRange_of_flight() {
   return range_of_flight;
}

public void setRange_of_flight(int range_of_flight) {
   this.range_of_flight = range_of_flight;
}

public SmartProjectile(Cell position, int range_of_flight) {
   super(position, range_of_flight);
   setPosition(position);
   setRange_of_flight(range_of_flight);
}

Timer t;

public void fly(Cell cell) throws InterruptedException {

   WaveNavigator navigator = new WaveNavigator(cell.getField());
   ArrayList <Cell> array = navigator.findPath(getPosition(), cell);

   for(int i = 0; i<array.size()-1; i++) {
      if (getRange_of_flight() > 0) {

         move(array.get(i), array.get(i+1));
         range_of_flight--;
      }
   }
```

```
        if(array.size()!=0) {
            kill_cross();
        }
    }

    private void kill_cross() {
        if(getPosition().getGame_object() != null &&
getPosition().getGame_object().getDestructibility()) {
            GameObject game_object = getPosition().getGame_object();
            getPosition().kill();
            fireProjectileDestroyObject(game_object);
        }

        if(getPosition().neighborCell(Direction.SOUTH) != null){
            if(getPosition().neighborCell(Direction.SOUTH).getGame_object() !=
null &&
getPosition().neighborCell(Direction.SOUTH).getGame_object().getDestructibi
lity()) {
                GameObject game_object = getPosition().getGame_object();
                getPosition().neighborCell(Direction.SOUTH).kill();
                fireProjectileDestroyObject(game_object);
            }
        }
        if(getPosition().neighborCell(Direction.NORTH) != null){
            if(getPosition().neighborCell(Direction.NORTH).getGame_object() !=
null &&
getPosition().neighborCell(Direction.NORTH).getGame_object().getDestructibi
lity()) {
```

```
            GameObject game_object = getPosition().getGame_object();
            getPosition().neighborCell(Direction.NORTH).kill();
            fireProjectileDestroyObject(game_object);
        }
    }
    if(getPosition().neighborCell(Direction.WEST) != null){
        if(getPosition().neighborCell(Direction.WEST).getGame_object() !=
null &&
getPosition().neighborCell(Direction.WEST).getGame_object().getDestructibili
ty()) {
            GameObject game_object = getPosition().getGame_object();
            getPosition().neighborCell(Direction.WEST).kill();
            fireProjectileDestroyObject(game_object);
        }
    }
    if(getPosition().neighborCell(Direction.EAST) != null){
        if(getPosition().neighborCell(Direction.EAST).getGame_object() != null
&&
getPosition().neighborCell(Direction.EAST).getGame_object().getDestructibilit
y()) {
            GameObject game_object = getPosition().getGame_object();
            getPosition().neighborCell(Direction.EAST).kill();
            fireProjectileDestroyObject(game_object);
        }
    }
  }

  public void move(Cell oldPosition, Cell newPosition) throws
InterruptedException {
```

```
        newPosition.setProjectile(this);
        oldPosition.takeProjectile();
        this.setPosition(newPosition);
        fireProjectileIsMoved(oldPosition,newPosition);
    }

    // events
    private ArrayList<ProjectileActionListener> projectileListListener = new
ArrayList<>();

    public void addProjectileActionListener(ProjectileActionListener listener) {
        projectileListListener.add(listener);
    }

    public void removeProjectileActionListener(ProjectileActionListener
listener) {
        projectileListListener.remove(listener);
    }

    private void fireProjectileDestroyObject(GameObject game_object) {
        for(ProjectileActionListener listener: projectileListListener) {
            ProjectileActionEvent event = new ProjectileActionEvent(listener);
            event.setProjectile(this);
            event.setGame_object(game_object);
            listener.projectileDestroyObject(event);
        }
    }
```

```
    private void fireProjectileIsMoved(Cell oldPosition, Cell newPosition)
throws InterruptedException {
        for(ProjectileActionListener listener: projectileListListener) {
            ProjectileActionEvent event = new ProjectileActionEvent(listener);
            event.setProjectile(this);
            event.setFromCell(oldPosition);
            event.setToCell(newPosition);
            listener.projectileIsMoved(event);
        }
    }

    private void fireProjectile() throws InterruptedException {
        for(ProjectileActionListener listener: projectileListListener) {
            ProjectileActionEvent event = new ProjectileActionEvent(listener);
            event.setProjectile(this);
            listener.projectile(event);
        }
    }
}

// Простой снаряд
public class SimpleProjectile extends Projectile{

    private Cell position;

    public Cell getPosition() {
        return position;
    }
```

```
public void setPosition(Cell position) {
    this.position = position;
}

public SimpleProjectile(Cell position, int range_of_flight) {
    super(position, range_of_flight);
    setPosition(position);
    setRange_of_flight(range_of_flight);
}

private int range_of_flight;

public int getRange_of_flight() {
    return range_of_flight;
}

public void setRange_of_flight(int range_of_flight) {
    this.range_of_flight = range_of_flight;
}

Timer t;

public void fly(Direction direction) {

    final Cell[] newPosition = {canMove(direction)};
    final GameObject[] game_object = {null};

    ActionListener performer = new ActionListener() {
        final int a = 0;
```

```
        @Override
        public void actionPerformed(ActionEvent e) {

            if (newPosition[0] != null && t.isRunning()) {
                try {
                    move(getPosition(), newPosition[0]);
                } catch (InterruptedException ex) {
                    throw new RuntimeException(ex);
                }
                game_object[0] = newPosition[0].getGame_object();
                if (game_object[0] != null && game_object[0].getDestructibility())
{
                    newPosition[0].kill();
                    fireProjectileDestroyObject(game_object[0]);
                    t.stop();
                    try {
                        fireProjectile();
                    } catch (InterruptedException ex) {
                        throw new RuntimeException(ex);
                    }
                }
                newPosition[0] = canMove(direction);
                System.out.println("a");
            }
            else {
                t.stop();
                try {
                    fireProjectile();
                } catch (InterruptedException ex) {
```

```
                    throw new RuntimeException(ex);
                }
            }
        }
    };

    t = new Timer(2000, performer);
    t.addActionListener(performer);
    t.start();
  }

  public void move(Cell oldPosition, Cell newPosition) throws
InterruptedException {

    newPosition.setProjectile(this);
    oldPosition.takeProjectile();
    this.setPosition(newPosition);
    fireProjectileIsMoved(oldPosition,newPosition);
  }

  private Cell canMove(@NotNull Direction direction) {
    Cell result = null;
    if(getPosition()!=null){
      Cell neighborCell = getPosition().neighborCell(direction);

      if(neighborCell != null) {
        result = neighborCell;
      }
    }
```

```
        return result;
    }

    // events
    private ArrayList<ProjectileActionListener> projectileListListener = new
ArrayList<>();

    public void addProjectileActionListener(ProjectileActionListener listener) {
        projectileListListener.add(listener);
    }

    public void removeProjectileActionListener(ProjectileActionListener
listener) {
        projectileListListener.remove(listener);
    }

    private void fireProjectileDestroyObject(GameObject game_object) {
        for(ProjectileActionListener listener: projectileListListener) {
            ProjectileActionEvent event = new ProjectileActionEvent(listener);
            event.setProjectile(this);
            event.setGame_object(game_object);
            listener.projectileDestroyObject(event);
        }
    }

    private void fireProjectileIsMoved(Cell oldPosition, Cell newPosition)
throws InterruptedException {
        for(ProjectileActionListener listener: projectileListListener) {
```

```
            ProjectileActionEvent event = new ProjectileActionEvent(listener);
            event.setProjectile(this);
            event.setFromCell(oldPosition);
            event.setToCell(newPosition);
            listener.projectileIsMoved(event);
        }
    }

    private void fireProjectile() throws InterruptedException {
        for(ProjectileActionListener listener: projectileListListener) {
            ProjectileActionEvent event = new ProjectileActionEvent(listener);
            event.setProjectile(this);
            listener.projectile(event);
        }
    }
}

// Волновой навигатор
public class WaveNavigator{

    Field _field;

    public WaveNavigator(Field field) {
        this._field = field;
    };

    public ArrayList<Cell> findPath(Cell from, Cell to) {
        HashMap<Cell, Integer> cellMarks = waveProrogation(from, to);
```

```
    ArrayList<Cell> reversPath = pathRecovery(from, to, cellMarks);
    ArrayList<Cell> path = new ArrayList<>();
    for(int i = reversPath.size()-1; i >= 0; i--)
        path.add(reversPath.get(i));
    return path;
}

// Этап распространения волны по полю. Расставляются метки.
private HashMap<Cell, Integer> waveProrogation(Cell from, Cell to) {
    HashMap<Cell, Integer> cellMarks = new HashMap<>();

    int mark = 1;
    for (Cell cell : _field.cells_) {
        cellMarks.put(cell, 0);
    }
    cellMarks.replace(from, mark);

    while(cellMarks.get(to) == 0 && mark <= cellMarks.size()) {
        for (Cell cell : _field.cells_) {
            if (cellMarks.get(cell) == mark) {
                ArrayList<Direction> neighborsDir = cell.neighbors();
                for (Direction obj : neighborsDir) {
                    if (cellMarks.get(cell.neighborCell(obj)) == 0)
                        cellMarks.replace(cell.neighborCell(obj), mark + 1);
                }
            }
        }
        mark++;
    }
```

```
        return cellMarks;
    }

    // Восстановление пути
    private ArrayList<Cell> pathRecovery(Cell from, Cell to, HashMap<Cell,
Integer> cellMarks) {
        ArrayList<Cell> path = new ArrayList<>();
        if(cellMarks.get(to) != 0) {
            path.add(to);
            Cell curCell = to;
            while (curCell != from) {
                ArrayList<Direction> neighborsDir = curCell.neighbors();
                for (int i = 0; i < neighborsDir.size(); i++) {
                    Direction dir = neighborsDir.get(i);
                    if (cellMarks.get(curCell.neighborCell(dir)) ==
cellMarks.get(curCell) - 1) {
                        curCell = curCell.neighborCell(dir);
                        path.add(curCell);
                        i = neighborsDir.size();
                    }
                }
            }
        }
        return path;
    }
}

// Класс виджет Снаряда
public abstract class ProjectileWidget extends CellItemWidget{
```

```
    @Override
    protected BufferedImage getImage() {
        return null;
    }

    @Override
    public CellWidget.Layer getLayer() {
        return null;
    }

    @Override
    protected Dimension getDimension() {
        return null;
    }

    private static File getProjectileImageFile() {
        return null;
    }

}

// Виджет Простого Снаряда
public class SimpleProjectileWidget extends ProjectileWidget{

    private final SimpleProjectile projectile;

    public SimpleProjectileWidget(SimpleProjectile projectile) {
```

```
        this.projectile = projectile;
        setPreferredSize(new Dimension(120,120));
    }

    @Override
    protected BufferedImage getImage() {
        BufferedImage image = null;
        try {
            image = ImageIO.read(new File("simple_projectile.png"));
            image = ImageUtils.resizeImage(image, 60, 60);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return image;
    }

    @Override
    public CellWidget.Layer getLayer() {
        return CellWidget.Layer.TOP;
    }

    @Override
    protected Dimension getDimension() {
        return new Dimension(60, 60);
    }
}

// Класс виджета Умного Снаряда
public class SmartProjectileWidget extends ProjectileWidget {
```

```
    private final SmartProjectile projectile;

    public SmartProjectileWidget(SmartProjectile projectile) {
        this.projectile = projectile;
        setPreferredSize(new Dimension(120,120));
    }

    @Override
    protected BufferedImage getImage() {
        BufferedImage image = null;
        try {
            image = ImageIO.read(new File("smart_projectile.png"));
            image = ImageUtils.resizeImage(image, 60, 60);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return image;
    }

    @Override
    public CellWidget.Layer getLayer() {
        return CellWidget.Layer.TOP;
    }
    @Override
    protected Dimension getDimension() {
        return new Dimension(60, 60);
    }
}
```

4.7 Реализация ключевых тестовых случаев

```
// Тест Умного Снаряда
public class SmartProjectileTest {

    @Test
    public void test_shoot_in_free_cell() throws InterruptedException {
        Game game;
        Tank tank;

        game = new Game(new Game_generationTest());
        tank = game.activeTank();
        tank.setProjectileCell(game.getGameField().getCell(new Point(1,1)));
        tank.shootCell();

        assertEquals(GameState.GAME_IS_ON, game.status());

        // Проверка на убийство крестом, то есть соседи должны быть
уничтожены
        assertNull(game.getGameField().getCell(new Point(1,
0)).getGame_object());
    }

    @Test
    public void test_shoot_in_cell_with_headquarter() throws
InterruptedException {
        Game game;
        Tank tank;
```

```
        game = new Game(new Game_generationTest());
        tank = game.activeTank();
        tank.setProjectileCell(game.getGameField().getCell(new Point(3,0)));
        tank.shootCell();

        assertEquals(GameState.WINNER_FOUND, game.status());


        assertNull(game.getGameField().getCell(new Point(4,
0)).getGame_object());
        assertNotNull(game.getGameField().getCell(new Point(2,
0)).getGame_object());
    }

    @Test
    public void test_shoot_in_cell_with_tank() throws InterruptedException {
        Game game;
        Tank tank;

        game = new Game(new Game_generationTest());
        tank = game.activeTank();
        tank.setProjectileCell(game.getGameField().getCell(new Point(4,0)));
        tank.shootCell();

        assertEquals(GameState.WINNER_FOUND, game.status());


        assertNull(game.getGameField().getCell(new Point(3,
0)).getGame_object());
```

```
    }

    @Test
    public void test_shoot_in_cell_with_tank_surrounded_walls() throws
InterruptedException {
        Game game;
        Tank tank;

        Wall wall = new Wall();
        wall.setDestructibility(true);

        game = new Game(new Game_generationTest());

        game.getGameField().getCell(new Point(4,1)).setGame_object(wall);
        game.getGameField().getCell(new Point(3,1)).setGame_object(wall);
        game.getGameField().getCell(new Point(3,2)).setGame_object(wall);
        game.getGameField().getCell(new Point(2,1)).setGame_object(wall);

        tank = game.activeTank();
        tank.setProjectileCell(game.getGameField().getCell(new Point(3,0)));
        tank.shootCell();

        assertEquals(GameState.WINNER_FOUND, game.status());
    }

    @Test
    public void test_shoot_in_cell_with_tank_surrounded_water() throws
InterruptedException {
        Game game;
```

```
        Tank tank;

        Water water = new Water();
        water.setDestructibility(false);

        game = new Game(new Game_generationTest());

        game.getGameField().getCell(new Point(4,1)).setGame_object(water);
        game.getGameField().getCell(new Point(3,1)).setGame_object(water);
        game.getGameField().getCell(new Point(2,1)).setGame_object(water);

        tank = game.activeTank();
        tank.setProjectileCell(game.getGameField().getCell(new Point(3,0)));
        tank.shootCell();

        assertEquals(GameState.WINNER_FOUND, game.status());
    }
}
```

5 Список использованной литературы и других источников

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

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

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

Перечень замечаний к работе