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\_01)

Студент: Деменков Д. В.

Группа: ПрИн-366

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: Деменков Д. В.

Группа: ПрИн-366

1. Тема: «Проектирование и реализация программы с использованием объектно-ориентированного подхода» (индивидуальное задание – вариант №17\_01)

Утверждена приказом от «09» января 2023г. № 13-ст

2. Срок представления работы к защите «01» июня 2023 г.

3. Содержание пояснительной записки:

формулировка задания, требования к программе, структура программы, типовые процессы в программе, человеко-машинное взаимодействие, код программы и модульных тестов

4. Перечень графического материала:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Дата выдачи задания «15» февраля 2023 г.

Руководитель проекта: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Литовкин Д. В.

Задание принял к исполнению: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Деменков Д. В.

«15» февраля 2023 г.

**Содержание**

[1 Формулировка задания 4](#_Toc136466176)

[2 Нефункциональные требования 5](#_Toc136466177)

[3 Первая итерация разработки 6](#_Toc136466178)

[3.1 Формулировка упрощённого варианта задания 6](#_Toc136466179)

[3.2 Функциональные требования (сценарии) 7](#_Toc136466180)

[3.3 Словарь предметной области 13](#_Toc136466181)

[3.4 Структура программы на уровне классов 14](#_Toc136466182)

[3.5 Типовые процессы в программе 16](#_Toc136466183)

[3.6 Человеко-машинное взаимодействие 28](#_Toc136466184)

[3.7 Реализация ключевых классов 32](#_Toc136466185)

[3.8 Реализация ключевых тестовых случаев 62](#_Toc136466186)

[4 Вторая итерация разработки 85](#_Toc136466187)

[4.1 Формулировка модифицированного варианта задания. 85](#_Toc136466188)

[4.2 Функциональные требования (сценарии) 86](#_Toc136466189)

[4.3 Словарь предметной области. 92](#_Toc136466190)

[4.4 Структура программы на уровне классов 94](#_Toc136466191)

[4.5 Типовые процессы в программе 96](#_Toc136466192)

[4.6 Человеко машинное взаимодействие 103](#_Toc136466193)

[4.7 Реализация ключевых классов 105](#_Toc136466194)

[4.8 Реализация ключевых тестовых случаев 127](#_Toc136466195)

[5. Список используемой литературы 146](#_Toc136466196)

# 1 Формулировка 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

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

**Вариативность**: предусмотреть возможность создания других "препятствий", имеющих различное поведение при попадании снаряда: собственное разрушение, разрушение других объектов, возможность перемещения танка и снаряда.

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

* препятствие "заросли" - танки могут заезжать в заросли, это делает их невидимыми для противника - снаряд пролетает насквозь зарослей, не уничтожая танки и заросли.
* препятствие "бочка мазута" - при попадании она взрывается и наносит урон рядом стоящим объектам (возможна цепная реакция, которая должна анимироваться)

# 2 Нефункциональные требования

1. Программа должна быть реализована на языке Java SE 18 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. **По указанию пользователя** Игра стартует.

2. **По указанию Игры** Поле создаёт Ячейки и формирует из них себя.

3. **По указанию Игры** Окружение создает и размещает на Поле Препятствия, Танки и Штабы.

4. **В ответ на запрос Игры** Поле сообщает о Танках, которые

находятся на нём.

5. Игра случайным образом выбирает один из Танков и делает его

активным.

6. **Делать**

6.1. **По указанию пользователя** активному Танку задается направление хода.

6.2. **По указанию пользователя** Танк перемещается в соседнюю Ячейку в заданном направлении и сообщает об этом Игре.

6.3. Игра делает активным другой Танк.

**Пока** оба Танка живы и ни один Штаб не уничтожен.

7. Игра считает победителем выживший Танк с целым Штабом.

8. **Сценарий завершается.**

**2) Дочерний сценарий** «Окружение создает и размещает на Поле Препятствия, Танки и Штабы».

1. Окружение создает и помещает Штабы в Ячейки на противоположных сторонах Поля.

2. Окружение создает Препятствия и помешает их в свободные Ячейки произвольным образом.

3. Окружение создает Танки и помещает их в свободные Ячейки.

4. **Сценарий завершается.**

3) **Дочерний сценарий** «Танку задается направление хода».

1. **Пока** пользователь не определится с направление хода **делать**

1.1. **По указанию пользователя** Танк меняет направление хода на указанное.

2. **Сценарий завершается.**

4) **Дочерний сценарий** «Танк перемещается в соседнюю Ячейку в заданном направлении».

1. **Пока** ход не будет завершен **делать**

1.1. **По указанию пользователя** Танк решает переместиться в соседнюю Ячейку в направлении хода.

1.2. **В ответ на запрос Танка** Ячейка сообщает о соседней Ячейке в направлении хода.

1.3. **По указанию Танка** соседняя Ячейка помещает его в себя, так как в ней больше нет объектов.

1.4. **По указанию Танка** Ячейка извлекает его из себя.

1.5. Танк обновляет счетчик перезарядки орудия.

1.6. Танк сообщает Игре о завершении хода.

2. **Сценарий завершается.**

4.1) **Альтернативный сценарий** «В соседней Ячейке находится объект, препятствующий перемещению Танка». Сценарий выполняется с пункта п. 1.3 сценария 4.

1. **В ответ на запрос Танка** соседняя Ячейка сообщает о том, что она не может поместить в себя Танк, т. к. в ней находится объект, с которым Танк не может сталкиваться (Танк противника, Штаб, Препятствие).

2. Танк остается в текущей Ячейке.

3. **Сценарий продолжается с п. 1 сценария 4.**

4.2) **Альтернативный сценарий** «Танк находится на краю поля». Сценарий выполняется с пункта п. 1.2 сценария 4.

1. **В ответ на запрос Танка** Ячейка сообщает, что у нее нет соседа в направлении хода.

2. Танк решает, что не он не может передвигаться в указанном направлении, и остается в текущей Ячейке.

3. **Сценарий продолжается с п. 1 сценария 4.**

4.3) **Альтернативный сценарий** «Танк совершает выстрел». Сценарий выполняется с п. 1.1. сценария 4.

1. **По указанию пользователя** Танк решает совершить выстрел в направлении хода.

2. **В ответ на запрос Танка** Ячейка сообщает о соседней Ячейке в направлении хода.

3. **Если** оружие Танка не на перезарядке **делать**

3.1. Танк размещает Снаряд в соседней Ячейке и сообщает ему направление выстрела.

3.2. Снаряд поражает Танк противника, находящийся в направлении выстрела.

3.3. Танк начинает перезарядку орудия.

4. **Сценарий продолжается с п. 1.7 сценария 4.**

4.4) **Альтернативный сценарий** «Пользователь решает пропустить ход». Сценарий выполняется с п. 1.1. сценария 4.

1. **По указанию пользователя** Танк завершает ход.

2. **Сценарий продолжается с п. 1 сценария 4.**

5) Дополнительный сценарий «Снаряд поражает Танк противника, находящийся в направлении выстрела».

1. **Делать**

1.1**. По указанию Снаряда** текущая Ячейка помещает его в себя и сталкивает с Танком, который уже находится в ячейке.

1.2. **При столкновении со Снарядом** Танк уменьшает свое количество здоровья на 1.

1.3. **При столкновении с Танком** Снаряд решает уничтожиться.

1.4. **По запросу Снаряда** Ячейка, в которой он находится, извлекает его из себя.

**Пока** Снаряд находится на Поле.

2. **Сценарий завершается**.

5.1) Альтернативный сценарий «Снаряд поражает Штаб, находящийся в направлении выстрела». Сценарий выполняется с п. 1.1. сценария 5.

1. **По указанию Снаряда** текущая Ячейка помещает его в себя и сталкивает со Штабом, который находится в ячейке.

2. **При столкновении со Снарядом** Штаб решает уничтожится.

3. **По запросу Штаба** Ячейка, в которой он находится, извлекает его из себя.

4. **При столкновении со Штабом** Снаряд решает уничтожится.

5. **По запросу Снаряда** Ячейка, в которой он находится, извлекает его из себя.

6. **Сценарий завершается.**

5.2) Альтернативный сценарий «Снаряд поражает Стену, находящуюся в направлении выстрела». Сценарий выполняется с п. 1.1. сценария 5.

1. **По указанию Снаряда** текущая Ячейка помещает его в себя и сталкивает со Стеной, которая находится в ячейке.

2. **При столкновении со Снарядом** Стена решает уничтожится.

3. **По запросу Стены** Ячейка, в которой он находится, извлекает его из себя.

4. **При столкновении со Стеной** Снаряд решает уничтожится.

4. **По запросу Снаряда** Ячейка, в которой он находится, извлекает его из себя.

5. **Сценарий завершается.**

5.3) Альтернативный сценарий «Снаряд пролетает через Воду, находящуюся в направлении выстрела». Сценарий выполняется с п. 1.1. сценария 5.

1. **По указанию Снаряда** текущая Ячейка помещает его в себя и сталкивает с Водой, которая находится в ячейке.

2. **При столкновении с Водой** Снаряд решает двигаться дальше.

3. **По запросу Снаряда** текущая Ячейка сообщает ему о соседней Ячейке в направлении выстрела.

4. **По запросу Снаряда** текущая Ячейка извлекает его из себя.

5. **По запросу Снаряда** соседняя Ячейка помещает его в себя.

6. **Сценарий продолжается с п. 1.1 сценария 5.**

5.4) Альтернативный сценарий «Снаряд пролетает через пустую Ячейку, находящуюся в направлении выстрела». Сценарий выполняется с п. 1.1. сценария 5.

1. **По запросу Снаряда** текущая Ячейка сообщает ему о соседней Ячейке в направлении выстрела.

2. **По запросу Снаряда** текущая Ячейка извлекает его из себя.

3. **По запросу Снаряда** соседняя Ячейка помещает его в себя.

4. **Сценарий продолжается с п. 1.1 сценария 5.**

5.4.1) Альтернативный сценарий «Снаряд достигает края Поля». Сценарий выполняется с п. 2. сценария 5.4.

1. **По запросу Снаряда** текущая Ячейка сообщает о том, что у нее нет соседа в направлении выстрела.

2. **По запросу Снаряда** текущая Ячейка извлекает его из себя.

3. **Сценарий завершается.**

6) **Дополнительный сценарий** «Игра считает победителем выживший Танк».

1. **По запросу Игры** Поле сообщает ей об одном Танке, который на нем расположен.

2. Игра считает Танк победителем, т. к. его противник уничтожен.

3. **Сценарий завершается.**

6.1) **Альтернативный сценарий** «Игра считает победителем выживший Танк с целым Штабом». Сценарий выполняется с п. 1 сценария 6.

1. **По запросу Игры** Поле сообщает Игре о двух Танках, которые на нем расположены.

2. **По запросу Игры** Танки сообщают ей о своих Штабах.

3. Игра узнает об одном Штабе.

4. Игра считает победителем Танк, Штаб которого цел.

5. **Сценарий завершается.**

6.1.1) **Альтернативный сценарий** «Игра завершается вничью т. к. уничтожены оба штаба». Сценарий выполняется с п. 3 сценария 6.1.

1. Игра не узнает ни об одном Штабе.

2. Игра завершается в ничью, т. к. оба Штаба уничтожены.

3. **Сценарий завершается.**

6.2) **Альтернативный сценарий** «Игра завершается вничью т. к. оба танка уничтожены». Сценарий выполняется с п. 1 сценария 6.

1. **По запросу Игры** Поле не сообщает Игре ни об одном Танке.

2. Игра завершается в ничью, т. к. оба танка уничтожены.

3. **Сценарий завершается.**

7) **Альтернативный сценарий** «Досрочное завершение игры».

Сценарий выполняется в любой точке главного сценария

1. **По указанию пользователя** программа завершается без

определения победителя.

2. **Сценарий завершается.**

## 3.3 Словарь предметной области

**Игра –** знает о Поле и активном Танке. Инициирует создание и заполнение Поля. Игра определяет следующего активного игрока и окончание игры.

**Поле** – прямоугольная область, состоящая из Ячеек. Знает обо всех Ячейках, из которых состоит, а также о Танках, которые на нем расположены.

**Окружение** – умеет создавать Танки, Штабы, Препятствия и размещать их в Ячейках на Поле.

**Ячейка –** квадратная часть Поля. Знает о четырех соседних Ячейках. На ней может располагаться Танк, Снаряд, Препятствие или Штаб.

**Снаряд** – принадлежит Танку. Может наносить урон Танку противника, Штабу и Стене, если находится с ними во одной Ячейке.

**Танк –** умеет поворачиваться в одном из четырех направлений и однократно перемещаться или стрелять в выбранном направлении, а также пропускать ход. Знает о Ячейке, в которой находится и о своем Штабе.

**Штаб –** располагается в Ячейке ипривязывается к Танку, также влияет на исход игры**.** Не может находиться с Танком в одной Ячейке.

**Препятствие –** располагается в Ячейке. Не может находиться с Танком в одной Ячейке.

**Стена** – является Препятствием. Может быть уничтожена Снарядом.

**Вода** – является Препятствием. Не может быть уничтожена Снарядом.

## 3.4 Структура программы на уровне классов

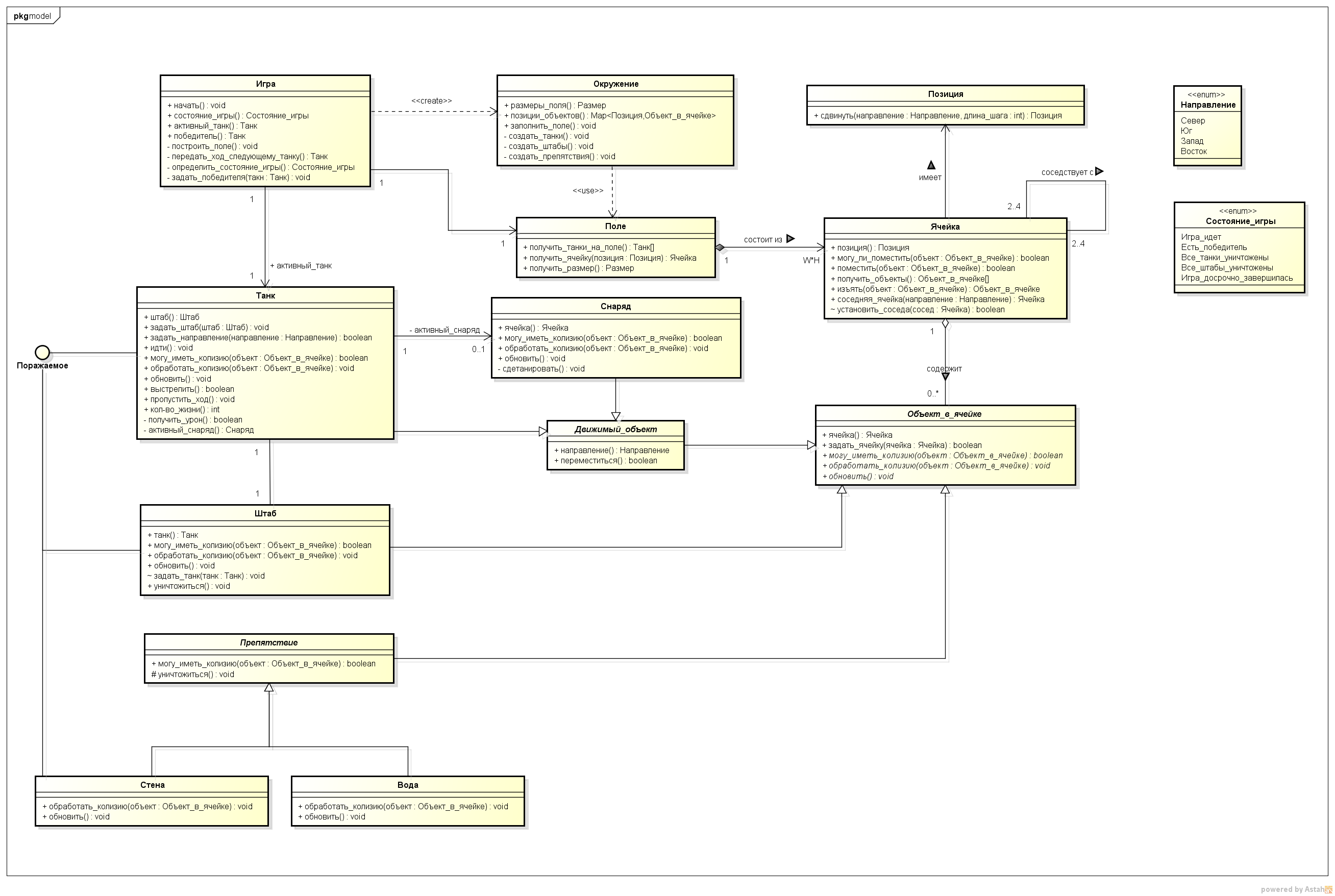


Диаграмма классов вычислительной модели

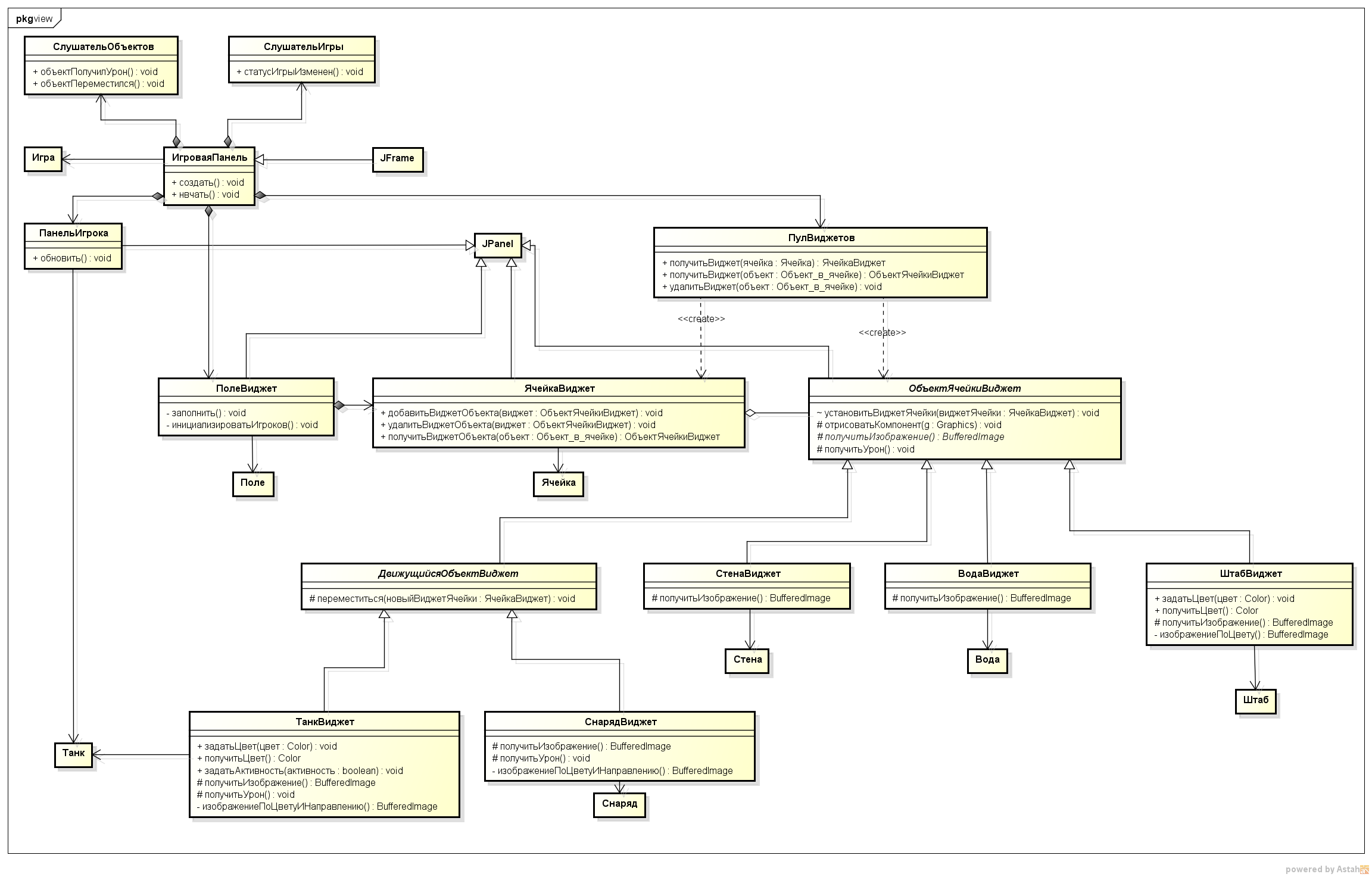
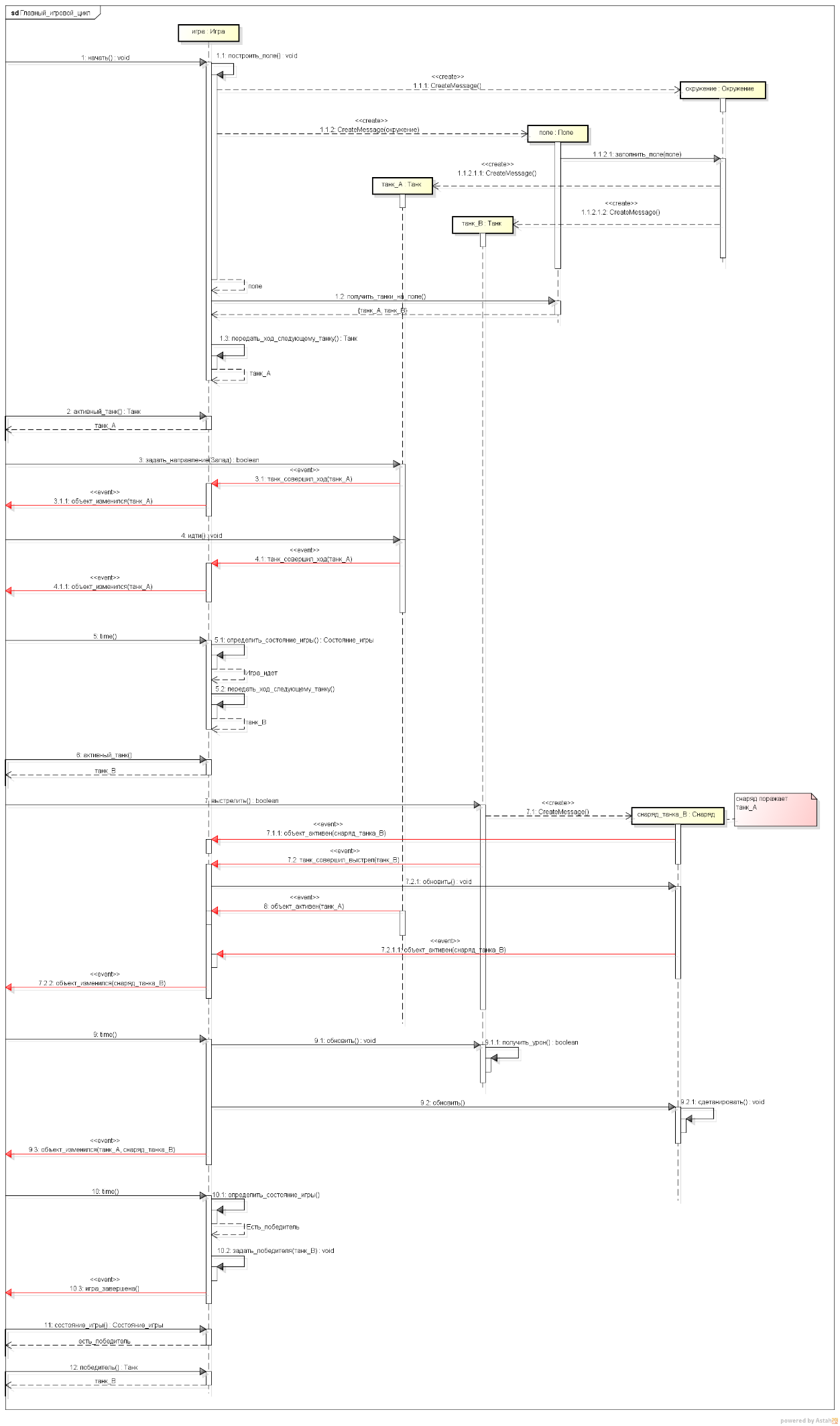
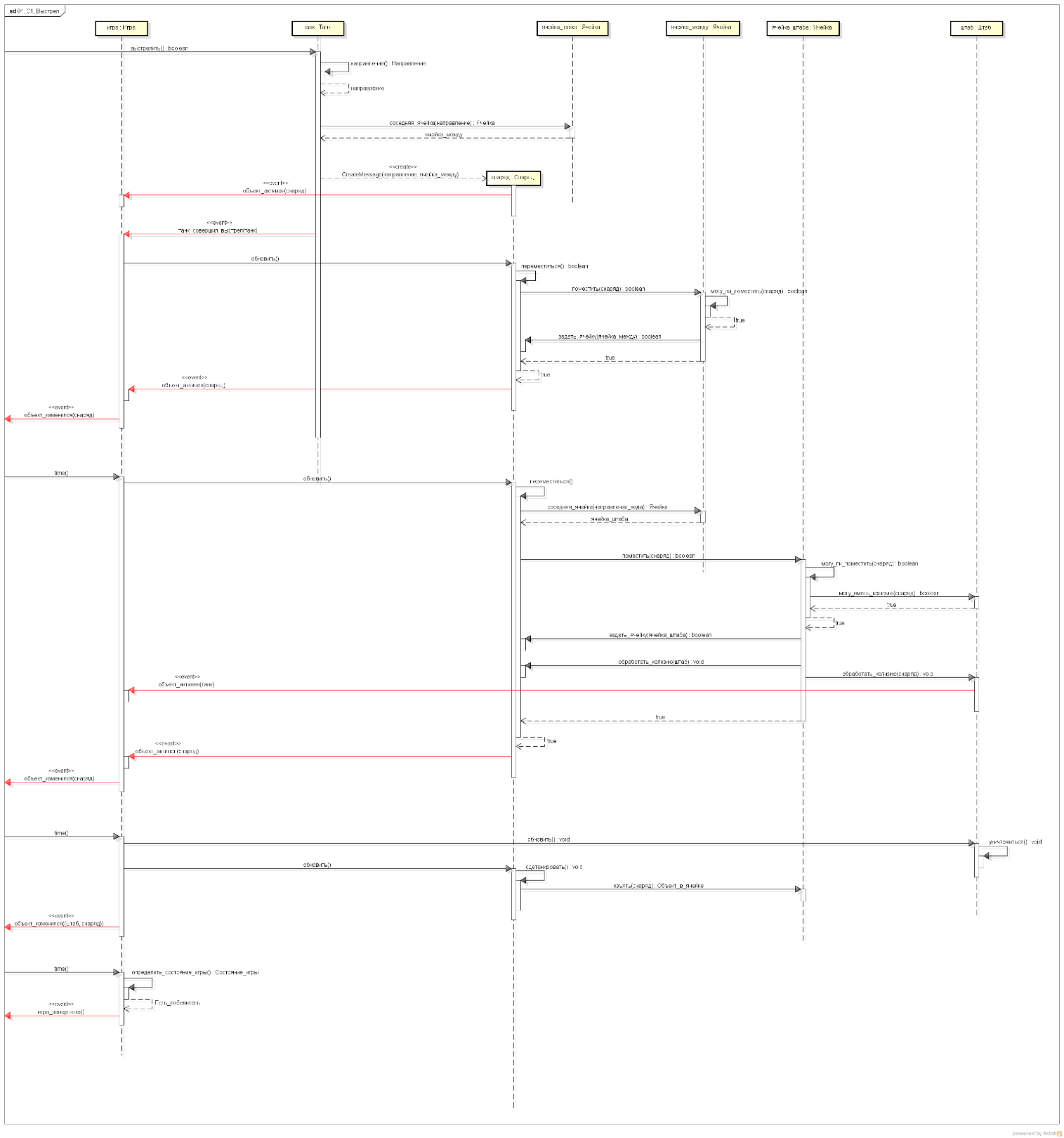


Диаграмма классов представления

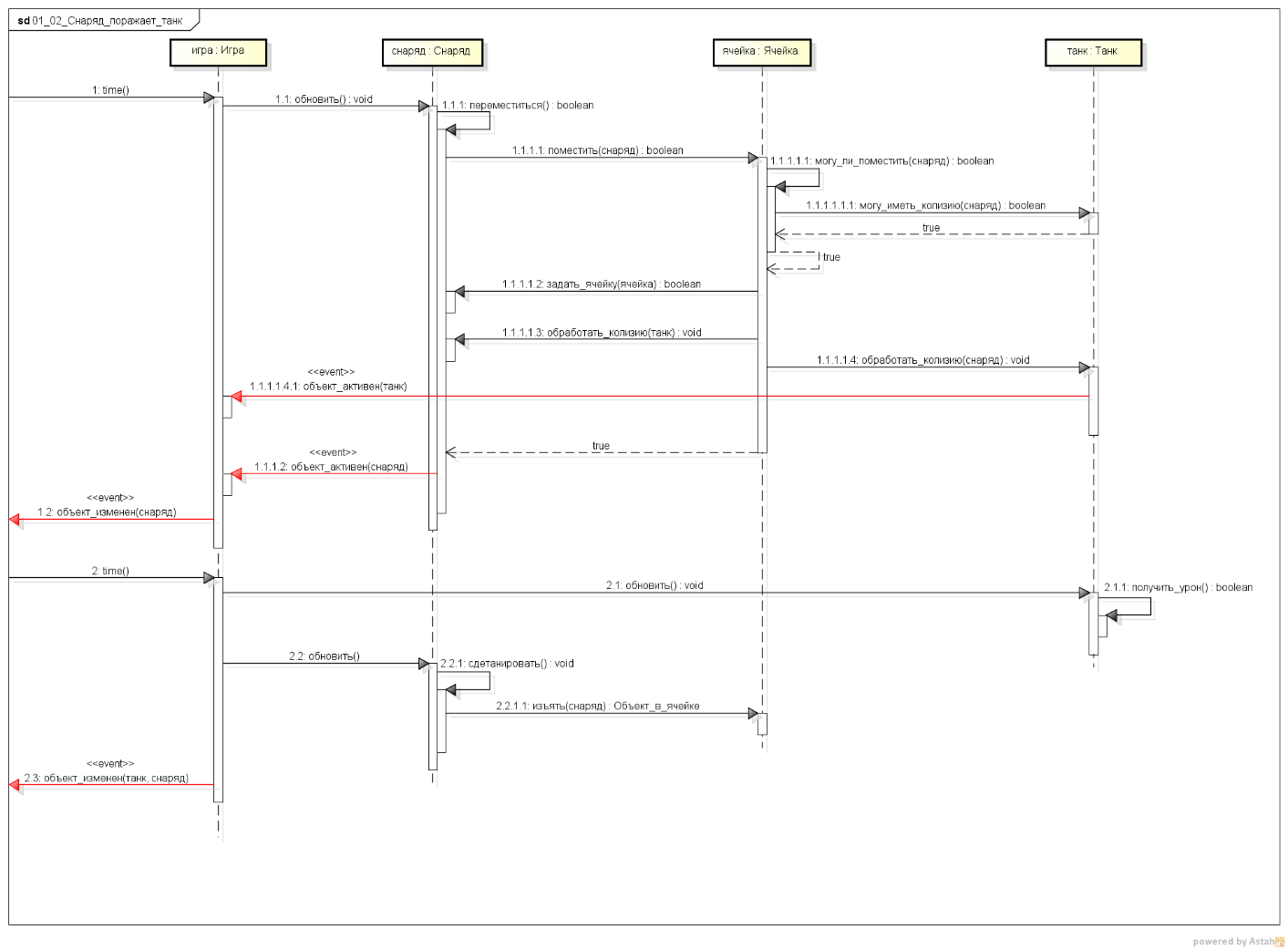
## 3.5 Типовые процессы в программе



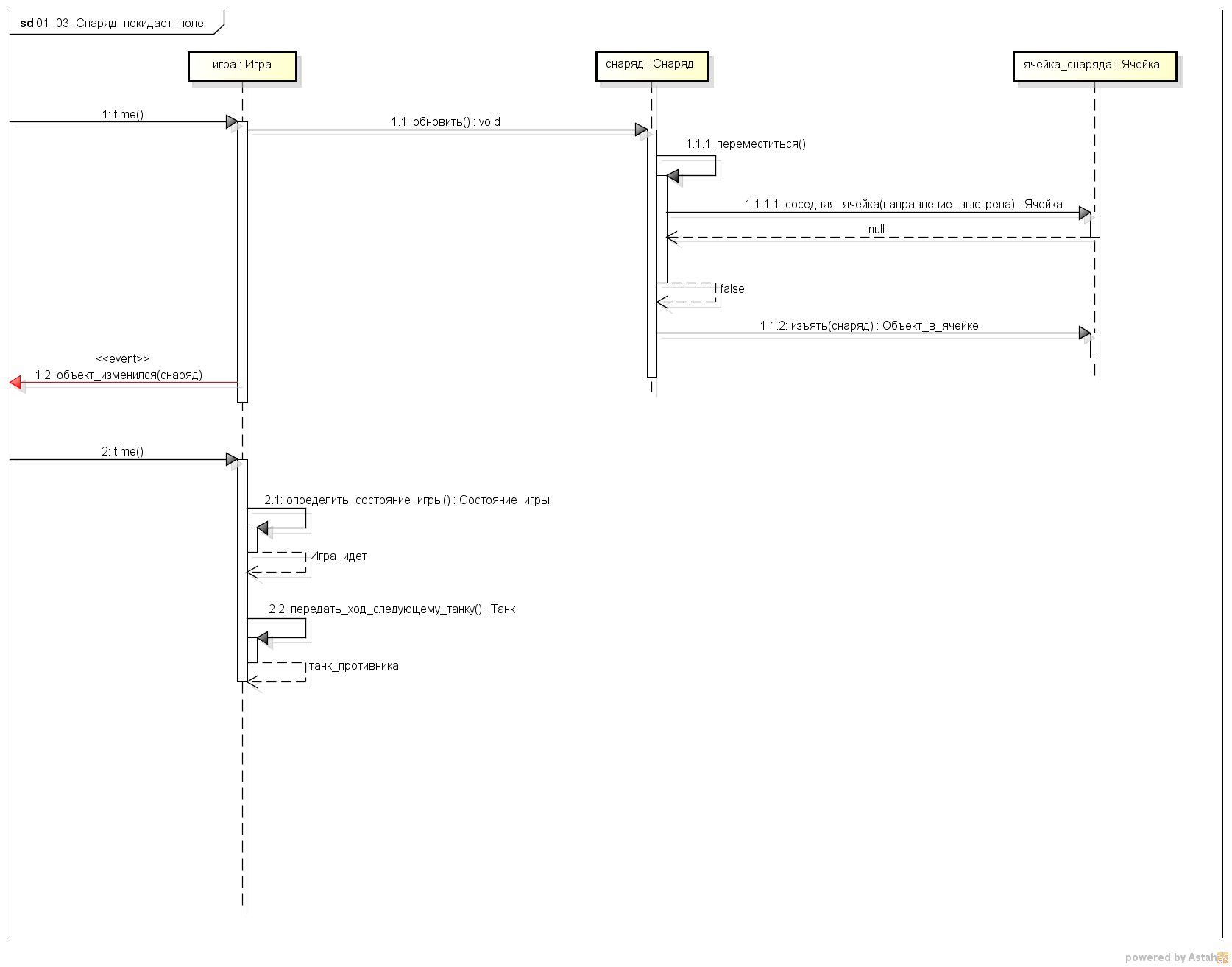
Общий игровой цикл



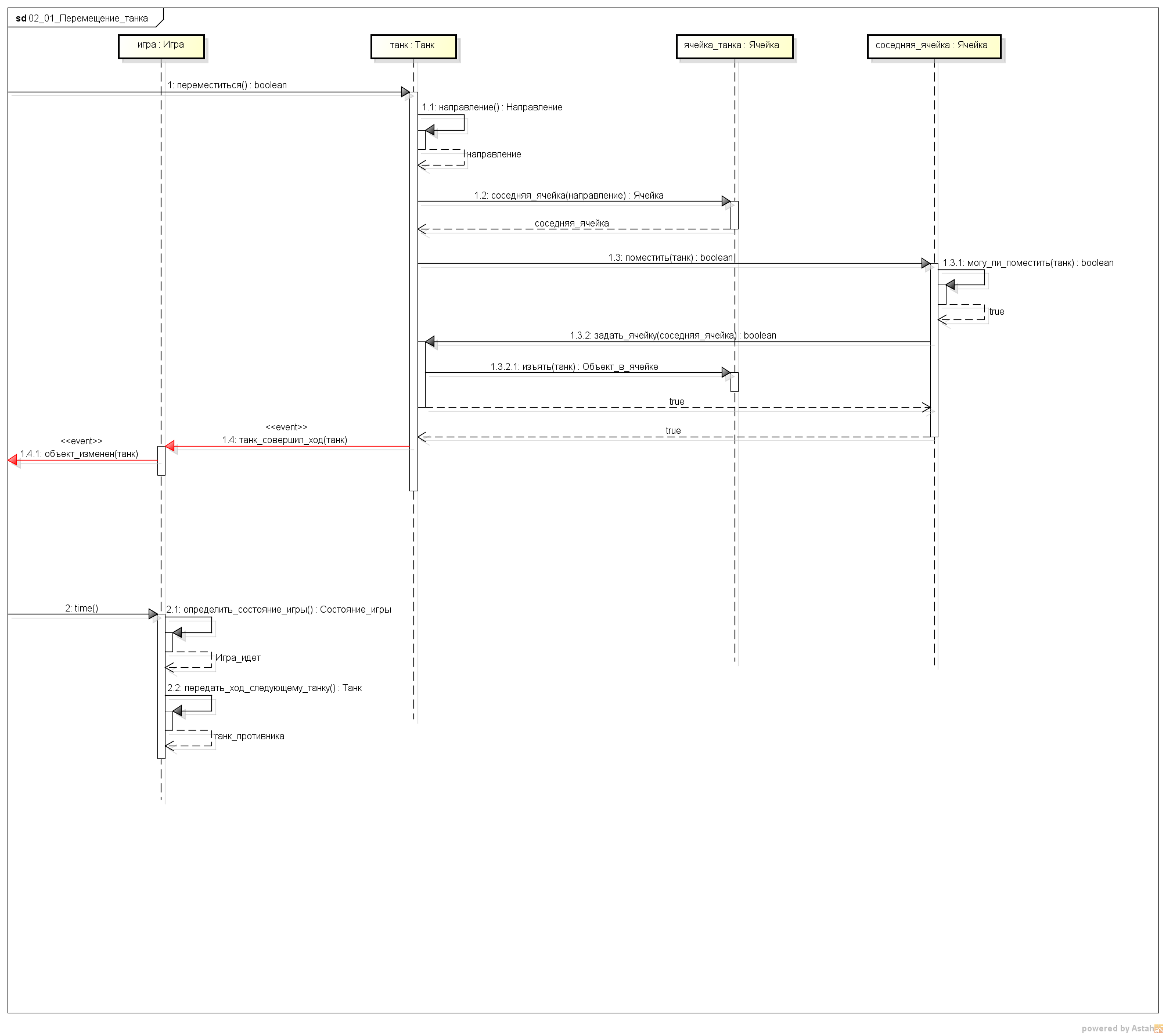
Выстрел



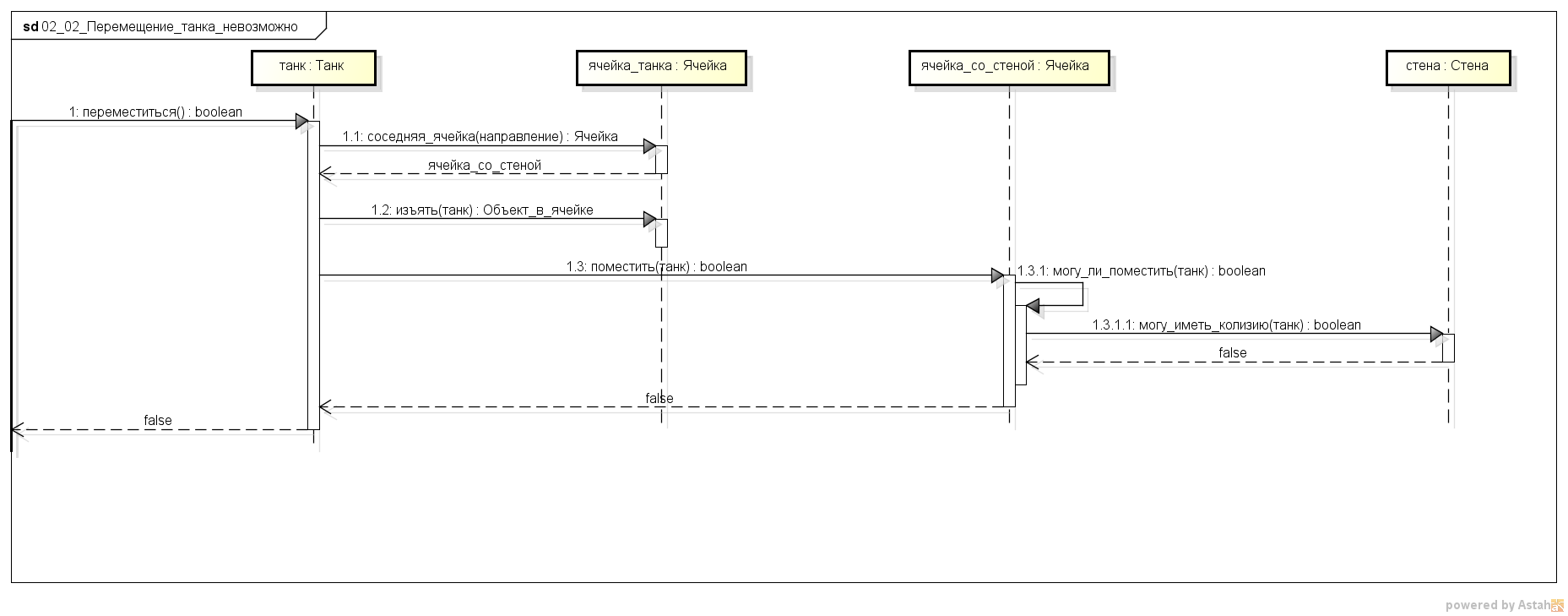
Снаряд поражает снаряд



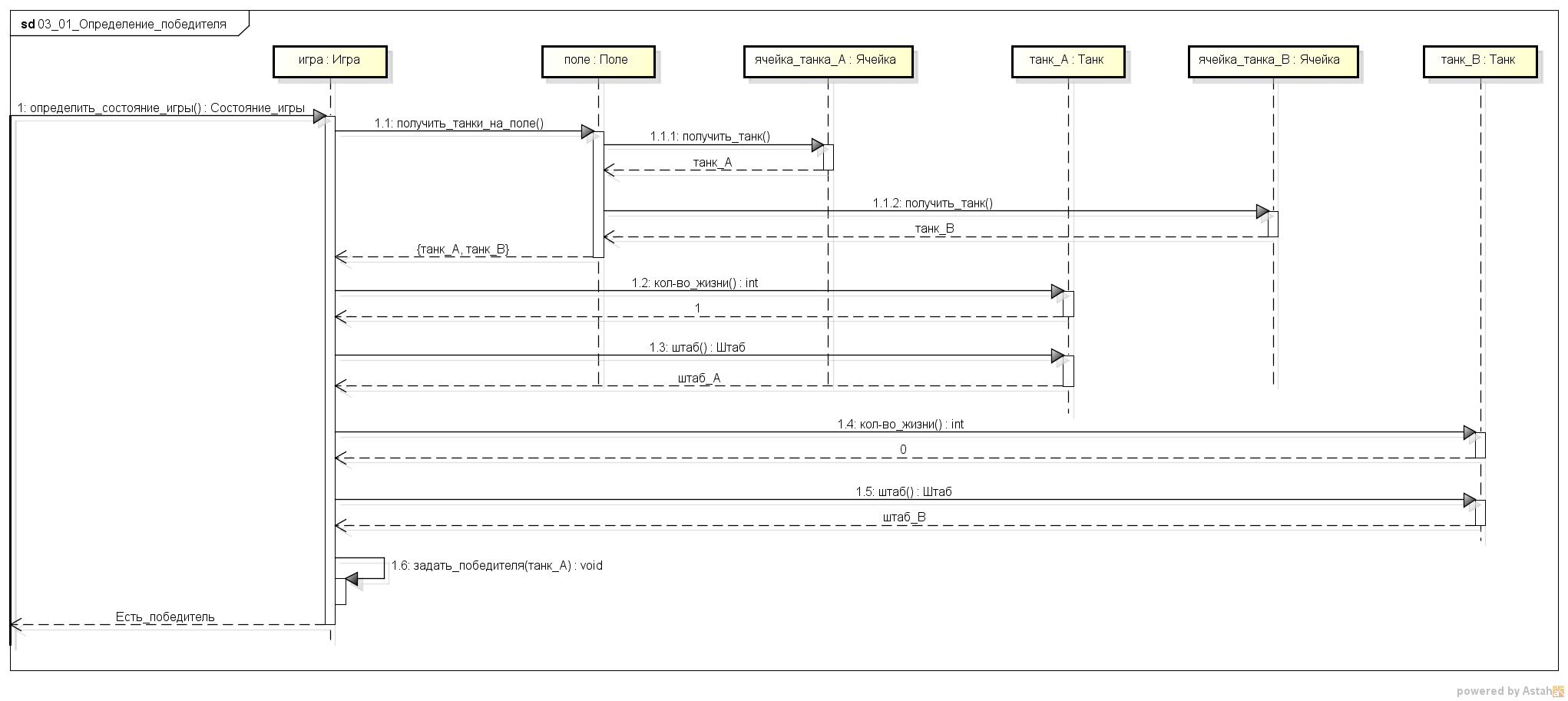
Снаряд покидает поле

****

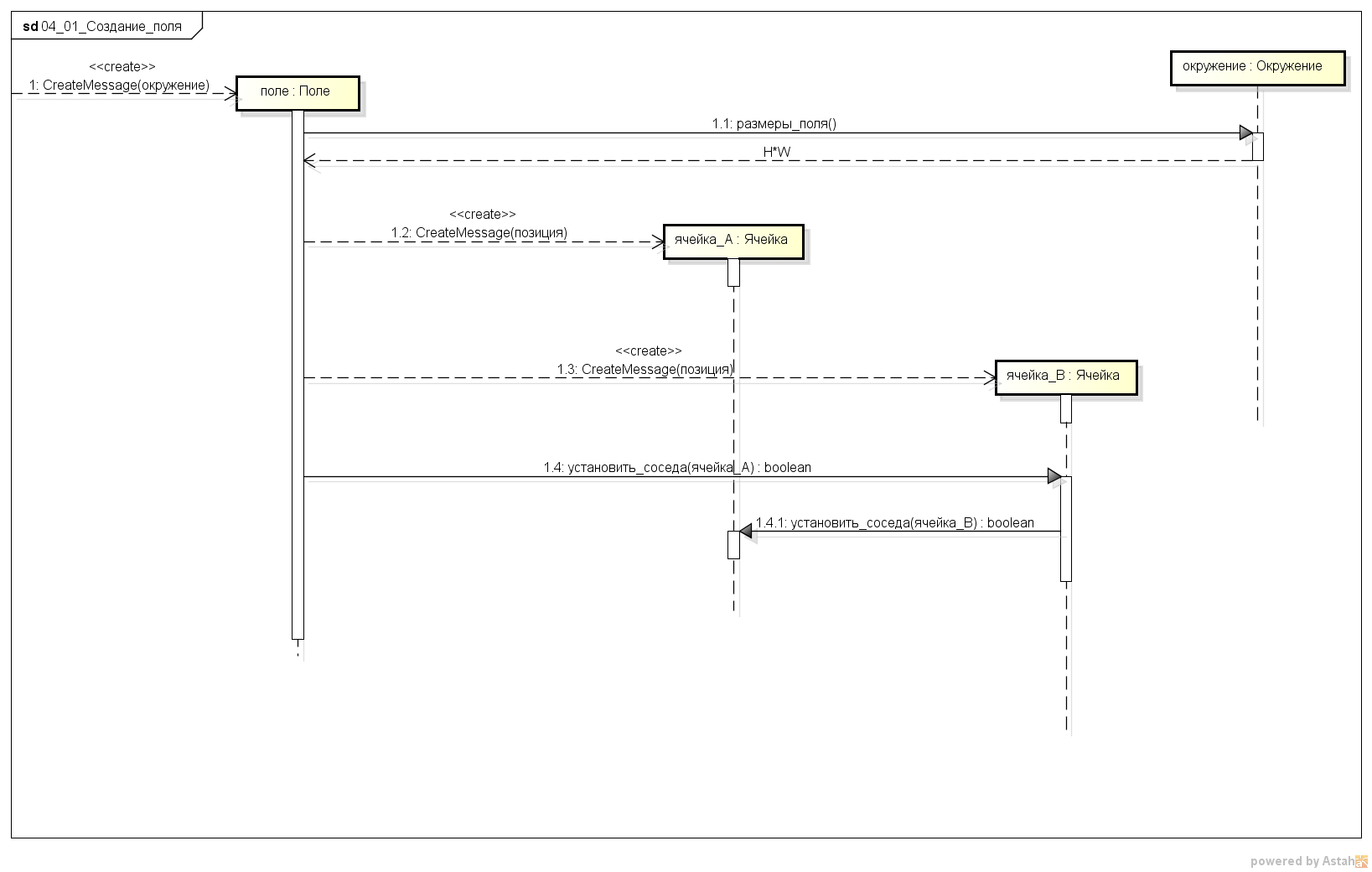
Перемещение танка



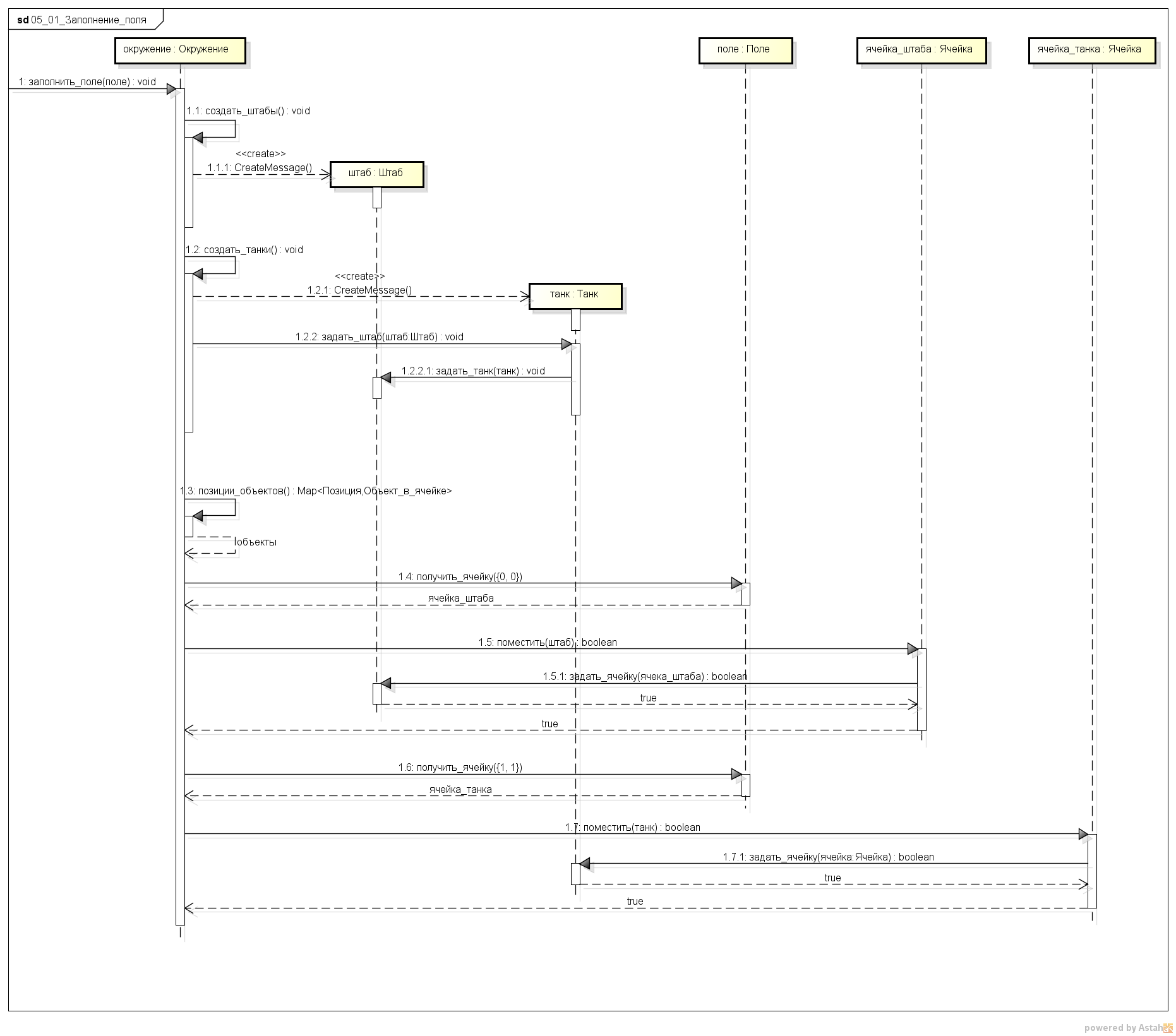
Перемещение танка невозможно



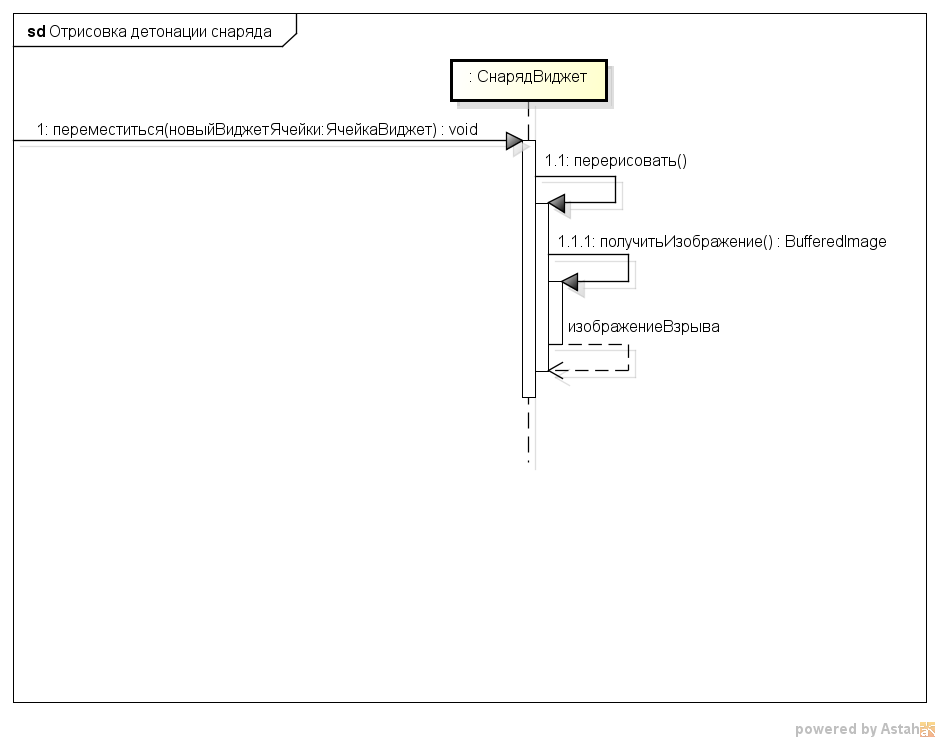
Определение победителя



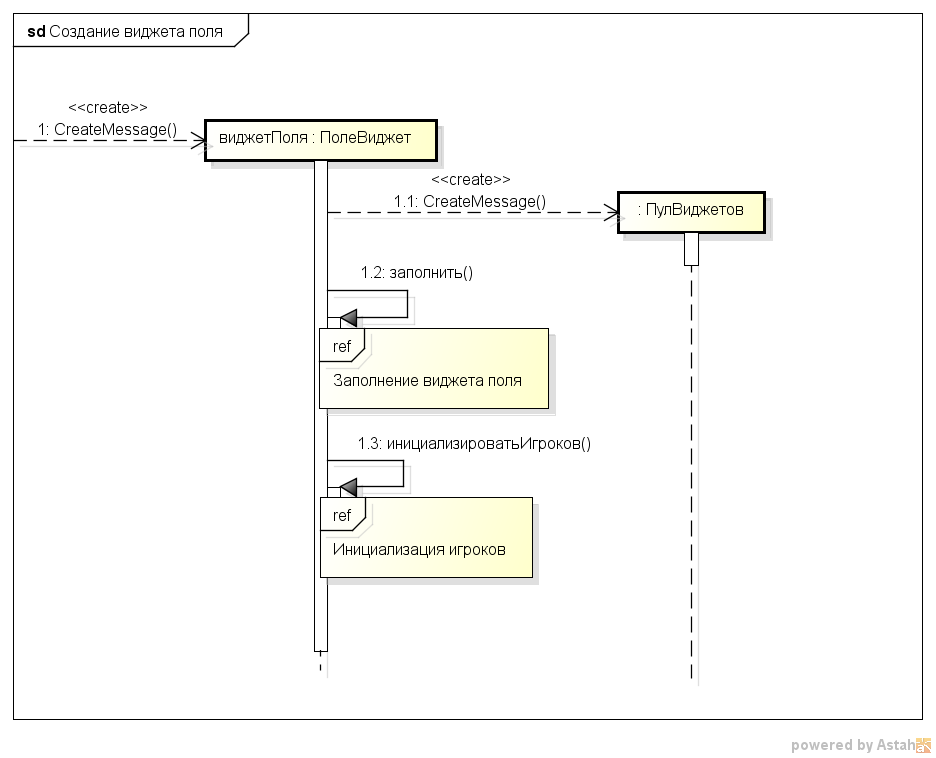
Создание поля



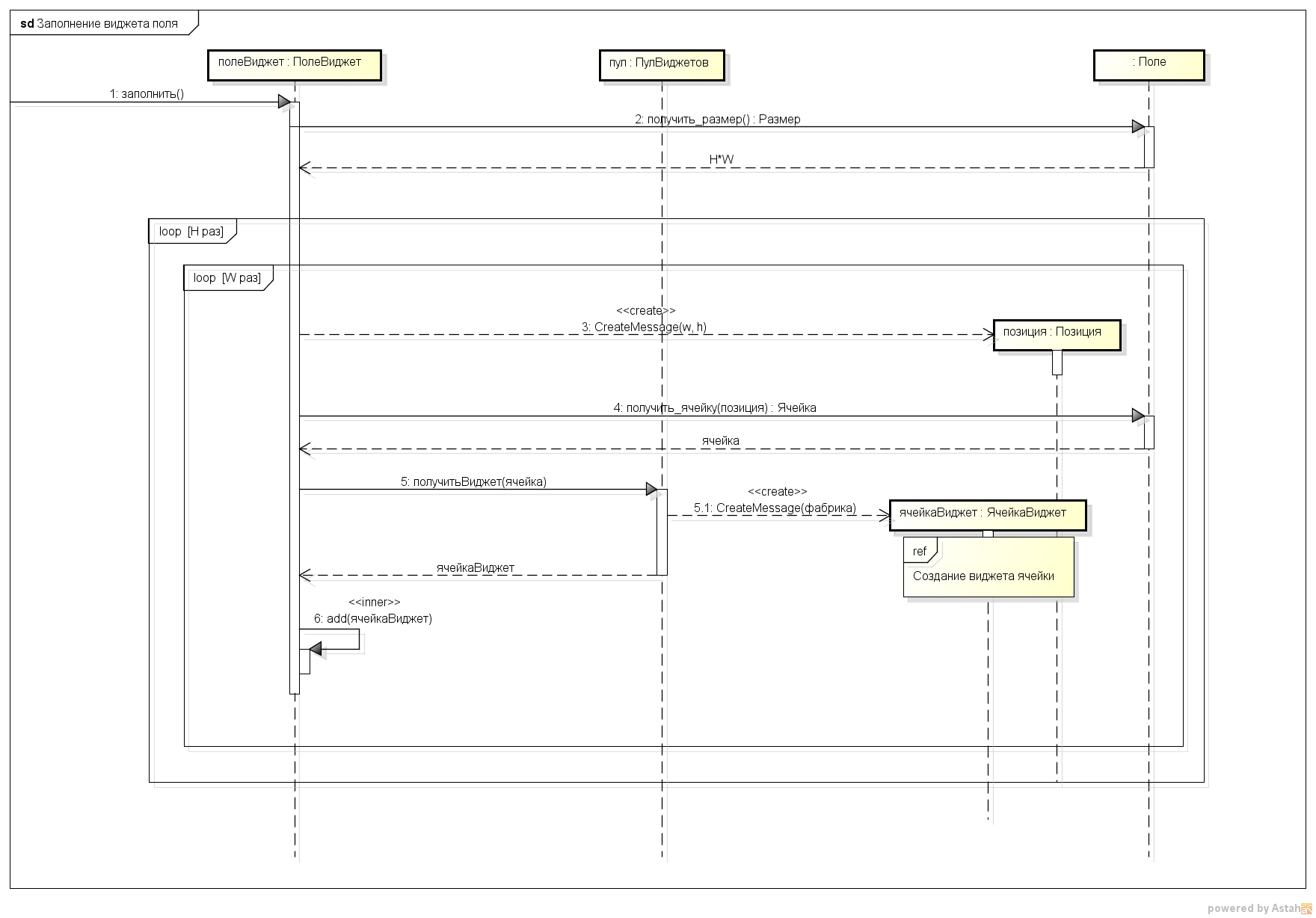
Заполнение поля



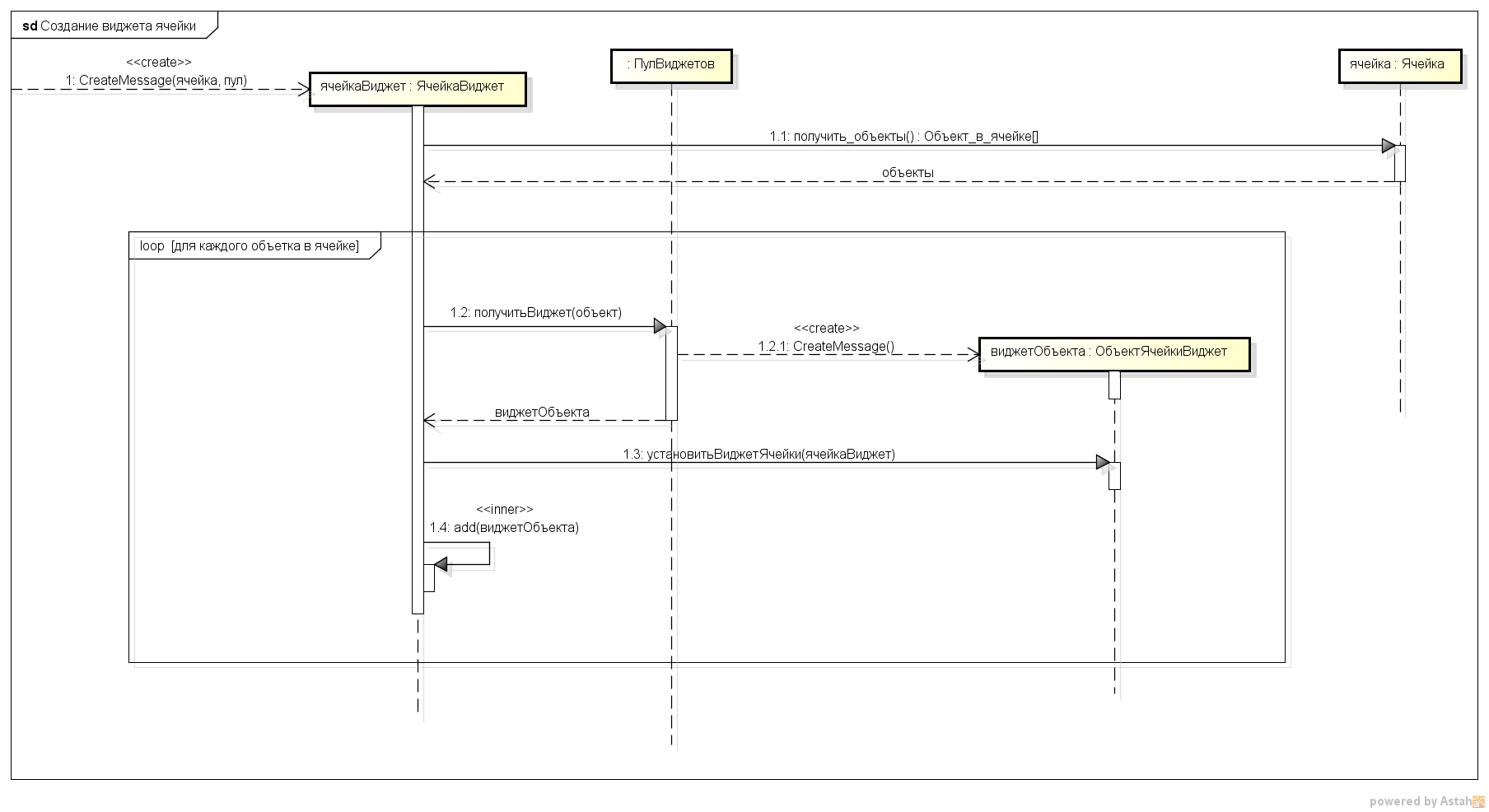
Отрисовка снаряда



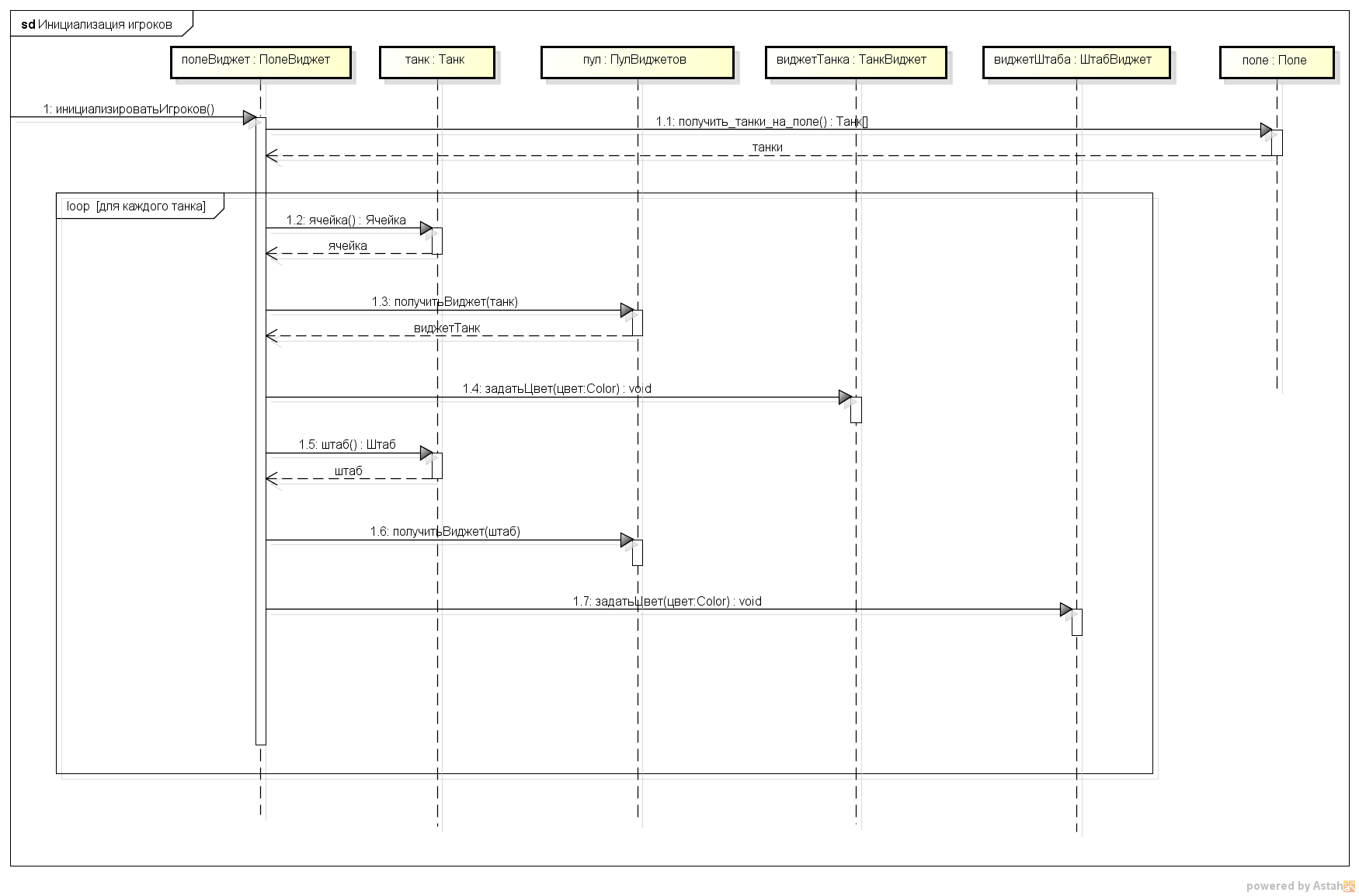
Создание виджета поля



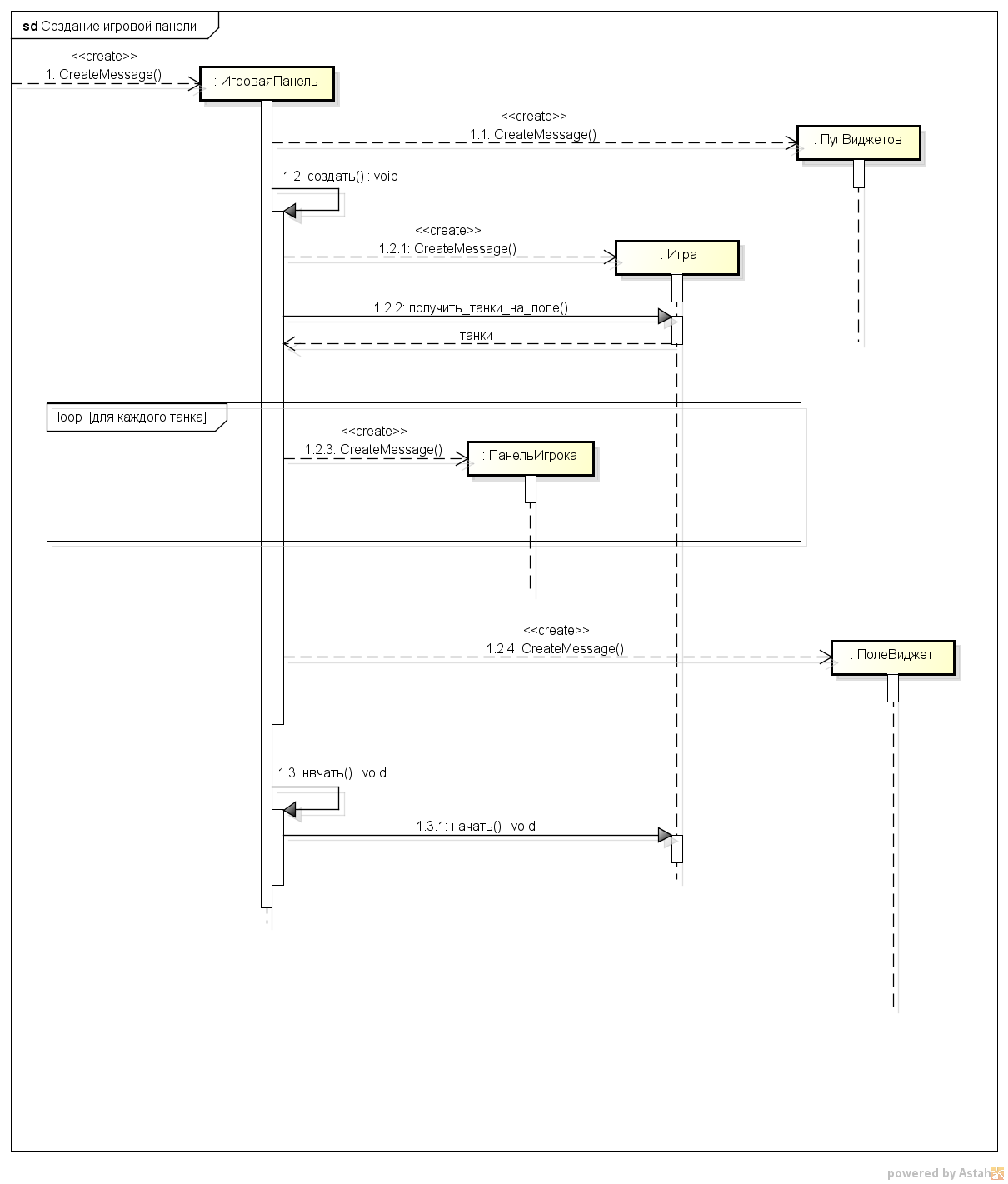
Заполнение виджета поля



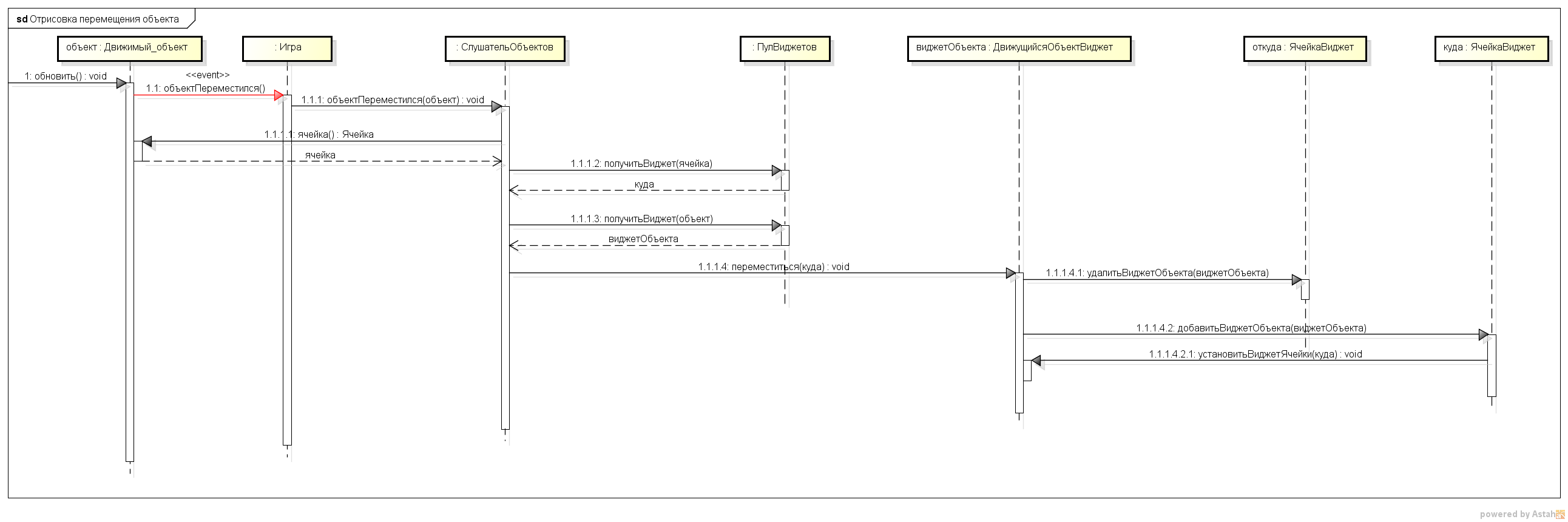
Создание виджета ячейки



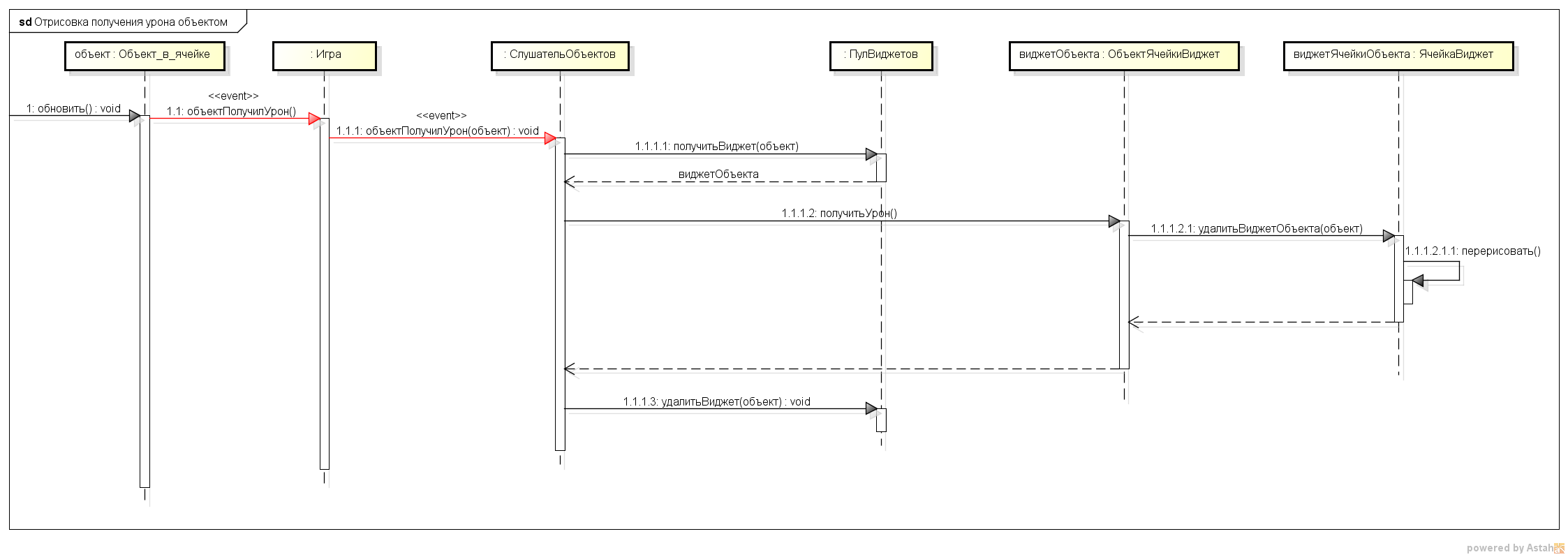
Инициализация игроков



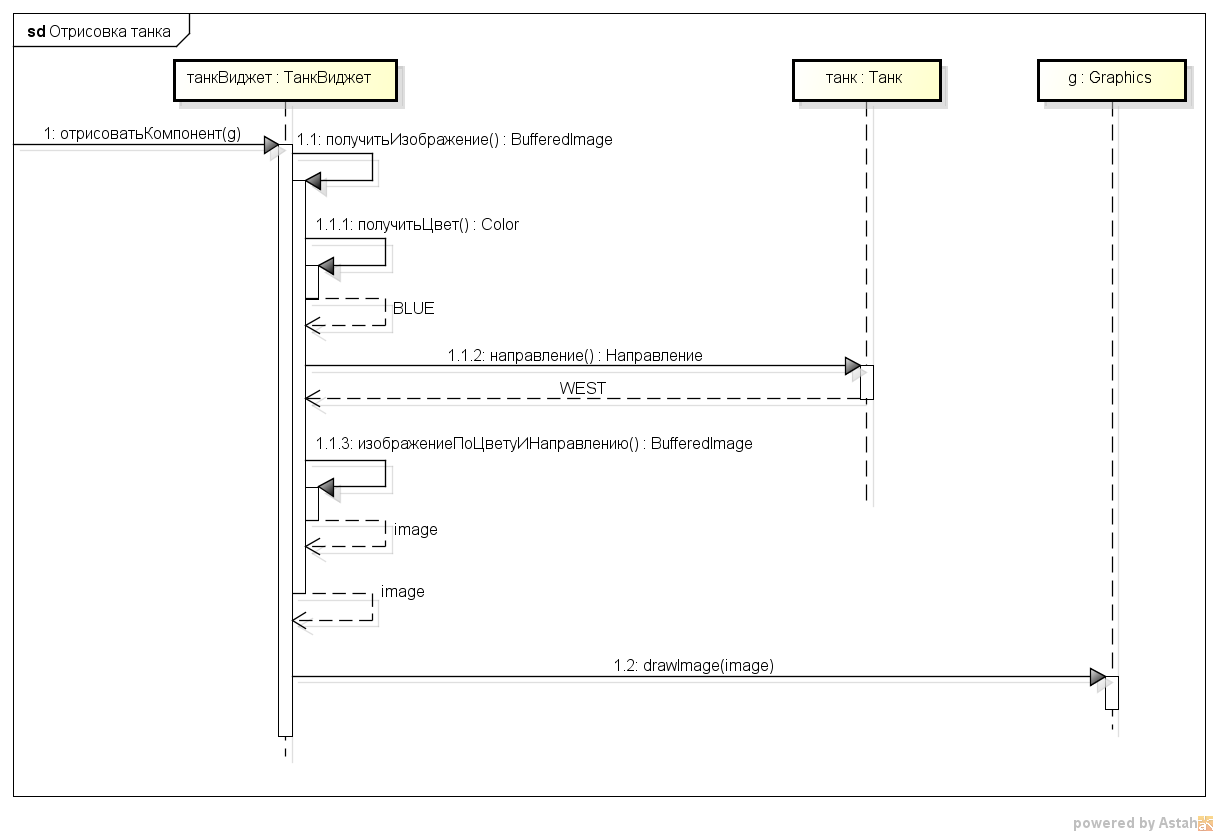
Создание игровой панели



Отрисовка перемещения объекта



Отрисовка получения уроном объекта



Отрисовка танка

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

Отрисовка перехода хода

## 3.6 Человеко-машинное взаимодействие

Общий вид главного экрана игры представлен на рисунке 1. На нем располагается игровое поле, на котором изображены два игрока (танка) синий и оранжевый, штабы танков, препятствия – стены и вода.

Изображение выглядит как текст, седзи, кроссворд, строительство

Автоматически созданное описание

Рис. 1 – Общий вид главного экрана игры

Управление активным танком осуществляется с помощью клавиатуры.

Изменение направления танка:

* W – вверх,
* S – вниз,
* D – вправо,
* A – влево.

“Space” – перемещение.

“Enter” – выстрел.

“Backspace” – пропуск хода.

Над и под игровым полем расположены панели с информацией об игроках, на рисунке 2 представлена панель первого игрока, на которой отображается количество жизни игрока (одно сердце – одна жизнь). Цвет флага показывает танк какого цвета принадлежит игроку.

Изображение выглядит как текст, свет, темный

Автоматически созданное описание

Рис. 2 – Панель с информацией об игроке

В нижней части панели игрока отображается индикатор активности орудия. Если орудие активно, отображается желтый снаряд, иначе – серый, а рядом с ним количество ходов до завершения перезарядки.

Изображение выглядит как логотип

Автоматически созданное описание Изображение выглядит как логотип

Автоматически созданное описание

Рис. 3 – Индикатор активности орудия.

Изображение танка представлено на рисунке 4. Цвет танка игрока совпадает с цветом флага на панели игрока на соответствующей информационной панели. Активность танка показывается зеленой подцветкой ячейки.



Рис. 4 - Танк

У каждого игрока также есть свой штаб, изображение которого представлено на рисунке 5. Цвет игрока совпадает с цветом флага штаба, который ему принадлежит.

Изображение выглядит как диаграмма

Автоматически созданное описание

Рис. 5 – Штаб

Танк при выстреле испускает снаряд, который изображен на рисунке 6.

Изображение выглядит как логотип

Автоматически созданное описание

Рис. 6 – Снаряд

При поражении снарядом другого объекта, на месте снаряда отображается взрыв, так как это показано на рисунке 7.

Изображение выглядит как текст, свет

Автоматически созданное описание 

Рис. 7 – Детонация снаряда.

## 3.7 Реализация ключевых классов

*/\*\*  
 \* Объект, располагающийся в ячейке на поле  
 \*/***public abstract class** ObjectInCell {  
  
 */\* --------------- Ячейка ---------------------- \*/  
  
 /\*\*  
 \* Ячейка в которой находится объект  
 \*/* **private** Cell **\_cell**;  
  
 **public** Cell getCell(){  
 **return \_cell**;  
 }  
  
 **void** setCell(Cell cell){  
 **if** (**\_cell** != **null**){  
 **\_cell**.takeObject(**this**);  
 }  
 **\_cell** = cell;  
 }  
  
 **void** unsetCell(){  
 **\_cell** = **null**;  
 }  
  
 */\* -------------------------- Взаимодействие с другими объектами --------------- \*/  
  
 /\*\*  
 \* Может ли объект находится в одной ячейке с указанным объектом  
 \** ***@param object*** *объект, с котоым проверяется столкновение  
 \** ***@return*** *true если столькновение возможно  
 \*/* **public abstract boolean** canFaceWith(ObjectInCell object);  
  
 */\*\*  
 \* Столкнуть объект с переданным  
 \** ***@param object*** *объект для столкновения  
 \** ***@throws*** *IllegalArgumentException если столкновение невозможно  
 \*/* **public void** faceWith(ObjectInCell object){  
 **if** (!canFaceWith(object)){  
 **throw new** IllegalArgumentException();  
 }  
 }  
  
 */\*\*  
 \* Обновление состояния объекта, вызываемое из игрового цикла  
 \*/* **abstract void** update();  
  
 */\* -------------------- Слушатели и события ------------------- \*/  
  
 /\*\*  
 \* Слушатели игрового объекта  
 \*/* **private final** Set<IObjectInCellEventListener> **\_listeners** = **new** HashSet<>();  
  
 **public void** addListener(IObjectInCellEventListener listener) {  
 **\_listeners**.add(listener);  
 }

*/\*\*  
 \* Сообщить об изменение игрового объекта  
 \** ***@param event*** *событие с информацией об объекте  
 \*/* **protected void** fireEvent(ObjectInCellEvent event){  
 **for** (IObjectInCellEventListener listener : **\_listeners**){  
 listener.onObjectInCellAction(event);  
 }  
 }  
}

*/\*\*  
 \* Перемещающийся объект  
 \*/*

**public abstract class** MovableObject **extends** ObjectInCell{  
  
 */\*\*  
 \* Направление, в котором движется объект  
 \*/* **private** Direction **\_direction**;  
  
 **public** Direction getDirection(){  
 **return \_direction**;  
 }  
  
 **void** setDirection(Direction direction){  
 **\_direction** = direction;  
 }  
  
 */\*\*  
 \* Перемещение объекта в соседнюю ячейку в текущем направлении  
 \** ***@return*** *true если побъект перемещен в новую ячейку  
 \*/* **public boolean** move(){  
 Cell nextCell = getCell().getNeighbour(getDirection());  
 **if** (nextCell == **null**){  
 **return false**;  
 }  
  
 **return** nextCell.addObject(**this**);  
 }  
}

*/\*\*  
 \* Класс танка  
 \*/*

**public class** Tank **extends** MovableObject **implements** Damageable {  
  
 **public** Tank(){  
 setDirection(Direction.***NORTH***);  
 }  
  
 */\* --------------------- Состояние танка ------------------ \*/  
  
 /\*\*  
 \* Интерактивность танка, true если танком можно управлять  
 \*/* **private boolean \_isActive** = **true**;  
  
 **public boolean** isActive(){  
 **return \_isActive**;  
 }  
  
 **void** setActive(**boolean** active){  
 **\_isActive** = active;  
 }  
  
 */\*\*  
 \* Состояние танка, true если танк получает урон  
 \*/* **private boolean \_isDamaged** = **false**;  
  
 */\* -------------------------- Штаб ----------------- \*/  
  
 /\*\*  
 \* Штаб, к которому относиться танк  
 \*/* **private** Headquarters **\_headquarters**;  
  
 **public** Headquarters getHeadquarters(){  
 **return \_headquarters**;  
 }  
  
 **public void** setHeadquarters(Headquarters headquarters){  
 **if** (headquarters == **\_headquarters**){  
 **return**;  
 }  
  
 **\_headquarters** = headquarters;  
 headquarters.setTank(**this**);  
 }  
  
 */\* ----------------- Управление танком ------------ \*/  
  
 /\*\*  
 \* Поворот танка в указанном направлении  
 \** ***@param direction*** *направление поворота  
 \*/* **public void** rotate(Direction direction){  
 **if** (!isActive()){  
 **return**;  
 }  
  
 **if** (direction == getDirection()){  
 **return**;  
 }  
  
 setDirection(direction);  
 fireEvent(**new** ObjectInCellEvent(**this**, ObjectInCellEvent.EventType.***MOVING***));  
 }  
  
 */\*\*  
 \* Время перезарядки орудия  
 \*/* **private final int RELOAD\_TIME** = 3;  
  
 */\*\*  
 \* Текущее время перезарядки  
 \*/* **private int \_currentReloadTime** = 0;  
  
 **public int** getCurrentReloadTime() { **return \_currentReloadTime**; }  
  
 */\*\*  
 \* Соверщить выстрел  
 \** ***@return*** *true если выстрел совершен успешно  
 \*/* **public boolean** shoot(){  
 **if** (!isActive() || getCurrentReloadTime() > 0){  
 **return false**;  
 }  
  
 Cell targetCell = getCell().getNeighbour(getDirection());  
 **if** (targetCell == **null**){  
 **return false**;  
 }  
  
 Bullet bullet = **new** Bullet(getDirection(), getCell());  
 bullet.addListener(**new** BulletListener());  
 fireShot(**new** ObjectInCellEvent(bullet, ObjectInCellEvent.EventType.***MOVING***));  
 **\_currentReloadTime** = **RELOAD\_TIME**;  
 **return true**;  
 }  
  
 */\*\*  
 \* Пропустить ход  
 \*/* **public void** pass(){  
 **if** (!isActive()){  
 **return**;  
 }  
  
 **\_currentReloadTime** -= 1;  
 fireSkip(**new** ObjectInCellEvent(**this**, ObjectInCellEvent.EventType.***MOVING***));  
 }  
  
 @Override  
 **public boolean** move() {  
 **if** (!isActive()){  
 **return false**;  
 }  
  
 **boolean** isMoved = **super**.move();  
 **if** (isMoved){  
 **\_currentReloadTime** -= 1;  
 fireMoved(**new** ObjectInCellEvent(**this**, ObjectInCellEvent.EventType.***MOVING***));  
 }  
 **return** isMoved;  
 }  
  
 */\* -------------------- Взаимодействие с другими объектами ------------------ \*/  
  
 /\*\*  
 \* Здоровье танка  
 \*/* **private int \_health** = 3;  
  
 **public int** getHealth(){  
 **return \_health**;  
 }  
  
 @Override  
 **public boolean** canFaceWith(ObjectInCell object) {  
 **return** object **instanceof** Bullet;  
 }  
  
 @Override  
 **public void** faceWith(ObjectInCell object) {  
 **super**.faceWith(object);  
  
 **if** (object **instanceof** Bullet){  
 **\_isDamaged** = **true**;  
 fireEvent(**new** ObjectInCellEvent(**this**, ObjectInCellEvent.EventType.***DAMAGED***));  
 }  
 }  
  
 @Override  
 **void** update() {  
 **if** (**\_isDamaged** && getHealth() > 0){  
 **\_health** = getHealth() - 1;  
 **\_isDamaged** = **false**;  
 }  
 }

*/\* -------------------------- Слушатели и события ------------------- \*/  
  
 /\*\*  
 \* Слушатели танка  
 \*/* **private final** Set<ITankEventListener> **\_listeners** = **new** HashSet<>();  
  
 @Override  
 **public void** addListener(IObjectInCellEventListener listener){  
 **if** (!(listener **instanceof** ITankEventListener tankListener)){  
 **return**;  
 }  
  
 **\_listeners**.add(tankListener);  
 }  
  
 @Override  
 **protected void** fireEvent(ObjectInCellEvent event) {  
 **for** (ITankEventListener listener : **\_listeners**){  
 listener.onObjectInCellAction(event);  
 }  
 }  
  
 */\*\*  
 \* Сообщить об успешном перемещении танка  
 \** ***@param event*** *событие с информацией о танке  
 \*/* **private void** fireMoved(ObjectInCellEvent event){  
 **for** (ITankEventListener listener : **\_listeners**){  
 listener.onTankMoved(event);  
 }  
 }  
  
 */\*\*  
 \* Сообщить об успешном выстреле  
 \** ***@param event*** *событие с информацией о выпущенном снаряде  
 \*/* **private void** fireShot(ObjectInCellEvent event){  
 **for** (ITankEventListener listener : **\_listeners**){  
 listener.onTankShot(event);  
 }  
 }  
  
 */\*\*  
 \* Сообщение о пропуске хода  
 \** ***@param event*** *событие с информацией о танке  
 \*/* **private void** fireSkip(ObjectInCellEvent event){  
 **for** (ITankEventListener listener : **\_listeners**){  
 listener.onTankSkipStep(event);  
 }  
 }  
  
 */\*\*  
 \* Слушатель снаряда  
 \*/* **private class** BulletListener **implements** IObjectInCellEventListener{  
  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 fireEvent(event);  
 }  
 }  
}

*/\*\*  
 \* Снаряд выпускаемый танком  
 \*/***public class** Bullet **extends** MovableObject{  
  
 */\*\*  
 \* Сосотояние снаряда, true - если мнаряд уничтожен  
 \*/* **private boolean \_isDestroying** = **false**;  
  
 **public boolean** isDetonating(){  
 **return \_isDestroying**;  
 }  
  
 **public** Bullet(Direction direction, Cell startCell){  
 setDirection(direction);  
 setCell(startCell);  
 }  
  
 @Override  
 **public boolean** canFaceWith(ObjectInCell object) {  
 **return** !(**this** == object);  
 }  
  
 @Override  
 **public void** faceWith(ObjectInCell object) {  
 **super**.faceWith(object);  
  
 **if** (object **instanceof** Damageable){  
 **\_isDestroying** = **true**;  
 fireEvent(**new** ObjectInCellEvent(**this**, ObjectInCellEvent.EventType.***DESTROYING***));  
 }  
 }  
  
 @Override  
 **void** update() {  
 **if** (**\_isDestroying**){  
 getCell().takeObject(**this**);  
 } **else** {  
 move();  
 }  
 }  
  
 @Override  
 **public boolean** move(){  
 **boolean** isMoved = **super**.move();  
 **if** (!**\_isDestroying** && isMoved){  
 fireEvent(**new** ObjectInCellEvent(**this**, ObjectInCellEvent.EventType.***MOVING***));  
 } **else if** (!isMoved){  
 getCell().takeObject(**this**);  
 }  
 **return** isMoved;  
 }  
}

*/\*\*  
 \* Штаб танка  
 \*/***public class** Headquarters **extends** ObjectInCell **implements** Damageable {  
  
 */\*\*  
 \* Состояние штабы, true если уничтожен  
 \*/* **private boolean \_isDestroying** = **false**;

*/\* ------------------ Танк -------------- \*/  
  
 /\*\*  
 \* Танк, относящийся с штабу  
 \*/* **private** Tank **\_tank**;  
  
 **public** Tank getTank(){  
 **return \_tank**;  
 }  
  
 **public void** setTank(Tank tank){  
 **if** (tank == **\_tank**){  
 **return**;  
 }  
  
 **\_tank** = tank;  
 tank.setHeadquarters(**this**);  
 }  
  
 @Override  
 **public boolean** canFaceWith(ObjectInCell object) {  
 **return** object **instanceof** Bullet;  
 }  
  
 @Override  
 **public void** faceWith(ObjectInCell object) {  
 **super**.faceWith(object);  
  
 **if** (object **instanceof** Bullet){  
 **\_isDestroying** = **true**;  
 fireEvent(**new** ObjectInCellEvent(**this**, ObjectInCellEvent.EventType.***DESTROYING***));  
 }  
 }  
  
 @Override  
 **void** update() {  
 **if** (**\_isDestroying**){  
 getCell().takeObject(**this**);  
 }  
 }  
}

*/\*\*  
 \* Препятствия, которые мешают танку  
 \*/***public abstract class** Obstacle **extends** ObjectInCell {  
 @Override  
 **public boolean** canFaceWith(ObjectInCell object){  
 **return** object **instanceof** Bullet;  
 }  
}

*/\*\*  
 \* Стена, мешающая танку и уничтожаемая снарядом  
 \*/***public class** Wall **extends** Obstacle **implements** Damageable{  
  
 */\*\*  
 \* Состояние стены true если стена уничтожена  
 \*/* **private boolean \_isDestroyed** = **false**;  
  
 @Override  
 **public void** faceWith(ObjectInCell object) {  
 **super**.faceWith(object);  
  
 **if** (object **instanceof** Bullet){  
 **\_isDestroyed** = **true**;  
 fireEvent(**new** ObjectInCellEvent(**this**, ObjectInCellEvent.EventType.***DESTROYING***));  
 }  
 }  
  
 @Override  
 **void** update() {  
 **if** (**\_isDestroyed**){  
 getCell().takeObject(**this**);  
 }  
 }  
}

*/\*\*  
 \* Вода, мешающая танку и неуязвима для снаряла  
 \*/***public class** Water **extends** Obstacle{  
 @Override  
 **void** update() {  
  
 }  
}

*/\*\*  
 \* Размеры игрового поля  
 \*/***public class** Size {  
  
 **public** Size(**int** width, **int** height){  
 **\_width** = width;  
 **\_height** = height;  
 }  
  
 **private final int \_width**;  
  
 **public int** width(){  
 **return \_width**;  
 }  
  
 **private final int \_height**;  
  
 **public int** height(){  
 **return \_height**;  
 }  
}

*/\*\*  
 \* Позиция ячейки на поле  
 \*/***public class** Position {  
  
 **private final int \_x**;  
  
 **public int** getX() {  
 **return \_x**;  
 }  
  
 **private final int \_y**;  
  
 **public int** getY(){  
 **return \_y**;  
 }

**public** Position(**int** x, **int** y){  
 **\_x** = x;  
 **\_y** = y;  
 }  
  
 */\*\*  
 \* Получить новую позицию, путем смещения данной  
 \** ***@param direction*** *направление смещения  
 \** ***@param delta*** *растояние смещения  
 \** ***@return*** *новая позиция  
 \*/* **public** Position shift(Direction direction, **int** delta){  
 **int** xDirection = 0;  
 **int** yDirection = 0;  
  
 **switch** (direction){  
 **case *NORTH*** -> yDirection = -1;  
 **case *SOUTH*** -> yDirection = 1;  
 **case *WEST*** -> xDirection = -1;  
 **case *EAST*** -> xDirection = 1;  
 }  
  
 **int** newX = getX() + xDirection \* delta;  
 **int** newY = getY() + yDirection \* delta;  
  
 **return new** Position(newX, newY);  
 }  
  
 @Override  
 **public boolean** equals(Object obj) {  
 **if** (!(obj **instanceof** Position other)){  
 **return false**;  
 }  
  
 **return this**.getX() == other.getX() && **this**.getY() == other.getY();  
 }  
}

*/\*\*  
 \* Направление перемещения движущихся объектов  
 \*/***public enum** Direction {  
  
 ***NORTH***,  
 ***SOUTH***,  
 ***EAST***,  
 ***WEST***}

*/\*\*  
 \* Фрагмент поля, который может содержать в себе объекты  
 \*/***public class** Cell {  
  
 */\*\*  
 \* Позиция ячейки  
 \*/* **private final** Position **\_position**;  
  
 **public** Position getPosition(){  
 **return \_position**;  
 }  
  
 **public** Cell(Position position){  
 **\_position** = position;  
 }  
  
 */\* -------------- Объекты в ячейке ---------------- \*/  
  
 /\*\*  
 \* Множество объектов, содержащихся в ячейке  
 \*/* **private final** Set<ObjectInCell> **\_objects** = **new** HashSet<>();  
  
 */\*\*  
 \* Добавить в ячейку новый объект  
 \** ***@param newObject*** *новый объект  
 \** ***@return*** *true если объект успешно добавлен  
 \*/* **public boolean** addObject(ObjectInCell newObject){  
 **if** (!canAdd(newObject)){  
 **return false**;  
 }  
  
 **\_objects**.add(newObject);  
 newObject.setCell(**this**);  
 faceObjectsWith(newObject);  
  
 **return true**;  
 }  
  
 */\*\*  
 \* Может ли ячейка добавить заданый объект  
 \** ***@param newObject*** *объект  
 \** ***@return*** *true если объект может быть добавлен  
 \*/* **private boolean** canAdd(ObjectInCell newObject){  
 **return \_objects**.stream().allMatch(object -> object.canFaceWith(newObject));  
 }  
  
 */\*\*  
 \* Столкнуть новый объект с объектами уже находящимися в ячейке  
 \** ***@param newObject*** *новый объект  
 \*/* **private void** faceObjectsWith(ObjectInCell newObject){  
 **\_objects**.forEach(object -> {  
 **if** (object != newObject){  
 object.faceWith(newObject);  
 newObject.faceWith(object);  
 }  
 });  
 }  
  
 **public** List<ObjectInCell> getObjects(){  
 **return \_objects**.stream().toList();  
 }  
  
 */\*\*  
 \* Извлечь указанный объект из ячейке  
 \** ***@param object*** *объект, подлежащий извлечению  
 \** ***@return*** *извлеченный объект, null если объекта в ячейке нет  
 \*/* **public** ObjectInCell takeObject(ObjectInCell object){  
 ObjectInCell removedObject = **null**;  
  
 **boolean** removed = **\_objects**.remove(object);  
 **if** (removed){  
 object.unsetCell();  
 removedObject = object;  
 }  
  
 **return** removedObject;  
 }

*/\* ------------------ Соседи ---------------- \*/  
  
 /\*\*  
 \* Соседи ячейки  
 \*/* **private final** Set<Cell> **\_neighbours** = **new** HashSet<>();  
  
 **public** Cell getNeighbour(Direction direction){  
 Position neighbourPosition = getPosition().shift(direction, 1);  
  
 **return \_neighbours**.stream().  
 filter(n -> n.getPosition().equals(neighbourPosition)).findAny().orElse(**null**);  
 }  
  
 **void** setNeighbour(Cell newNeighbour){  
 **if** (!canBeNeighbours(newNeighbour)){  
 **return**;  
 }  
  
 **\_neighbours**.add(newNeighbour);  
 newNeighbour.setNeighbour(**this**);  
 }  
  
 */\*\*  
 \* Может ли заданная ячейка быть соседом  
 \** ***@param newNeighbour*** *потенциальный сосед  
 \** ***@return*** *true если ячейки могут быть соседями  
 \*/* **private boolean** canBeNeighbours(Cell newNeighbour){  
 **if** (**\_neighbours**.stream().anyMatch(n -> n.getPosition().equals(newNeighbour.getPosition()))){  
 **return false**;  
 }  
  
 **int** deltaX = Math.*abs*(getPosition().getX() - newNeighbour.getPosition().getX());  
 **int** deltaY = Math.*abs*(getPosition().getY() - newNeighbour.getPosition().getY());  
  
 **int** deltaValid = 1;  
 **boolean** isDeltaXValid = deltaX == deltaValid;  
 **boolean** isDeltaYValid = deltaY == deltaValid;  
  
 **boolean** isNeighbourValid = !isDeltaXValid && isDeltaYValid || isDeltaXValid && !isDeltaYValid;  
 **if** (!isNeighbourValid){  
 **throw new** RuntimeException(**"Illegal neighbour"**);  
 }  
  
 **return true**;  
 }  
}

*/\*\*  
 \* Игровое поле, состоящее из ячеек  
 \*/***public class** Field {  
  
 **private final** Size **\_size**;  
  
 **public** Size getSize(){  
 **return \_size**;  
 }  
  
 **public** Field(Environment environment){  
 **\_size** = environment.fieldSize();  
 createCells(environment.fieldSize().width(), environment.fieldSize().height());  
 environment.fillField(**this**);  
 }

*/\* -------------------- Ячейки ------------------- \*/  
  
 /\*\*  
 \* Ячейки из которых состоит поле  
 \*/* **private final** ArrayList<Cell> **\_cells** = **new** ArrayList<>();  
  
 */\*\*  
 \* Получить ячейку по заданной позиции  
 \** ***@param position*** *позиция ячейки  
 \** ***@return*** *ячейка с заданной позиции, null если такой ячейки нет  
 \*/* **public** Cell getCell(Position position){  
 **return \_cells**.stream().  
 filter(cell -> cell.getPosition().equals(position)).findAny().orElse(**null**);  
 }  
  
 */\*\*  
 \* Создание ячеек и формирование из них поля  
 \** ***@param width*** *длина поля  
 \** ***@param height*** *высота поля  
 \*/* **private void** createCells(**int** width, **int** height){  
 **for** (**int** yCoordinate = 0; yCoordinate < height; yCoordinate++){  
 **for** (**int** xCoordinate = 0; xCoordinate < width; xCoordinate++){  
 Cell newCell = **new** Cell(**new** Position(xCoordinate, yCoordinate));  
  
 Cell northNeighbour = getCell(newCell.getPosition().shift(Direction.***NORTH***, 1));  
 Cell westNeighbour = getCell(newCell.getPosition().shift(Direction.***WEST***, 1));  
  
 **if** (northNeighbour != **null**){  
 newCell.setNeighbour(northNeighbour);  
 }  
  
 **if** (westNeighbour != **null**){  
 newCell.setNeighbour(westNeighbour);  
 }  
  
 **\_cells**.add(newCell);  
 }  
 }  
 }  
  
 */\* ----------------- Объекты на поле --------------------- \*/  
  
 /\*\*  
 \* Задать объектам на поле указанного слушателя  
 \** ***@param listener*** *слушатель объектов на поле  
 \*/* **void** addObjectInCellListener(IObjectInCellEventListener listener){  
 **for** (Cell cell : **\_cells**){  
 **for** (ObjectInCell object : cell.getObjects()){  
 **if** (!(object **instanceof** Tank)){  
 object.addListener(listener);  
 }  
 }  
 }  
 }

*/\*\*  
 \* Получить все танк, находящиеся на поле  
 \** ***@return*** *список танков  
 \*/* **public** ArrayList<Tank> getTanks(){  
 ArrayList<Tank> tanks = **new** ArrayList<>();  
 **for** (Cell cell : **\_cells**){  
 Tank tank = (Tank)cell.getObjects().stream().filter(o -> o **instanceof** Tank).findAny().orElse(**null**);  
 **if** (tank != **null**){  
 tanks.add(tank);  
 }  
 }  
 **return** tanks;  
 }  
}

*/\*\*  
 \* Главный класс игровой модели  
 \*/***public class** Game {  
  
 */\*\*  
 \* Состояние игры  
 \*/* **public enum** State {  
 ***GAME\_IS\_ON***,  
 ***WINNER\_FOUND*** }  
  
 **public** Game(Environment environment){  
 **\_field** = **new** Field(environment);  
 **\_field**.addObjectInCellListener(**new** ObjectListener());  
 }  
  
 */\* ----------------- Игровое поле ----------------- \*/  
  
 /\*\*  
 \* Игровое поле  
 \*/* **private final** Field **\_field**;  
  
 **public** Field getField(){  
 **return \_field**;  
 }  
  
 */\*\*  
 \* Запуск игры  
 \*/* **public void** start() {  
 ArrayList<Tank> tanks = **\_field**.getTanks();  
 TankListener tankListener = **new** TankListener();  
 **for** (Tank tank : tanks){  
 tank.setActive(**false**);  
 tank.addListener(tankListener);  
 }  
  
 **\_activeTank** = tanks.get(0);  
 activeTank().setActive(**true**);  
 }  
  
 */\* ---------------- Танки ----------------- \*/  
  
 /\*\*  
 \* Текущий активный танк, которым можно управлять  
 \*/* **private** Tank **\_activeTank** = **null**;  
  
 **public** Tank activeTank() {  
 **return \_activeTank**;  
 }  
  
 */\*\*  
 \* Танк - победитель, определенный после окончания игры  
 \*/* **private** Tank **\_winner** = **null**;  
  
 **public** Tank winner() {  
 **return \_winner**;  
 }  
  
 */\*\*  
 \* Получить танк-противник, для данного танка  
 \** ***@param tank*** *танк, для которого нужно определить противника  
 \** ***@return*** *танк-противник  
 \*/* **private** Tank getEnemyTank(Tank tank){  
 **return \_field**.getTanks().stream().filter(t -> t != tank).findAny().orElse(tank);  
 }  
  
 */\* ----------------- Игровой цикл ------------ \*/  
  
 /\*\*  
 \* События требующие обработки в игровом цикле  
 \*/* **private final** Queue<ObjectInCellEvent> **\_activeEvents** = **new** ArrayDeque<>();  
  
 **private final** Timer **\_timer** = **new** Timer();  
  
 */\*\*  
 \* Игровой цикл  
 \*/* **private void** gameLoop(){  
 **int** eventsCount = **\_activeEvents**.size();  
 **if** (eventsCount > 0){  
 **for** (**int** i = 0; i < eventsCount; i++){  
 ObjectInCellEvent event = **\_activeEvents**.remove();  
 event.getObject().update();  
 fireObjectChanged(event);  
 }  
 waitRendering();  
 } **else** {  
 State state = checkState();  
 **if** (state == State.***GAME\_IS\_ON***){  
 **\_activeTank** = getEnemyTank(activeTank());  
 activeTank().setActive(**true**);  
 }  
 fireGameStateGanged(state);  
 }  
 }  
  
 */\*\*  
 \* Ожидание отрисовки ереданных событий  
 \*/* **private void** waitRendering() {  
 **\_timer**.schedule(**new** TimerTask() {  
 @Override  
 **public void** run() {  
 gameLoop();  
 }  
 }, 500L);  
 }  
  
 */\*\*  
 \* Определить текущее состояние игры  
 \** ***@return*** *текущее состояние  
 \*/* **private** State checkState(){  
 ArrayList<Tank> tanks = **\_field**.getTanks();  
  
 **for** (**int** i = 0; i < tanks.size() && **\_winner** == **null**; i++){  
 Tank tank = tanks.get(i);  
 Headquarters headquarters = tank.getHeadquarters();  
 **if** (tank.getHealth() == 0 || headquarters.getCell() == **null**){  
 **\_winner** = getEnemyTank(tank);  
 }  
 }  
  
 State state = State.***GAME\_IS\_ON***;  
 **if** (**\_winner** != **null**){  
 state = State.***WINNER\_FOUND***;  
 }  
 **return** state;  
 }  
  
 */\* --------------- Игровые слушатели ---------------- \*/  
  
 /\*\*  
 \* Слушатели игровых событий  
 \*/* **private final** Set<IGameEventListener> **\_listeners** = **new** HashSet<>();  
  
 **public void** addListener(IGameEventListener listener){  
 **\_listeners**.add(listener);  
 }  
  
 */\* ---------------- События -------------------- \*/  
  
 /\*\*  
 \* Сообщить о том, что объект на поле изменился  
 \** ***@param event*** *событие, содержащий информацию об объекте  
 \*/* **private void** fireObjectChanged(ObjectInCellEvent event){  
 **for** (IGameEventListener listener : **\_listeners**){  
 listener.onObjectChanged(event);  
 }  
 }  
  
 */\*\*  
 \* Сообщить о том что состояние игры изиенились  
 \** ***@param state*** *новое состояние игры  
 \*/* **private void** fireGameStateGanged(State state){  
 **for** (IGameEventListener listener : **\_listeners**){  
 listener.onGameStateChanged(state);  
 }  
 }  
  
 */\*\*  
 \* Слушатель игровых объектов нв поле  
 \*/* **private class** ObjectListener **implements** IObjectInCellEventListener{  
  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 **\_activeEvents**.add(event);  
 }  
 }  
  
 */\*\*  
 \* Слушатель танков  
 \*/* **private class** TankListener **implements** ITankEventListener{  
  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 **if** (event.getObject() == activeTank() && event.getType() == ObjectInCellEvent.EventType.***MOVING***){  
 fireObjectChanged(event);  
 } **else** {  
 **\_activeEvents**.add(event);  
 }  
 }  
  
 @Override  
 **public void** onTankMoved(ObjectInCellEvent event) {  
 **if** (event.getObject() != activeTank()){  
 **return**;  
 }  
  
 activeTank().setActive(**false**);  
 fireObjectChanged(event);  
  
 **\_activeTank** = getEnemyTank(activeTank());  
 activeTank().setActive(**true**);  
  
 fireGameStateGanged(State.***GAME\_IS\_ON***);  
 }  
  
 @Override  
 **public void** onTankShot(ObjectInCellEvent event) {  
 activeTank().setActive(**false**);  
  
 **\_activeEvents**.add(event);  
 gameLoop();  
 }  
  
 @Override  
 **public void** onTankSkipStep(ObjectInCellEvent event) {  
 **if** (event.getObject() != activeTank()){  
 **return**;  
 }  
  
 activeTank().setActive(**false**);  
 **\_activeTank** = getEnemyTank(activeTank());  
 activeTank().setActive(**true**);  
  
 fireGameStateGanged(State.***GAME\_IS\_ON***);  
 }  
 }  
}

*/\*\*  
 \* Окружение, определяющее размеры поля и наполняющее его объектами  
 \*/***public abstract class** Environment {  
  
 **public abstract** Size fieldSize();  
  
 **public abstract void** fillField(Field field);  
}

*/\*\*  
 \* Слушатель игровых событий  
 \*/***public interface** IGameEventListener {  
  
 **void** onObjectChanged(ObjectInCellEvent event);  
  
 **void** onGameStateChanged(Game.State state);  
}

*/\*\*  
 \* Слушатель игровых объектов  
 \*/***public interface** IObjectInCellEventListener **extends** EventListener {  
  
 **void** onObjectInCellAction(ObjectInCellEvent event);  
}

*/\*\*  
 \* Слушатель танков  
 \*/***public interface** ITankEventListener **extends** IObjectInCellEventListener {  
  
 **void** onTankMoved(ObjectInCellEvent event);  
  
 **void** onTankShot(ObjectInCellEvent event);  
  
 **void** onTankSkipStep(ObjectInCellEvent event);  
}

*/\*\*  
 \* События, генерируемы игровыми объектами  
 \*/***public class** ObjectInCellEvent {  
  
 **public enum** EventType{  
 ***DESTROYING***,  
 ***MOVING***,  
 ***DAMAGED*** }  
  
 **private** ObjectInCell **\_object**;  
  
 **public** ObjectInCell getObject(){  
 **return \_object**;  
 }  
  
 **private** EventType **\_type**;  
  
 **public** EventType getType() { **return \_type**; }  
  
 **public** ObjectInCellEvent(ObjectInCell object, EventType type){  
 **\_object** = object;  
 **\_type** = type;  
 }  
}

*/\*\*  
 \* Представление объекта в ячейке  
 \*/***public abstract class** ObjectInCellWidget **extends** JPanel {  
  
 **public** ObjectInCellWidget(ObjectInCell object){  
 **\_object** = object;  
 setPreferredSize(getDimension());  
 setLayout(**new** GridBagLayout());  
 repaint();  
 revalidate();  
 setOpaque(**false**);  
 }  
  
 */\*\*  
 \* Виджет ячейки, в котором находится виджет объекта  
 \*/* **private** CellWidget **\_cellWidget**;  
  
 */\*\*  
 \* Установить виджет ячейки  
 \** ***@param cellWidget*** *задаваемый виджет ячейки  
 \*/* **void** setCellWidget(CellWidget cellWidget){  
 **\_cellWidget** = cellWidget;  
 }  
  
 */\*\*  
 \* Получить виджет ячейки  
 \** ***@return*** *виджет ячейки  
 \*/* **public** CellWidget getCellWidget() {  
 **return \_cellWidget**;  
 }  
  
 */\*\*  
 \* Модель объекта в ячейки  
 \*/* **private final** ObjectInCell **\_object**;  
  
 */\*\*  
 \* Получить объект, соответсвующий виджету  
 \** ***@return*** *модель объекта  
 \*/* **public** ObjectInCell getObject(){  
 **return \_object**;  
 }  
  
 */\*\*  
 \* Размеры объекта  
 \** ***@return*** *размер объекта  
 \*/* **public** Dimension getDimension(){  
 **return new** Dimension(80, 80);  
 }  
  
 */\*\*  
 \* Получить слой представления  
 \** ***@return*** *слой представления  
 \*/* **public** Layer getLayer(){  
 **return** Layer.***BOTTOM***;  
 }  
  
 */\*\*  
 \* Получить файл с изображением объекта  
 \** ***@return*** *файл с изображением  
 \*/* **protected abstract** File getImageFile();  
  
 */\*\*  
 \* Получить изображение объекта  
 \** ***@return*** *изображение объекта  
 \*/* **protected** BufferedImage getImage(){  
 File file = getImageFile();  
  
 BufferedImage image = **null**;  
 **try** {  
 image = ImageIO.*read*(file);  
 image = ImageUtils.*resizeImage*(image, getDimension().**width**, getDimension().**height**);  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **return** image;  
 }  
  
 */\*\*  
 \* Отрисовать получение урона объетктом  
 \*/* **protected void** getDamage(){  
 destroy();  
 }  
  
 */\*\*  
 \* Уничтожить представление объекта  
 \*/* **protected void** destroy(){  
 **\_cellWidget**.removeObjectWidget(**this**);  
 }  
  
 @Override  
 **protected void** paintComponent(Graphics g){  
 **super**.paintComponent(g);  
 setPreferredSize(getDimension());  
 revalidate();  
 g.drawImage(getImage(), 0, 0, **null**);  
 }  
}

*/\*\*  
 \* Представление движущегося объекта  
 \*/***public abstract class** MovableObjectWidget **extends** ObjectInCellWidget{  
  
 **public** MovableObjectWidget(MovableObject object) {  
 **super**(object);  
 }  
  
 */\*\*  
 \* Переместить виджет в указанный виджет ячейки  
 \** ***@param cellWidget*** *виджет ячейки, куда нужно переместиься  
 \*/* **protected void** moveTo(CellWidget cellWidget){  
 **if** (getCellWidget() != **null**) {  
 getCellWidget().setActive(**false**);  
 getCellWidget().removeObjectWidget(**this**);  
 }  
 **if** (cellWidget != **null**) {  
 cellWidget.addObjectWidget(**this**);  
 }  
 }  
  
 @Override  
 **protected** BufferedImage getImage() {  
 **return** getImageByDirection(((MovableObject)getObject()).getDirection());  
 }  
  
 */\*\*  
 \* Получить изображение объект, соответсвующее направлению движения  
 \** ***@param direction*** *еаправление движения объекта  
 \** ***@return*** *изображение объекта  
 \*/* **private** BufferedImage getImageByDirection(Direction direction){  
 Map<Direction, Double> directionToAngles = **new** HashMap<>();  
 directionToAngles.put(Direction.***NORTH***, 0.0);  
 directionToAngles.put(Direction.***SOUTH***, 180.0);  
 directionToAngles.put(Direction.***WEST***, 270.0);  
 directionToAngles.put(Direction.***EAST***, 90.0);  
  
 BufferedImage image = **super**.getImage();  
 image = ImageUtils.*rotateImage*(image, directionToAngles.get(direction));  
 **return** image;  
 }  
}

*/\*\*  
 \* Преставление танка  
 \*/***public class** TankWidget **extends** MovableObjectWidget {  
  
 **public** TankWidget(Tank tank){  
 **super**(tank);  
 addKeyListener(**new** KeyListener());  
 }  
  
 */\*\*  
 \* Получить модель танка  
 \** ***@return*** *модель танка  
 \*/* **private** Tank getTank(){  
 **return** (Tank) getObject();  
 }  
  
 */\*\*  
 \* Установить активность виджета  
 \** ***@param active*** *активность виджета  
 \*/* **public void** setActive(**boolean** active){  
 setFocusable(active);  
 getCellWidget().setActive(active);  
 requestFocus();  
 repaint();  
 }  
  
 */\*\*  
 \* Цвет танка  
 \*/* **private** Color **\_color**;  
  
 */\*\*  
 \* Задать цвет  
 \** ***@param color*** *задаваемый цвет  
 \*/* **void** setColor(Color color){  
 **\_color** = color;  
 }  
  
 */\*\*  
 \* Получить цвет танка  
 \** ***@return*** *цвет танка  
 \*/* Color getColor(){  
 **return \_color**;  
 }  
  
 */\*\*  
 \* Отрисовать получение урона  
 \*/* @Override  
 **public void** getDamage(){  
 **if** (getTank().getHealth() <= 0){  
 **super**.getDamage();  
 }  
 }  
  
 @Override  
 **public** Dimension getDimension(){  
 **return new** Dimension(60, 60);  
 }  
  
 @Override  
 **protected** File getImageFile() {  
 **return** getImageFileByColor(getColor());  
 }  
  
 */\*\*  
 \* Получить файл с изображением танка, соответствующее его цвету  
 \** ***@param color*** *цвет танка  
 \** ***@return*** *файл с изоборажением танка  
 \*/* **private** File getImageFileByColor(Color color){  
 String path = color == Color.***BLUE*** ? **"resources/blue\_tank.png"** : **"resources/orange\_tank.png"**;  
 **return new** File(path);  
 }  
  
 */\*\*  
 \* Слушатель пользовательского ввода  
 \*/* **private class** KeyListener **implements** java.awt.event.KeyListener{  
  
 @Override  
 **public void** keyTyped(KeyEvent e) {  
  
 }  
  
 @Override  
 **public void** keyPressed(KeyEvent e) {  
 **int** key = e.getKeyCode();  
 move(key);  
 pass(key);  
 shoot(key);  
 rotate(key);  
 setActive(getTank().isActive());  
 }  
  
 **private void** move(**int** keyCode){  
 **if** (keyCode == KeyEvent.***VK\_SPACE***){  
 getTank().move();  
 }  
 }  
  
 **private void** shoot(**int** keyCode){  
 **if** (keyCode == KeyEvent.***VK\_ENTER***){  
 getTank().shoot();  
 }  
 }  
  
 **private void** pass(**int** keyCode){  
 **if** (keyCode == KeyEvent.***VK\_BACK\_SPACE***){  
 getTank().pass();  
 }  
 }  
  
 **private void** rotate(**int** keyCode){  
 Map<Integer, Direction> keyCodeToDirection = **new** HashMap<>();  
 keyCodeToDirection.put(KeyEvent.***VK\_W***, Direction.***NORTH***);  
 keyCodeToDirection.put(KeyEvent.***VK\_A***, Direction.***WEST***);  
 keyCodeToDirection.put(KeyEvent.***VK\_S***, Direction.***SOUTH***);  
 keyCodeToDirection.put(KeyEvent.***VK\_D***, Direction.***EAST***);  
  
 **if** (keyCodeToDirection.containsKey(keyCode)){  
 getTank().rotate(keyCodeToDirection.get(keyCode));  
 }  
 }  
  
 @Override  
 **public void** keyReleased(KeyEvent e) {  
  
 }  
 }  
}

*/\*\*  
\* Представление снаряда  
\*/***public class** BulletWidget **extends** MovableObjectWidget {  
  
 **public** BulletWidget(Bullet bullet){  
 **super**(bullet);  
 }  
  
 @Override  
 **protected** File getImageFile() {  
 String path = ((Bullet)getObject()).isDetonating() ? **"resources/explosion.png"** : **"resources/bullet.png"**;  
 **return new** File(path);  
 }  
  
 @Override  
 **public** Dimension getDimension(){  
 **int** size = ((Bullet)getObject()).isDetonating() ? 60 : 20;  
 **return new** Dimension(size, size);  
 }  
  
 @Override  
 **public** Layer getLayer(){  
 **return** Layer.***TOP***;  
 }  
}

*/\*\*  
 \* Представление штаба  
 \*/***public class** HeadquartersWidget **extends** ObjectInCellWidget{  
  
 **public** HeadquartersWidget(Headquarters headquarters){  
 **super**(headquarters);  
 }  
  
 */\*\*  
 \* Цвет штаба  
 \*/* **private** Color **\_color**;  
  
 */\*\*  
 \* Задать цвет  
 \** ***@param color*** *задаваемый цвет  
 \*/* **void** setColor(Color color){  
 **\_color** = color;  
 }  
  
 */\*\*  
 \* Получить цвет  
 \** ***@return*** *цвет штаба  
 \*/* **public** Color getColor(){  
 **return \_color**;  
 }  
  
 @Override  
 **protected** File getImageFile() {  
 **return** getImageFileByColor(getColor());  
 }  
  
 */\*\*  
 \* Получить изображение штаба, соответсвующее его цвету  
 \** ***@param color*** *цвет штаба  
 \** ***@return*** *файл с изображением штаба  
 \*/* **private** File getImageFileByColor(Color color) {  
 String path = color == Color.***BLUE*** ? **"resources/blue\_HQ.png"** : **"resources/orange\_HQ.png"**;  
 **return new** File(path);  
 }  
}

*/\*\*  
 \* Представление стены  
 \*/***public class** WallWidget **extends** ObjectInCellWidget{  
  
 **public** WallWidget(Wall wall){  
 **super**(wall);  
 }  
  
 @Override  
 **protected** File getImageFile() {  
 **return new** File(**"resources/wall.png"**);  
 }  
}

*/\*\*  
 \* Представление воды  
 \*/***public class** WaterWidget **extends** ObjectInCellWidget{  
  
 **public** WaterWidget(Water water){  
 **super**(water);  
 }  
  
 @Override  
 **protected** File getImageFile() {  
 **return new** File(**"resources/water.png"**);  
 }  
}

*/\*\*  
 \* Представление ячейки  
 \*/***public class** CellWidget **extends** JPanel {  
  
 */\*\*  
 \* Размер ячейки  
 \*/* **private static final int *CELL\_SIZE*** = 80;  
  
 */\*\*  
 \* Цвет ячейки  
 \*/* **private static final** Color ***BACKGROUND*** = Color.*decode*(**"#585553"**);  
  
 */\*\*  
 \* Модель ячейки  
 \*/* **private final** Cell **\_cell**;  
  
 **public** CellWidget(Cell cell, WidgetPool pool){  
 **\_cell** = cell;  
 setPreferredSize(getDimension());  
 setBackground(***BACKGROUND***);  
 setLayout(**new** GridBagLayout());  
  
 **for** (ObjectInCell object : cell.getObjects()){  
 ObjectInCellWidget widget = pool.getWidget(object);  
 addObjectWidget(widget);  
 }  
 }  
  
 */\*\*  
 \* Изменить активность, если ячейка активна, она подцвечивается зеленым  
 \** ***@param active*** *активность ячейки  
 \*/* **void** setActive(**boolean** active){  
 **if** (active){  
 setBorder(BorderFactory.*createLineBorder*(Color.***GREEN***));  
 } **else** {  
 setBorder(BorderFactory.*createEmptyBorder*());  
 }  
 repaint();  
 }  
  
 */\*\*  
 \* Получить размер ячейки  
 \** ***@return*** *размер ячейки  
 \*/* **public** Dimension getDimension(){  
 **return new** Dimension(***CELL\_SIZE***, ***CELL\_SIZE***);  
 }  
  
 */\*\*  
 \* Дочерние виджеты, упорядоченные по слоям  
 \*/* **private final** List<ObjectInCellWidget> **\_layers** = **new** ArrayList<>();  
  
 */\*\*  
 \* Добавить виджет объекта  
 \** ***@param widget*** *виджет добавляемого объекта  
 \*/* **void** addObjectWidget(ObjectInCellWidget widget){  
 widget.setCellWidget(**this**);  
 **\_layers**.add(widget);  
 repaintLayers();  
 }  
  
 */\*\*  
 \* Удалить виджет объекта  
 \** ***@param widget*** *удаляемый виджет  
 \*/* **void** removeObjectWidget(ObjectInCellWidget widget){  
 widget.setCellWidget(**null**);  
 **\_layers**.remove(widget);  
 repaintLayers();  
 }  
  
 */\*\*  
 \* Переписовать ячейку с учетом порядка дочерних элементов  
 \*/* **private void** repaintLayers(){  
 removeAll();  
  
 **for** (ObjectInCellWidget layer : **\_layers**) {  
 layer.removeAll();  
 }  
  
 **\_layers**.sort(Comparator.*comparing*(ObjectInCellWidget::getLayer));  
  
 **if** (**\_layers**.size() > 0){  
 add(**\_layers**.get(0));  
 }  
  
 **for** (**int** i = 0; i < **\_layers**.size() - 1; i++){  
 **\_layers**.get(i).add(**\_layers**.get(i + 1));  
 }  
  
 repaint();  
 }  
}

*/\*\*  
 \* Слой представления  
 \*/***public enum** Layer {  
 ***BOTTOM***,  
 ***TOP***}

*/\*\*  
 \* Представление поля  
 \*/***public class** FieldWidget **extends** JPanel {  
  
 */\*\*  
 \* Модель поля  
 \*/* **private final** Field **\_field**;  
  
 **public** FieldWidget(Field field, WidgetPool pool){  
 **\_field** = field;  
 setLayout(**new** GridLayout(**\_field**.getSize().width(), **\_field**.getSize().height(), 10, 10));  
 setBackground(Color.***BLACK***);  
 fill(pool);  
 initializePlayers(pool);  
 }  
  
 */\*\*  
 \* Заполнение поля  
 \** ***@param pool*** *пул виджетов, содержащий виджеты ячейки  
 \*/* **private void** fill(WidgetPool pool){  
 **for** (**int** y = 0; y < **\_field**.getSize().height(); y++){  
 **for** (**int** x = 0; x < **\_field**.getSize().width(); x++){  
 CellWidget cellWidget = pool.getWidget(**\_field**.getCell(**new** Position(x, y)));  
 add(cellWidget);  
 }  
 }  
 }  
  
 */\*\*  
 \* Присвоить игрокам их цвета  
 \** ***@param pool,*** *пул виджетов, содержащий виджеты танков и штабов  
 \*/* **private void** initializePlayers(WidgetPool pool){  
 Color[] colors = **new** Color[] {Color.***BLUE***, Color.***ORANGE***};  
 **for** (**int** i = 0; i < **\_field**.getTanks().size(); i++){  
 Tank tank = **\_field**.getTanks().get(i);  
 TankWidget tankWidget = (TankWidget) pool.getWidget(tank);  
 tankWidget.setColor(colors[i]);  
  
 Headquarters headquarters = tank.getHeadquarters();  
 HeadquartersWidget headquartersWidget = (HeadquartersWidget) pool.getWidget(headquarters);  
 headquartersWidget.setColor(colors[i]);  
 }  
 }  
}

*/\*\*  
 \* Пул виджетов, расположенных на поле  
 \*/***public class** WidgetPool {  
  
 **private final** Map<ObjectInCell, ObjectInCellWidget> **\_objectWidgets** = **new** HashMap<>();  
  
 **private final** Map<Cell, CellWidget> **\_cellWidgets** = **new** HashMap<>();  
  
 **public** ObjectInCellWidget getWidget(ObjectInCell object){  
 ObjectInCellWidget widget = **\_objectWidgets**.get(object);  
 **if** (widget == **null**){  
 widget = createWidget(object);  
 }  
 **return** widget;  
 }  
  
 **public** ObjectInCellWidget removeWidget(ObjectInCell object){  
 **return \_objectWidgets**.remove(object);  
 }  
  
 **private** ObjectInCellWidget createWidget(ObjectInCell object){  
 **if** (**\_objectWidgets**.containsKey(object)){  
 **throw new** IllegalArgumentException();  
 }  
  
 ObjectInCellWidget widget = **null**;  
 **if** (object **instanceof** Tank tank){  
 widget = **new** TankWidget(tank);  
 } **else if** (object **instanceof** Wall wall){  
 widget = **new** WallWidget(wall);  
 } **else if** (object **instanceof** Headquarters headquarters){  
 widget = **new** HeadquartersWidget(headquarters);  
 } **else if** (object **instanceof** Water water){  
 widget = **new** WaterWidget(water);  
 } **else if** (object **instanceof** Bullet bullet){  
 widget = **new** BulletWidget(bullet);  
 }  
  
 **\_objectWidgets**.put(object, widget);  
 **return** widget;  
 }  
  
 **public** CellWidget getWidget(Cell cell){  
 CellWidget widget = **\_cellWidgets**.get(cell);  
 **if** (widget == **null**){  
 widget = createWidget(cell);  
 }  
 **return** widget;  
 }  
  
 **private** CellWidget createWidget(Cell cell){  
 **if** (**\_cellWidgets**.containsKey(cell)){  
 **throw new** IllegalArgumentException();  
 }  
  
 CellWidget widget = **new** CellWidget(cell, **this**);  
 **\_cellWidgets**.put(cell, widget);  
 **return** widget;  
 }  
}

*/\*\*  
 \* Представление панели игрока  
 \*/***public class** PlayerPanel **extends** JPanel {  
  
 */\*\*  
 \* Танк игрока  
 \*/* **private final** Tank **\_tank**;  
  
 */\*\*  
 \* Максимальное кол-во жизней игрока  
 \*/* **private final int \_maxHeath**;  
  
 */\*\*  
 \* Цвет игрока  
 \*/* **private final** Color **\_color**;  
  
 **public** PlayerPanel(TankWidget tankWidget){  
 **\_tank** = (Tank) tankWidget.getObject();  
 **\_maxHeath** = **\_tank**.getHealth();  
 **\_color** = tankWidget.getColor();  
  
 setPreferredSize(**new** Dimension(150, 600));  
 setLayout(**new** GridLayout(3, 1, 0, 50));  
 setBackground(Color.***BLACK***);  
  
 update();  
 }  
  
 */\*\*  
 \* Обновить информацию об игроке  
 \*/* **public void** update(){  
 removeAll();  
 add(getFlagBar());  
 add(getHeathBar());  
 add(getReloadGunBar());  
 repaint();  
 revalidate();  
 }  
  
 */\*\*  
 \* Получить панель с жизнями игрока  
 \** ***@return*** *панель с жизнями игрока  
 \*/* **private** JPanel getHeathBar(){  
 JPanel heathBar = **new** JPanel();  
 heathBar.setLayout(**new** GridLayout(**\_maxHeath**, 1));  
 heathBar.setBackground(Color.***BLACK***);  
  
 **for** (**int** i = 1; i <= **\_maxHeath**; i++){  
 **boolean** heathActive = i <= **\_tank**.getHealth();  
 heathBar.add(**new** JLabel(**new** ImageIcon(getHeartImage(heathActive))));  
 }  
  
 **return** heathBar;  
 }  
  
 */\*\*  
 \* Получить изображение сердца  
 \** ***@param isActive*** *активное сердце - красное, неактивное - серое  
 \** ***@return*** *изображения сердца  
 \*/* **private** BufferedImage getHeartImage(**boolean** isActive){  
 String path = isActive ? **"resources/red\_heart.png"** : **"resources/grey\_heart.png"**;  
  
 **return** getImage(**new** File(path), **new** Dimension(50, 50));  
 }  
  
 */\*\*  
 \* Получить панель с флагом игрока  
 \** ***@return*** *панель с флагом  
 \*/* **private** JPanel getFlagBar(){  
 JPanel flagBar = **new** JPanel();  
 flagBar.setLayout(**new** GridBagLayout());  
 flagBar.setBackground(Color.***BLACK***);  
 flagBar.add(**new** JLabel(**new** ImageIcon(getFlagImage())));  
 **return** flagBar;  
 }  
  
 */\*\*  
 \* Получить изображение флага игрока, соотыетсвующее его цвету  
 \** ***@return*** *изображение флага  
 \*/* **private** BufferedImage getFlagImage(){  
 String path = **\_color** == Color.***BLUE*** ? **"resources/blue\_flag.png"** : **"resources/orange\_flag.png"**;  
  
 **return** getImage(**new** File(path), **new** Dimension(100, 100));  
 }  
  
 */\*\*  
 \* Получить панель перезарядки орудия  
 \** ***@return*** *панель перезарядки  
 \*/* **private** JPanel getReloadGunBar(){  
 JPanel reloadPanel = **new** JPanel();  
 reloadPanel.setLayout(**new** GridBagLayout());  
 reloadPanel.setBackground(Color.***BLACK***);  
  
 JLabel label = **null**;  
 **if** (**\_tank**.getCurrentReloadTime() > 0){  
 label = **new** JLabel(Integer.*toString*(**\_tank**.getCurrentReloadTime()),  
 **new** ImageIcon(getReloadImage(**false**)),  
 JLabel.***CENTER***);  
 label.setHorizontalTextPosition(JLabel.***RIGHT***);  
 label.setVerticalTextPosition(JLabel.***CENTER***);  
 label.setFont(**new** Font(**"Serif"**, Font.***PLAIN***, 36));  
 label.setForeground(Color.*decode*(**"#6d6d6d"**));  
 } **else** {  
 label = **new** JLabel(**new** ImageIcon(getReloadImage(**true**)));  
 }  
  
 reloadPanel.add(label);  
 **return** reloadPanel;  
 }  
  
 */\*\*  
 \* Получить изображение перезарядки орудия  
 \** ***@param isActive*** *активность орудия  
 \** ***@return*** *изображение перезарядки  
 \*/* **private** BufferedImage getReloadImage(**boolean** isActive){  
 String path = isActive ? **"resources/bullet.png"** : **"resources/grey\_bullet.png"**;  
  
 **return** getImage(**new** File(path), **new** Dimension(50, 50));  
 }  
  
 */\*\*  
 \* Получить изображение из файла  
 \** ***@param file*** *файл с изображением  
 \** ***@param dimension*** *размер изображения  
 \** ***@return*** *изображение из файла  
 \*/* **private** BufferedImage getImage(File file, Dimension dimension){  
 BufferedImage image = **null**;  
 **try** {  
 image = ImageIO.*read*(file);  
 image = ImageUtils.*resizeImage*(image, dimension.**width**, dimension.**height**);  
 } **catch** (IOException e){  
 e.printStackTrace();  
 }  
 **return** image;  
 }  
}

*/\*\*  
 \* Представление игры  
 \*/***public class** GamePanel **extends** JFrame {  
  
 */\*\*  
 \* Пул виджетов, содержащий виджеты ячеек и объектов на поле  
 \*/* **private final** WidgetPool **\_pool** = **new** WidgetPool();  
  
 */\*\*  
 \* Панели игроков  
 \*/* **private final** List<PlayerPanel> **\_playerPanels** = **new** ArrayList<>();  
  
 */\*\*  
 \* Модель игры  
 \*/* **private** Game **\_game**;  
  
 **public** GamePanel(){  
 create();  
 pack();  
 setDefaultCloseOperation(***EXIT\_ON\_CLOSE***);  
 setResizable(**false**);  
 setVisible(**true**);  
 start();  
 }  
  
 */\*\*  
 \* Создать игру и ее представление  
 \*/* **private void** create(){  
 **\_game** = **new** Game(**new** SimpleEnvironment());  
 FieldWidget fieldWidget = **new** FieldWidget(**\_game**.getField(), **\_pool**);  
 **\_game**.addListener(**new** GameListener());  
  
 JPanel content = (JPanel) getContentPane();  
 content.removeAll();  
 content.setLayout(**new** BoxLayout(content, BoxLayout.***X\_AXIS***));  
  
 **for** (Tank tank : **\_game**.getField().getTanks()){  
 PlayerPanel panel = **new** PlayerPanel((TankWidget)**\_pool**.getWidget(tank));  
 content.add(panel);  
 **\_playerPanels**.add(panel);  
 }  
  
 content.add(fieldWidget, 1);  
  
 repaint();  
 revalidate();  
 }  
  
 */\*\*  
 \* Запустить игру  
 \*/* **private void** start(){  
 **\_game**.start();  
 ((TankWidget) **\_pool**.getWidget(**\_game**.activeTank())).setActive(**true**);  
 }  
  
 */\*\*  
 \* Перезапустить игру  
 \*/* **private void** restart(){  
 create();  
 start();  
 }

*/\*\*  
 \* Слушатель игровых событий  
 \*/* **private class** GameListener **implements** IGameEventListener {  
  
 @Override  
 **public void** onObjectChanged(ObjectInCellEvent event) {  
 ObjectInCellWidget widget = **\_pool**.getWidget(event.getObject());  
 **if** (event.getType() == ObjectInCellEvent.EventType.***MOVING*** && widget **instanceof** MovableObjectWidget movableWidget){  
 CellWidget cellWidget = event.getObject().getCell() != **null** ? **\_pool**.getWidget(event.getObject().getCell()) : **null**;  
 movableWidget.moveTo(cellWidget);  
 } **else**{  
 widget.getDamage();  
 **if** (event.getType() == ObjectInCellEvent.EventType.***DESTROYING***){  
 **\_pool**.removeWidget(event.getObject());  
 }  
 }  
 }  
  
 @Override  
 **public void** onGameStateChanged(Game.State state) {  
 **for** (PlayerPanel panel: **\_playerPanels** ) {  
 panel.update();  
 }  
  
 **if** (state == Game.State.***GAME\_IS\_ON***){  
 ((TankWidget) **\_pool**.getWidget(**\_game**.activeTank())).setActive(**true**);  
 } **else if** (state == Game.State.***WINNER\_FOUND***){  
 String winnerName = getColorName(((TankWidget)**\_pool**.getWidget(**\_game**.winner())).getColor());  
 JOptionPane.*showMessageDialog*(GamePanel.**this**, winnerName + **" танк победил"**);  
 restart();  
 }  
 }  
  
 **private** String getColorName(Color color){  
 **if** (color == Color.***BLUE***){  
 **return "Синий"**;  
 } **else** {  
 **return "Оранжевый"**;  
 }  
 }  
 }  
}

**public class** ImageUtils {  
  
 **public static** BufferedImage resizeImage(BufferedImage img, Integer width, Integer height) {  
 Image tmpImg = img.getScaledInstance(width, height, Image.***SCALE\_SMOOTH***);  
 BufferedImage bufferedImage = **new** BufferedImage(width, height, BufferedImage.***TYPE\_INT\_ARGB***);  
  
 Graphics2D g = bufferedImage.createGraphics();  
 g.drawImage(tmpImg, 0, 0, **null**);  
 g.dispose();  
  
 **return** bufferedImage;  
 }  
  
 **public static** BufferedImage rotateImage(BufferedImage img, **double** degree){  
 **double** angle = Math.*toRadians*(degree);  
  
 **return** *tilt*(img, angle);  
 }  
  
 **public static** BufferedImage tilt(BufferedImage image, **double** angle) {  
 **double** sin = Math.*abs*(Math.*sin*(angle)), cos = Math.*abs*(Math.*cos*(angle));  
 **int** w = image.getWidth(), h = image.getHeight();  
 **int** neww = (**int**)Math.*floor*(w\*cos+h\*sin), newh = (**int**)Math.*floor*(h\*cos+w\*sin);  
 GraphicsConfiguration gc = *getDefaultConfiguration*();  
 BufferedImage result = gc.createCompatibleImage(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(image, **null**);  
 g.dispose();  
 **return** result;  
 }  
  
 **public static** GraphicsConfiguration getDefaultConfiguration() {  
 GraphicsEnvironment ge = GraphicsEnvironment.*getLocalGraphicsEnvironment*();  
 GraphicsDevice gd = ge.getDefaultScreenDevice();  
 **return** gd.getDefaultConfiguration();  
 }  
}

## 3.8 Реализация ключевых тестовых случаев

**public class** BulletTest {  
  
 Bullet **bullet**;  
 Cell **startCell**;  
 Cell **targetCell**;  
  
 @BeforeEach  
 **public void** createTestConfiguration(){  
 **startCell** = **new** Cell(**new** Position(0, 0));  
 **targetCell** = **new** Cell(**new** Position(0, 1));  
 **startCell**.setNeighbour(**targetCell**);  
 **bullet** = **new** Bullet(Direction.***SOUTH***, **startCell**);  
 }  
  
 @Test  
 **public void** constructor\_creationBullet(){  
 **bullet** = **new** Bullet(Direction.***NORTH***, **startCell**);  
  
 Assertions.*assertSame*(**startCell**, **bullet**.getCell());  
 Assertions.*assertEquals*(Direction.***NORTH***, **bullet**.getDirection());  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithTank(){  
 Tank tank = **new** Tank();  
  
 **boolean** result = **bullet**.canFaceWith(tank);  
  
 Assertions.*assertTrue*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithWall(){  
 Obstacle wall = **new** Wall();  
  
 **boolean** result = **bullet**.canFaceWith(wall);  
  
 Assertions.*assertTrue*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithWater(){  
 Obstacle water = **new** Water();  
  
 **boolean** result = **bullet**.canFaceWith(water);  
  
 Assertions.*assertTrue*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithBullet(){  
 Bullet otherBullet = **new** BulletForTest();  
  
 **boolean** result = **bullet**.canFaceWith(otherBullet);  
  
 Assertions.*assertTrue*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithSameObject(){  
 **boolean** result = **bullet**.canFaceWith(**bullet**);  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithTank(){  
 ObjectInCellEvent[] actualEvents = {**null**};  
  
 Tank tank = **new** Tank();  
  
 **bullet**.addListener(**new** IObjectInCellEventListener() {  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 Assertions.*assertSame*(**bullet**, event.getObject());  
 Assertions.*assertEquals*(ObjectInCellEvent.EventType.***DESTROYING***, event.getType());  
 actualEvents[0] = event;  
 }  
 });  
  
 **bullet**.faceWith(tank);  
  
 Assertions.*assertNotNull*(actualEvents[0]);  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithWall(){  
 ObjectInCellEvent[] actualEvents = {**null**};  
  
 Wall wall = **new** Wall();  
  
 **bullet**.addListener(**new** IObjectInCellEventListener() {  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 Assertions.*assertSame*(**bullet**, event.getObject());  
 Assertions.*assertEquals*(ObjectInCellEvent.EventType.***DESTROYING***, event.getType());  
 actualEvents[0] = event;  
 }  
 });  
  
 **bullet**.faceWith(wall);  
  
 Assertions.*assertNotNull*(actualEvents[0]);  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithWater(){  
 ObjectInCellEvent[] actualEvents = {**null**};  
  
 Water water = **new** Water();  
  
 **bullet**.addListener(**new** IObjectInCellEventListener() {  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 Assertions.*assertSame*(**bullet**, event.getObject());  
 Assertions.*assertEquals*(ObjectInCellEvent.EventType.***MOVING***, event.getType());  
 actualEvents[0] = event;  
 }  
 });  
  
 **bullet**.faceWith(water);  
  
 Assertions.*assertNull*(actualEvents[0]);  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithHeadquarters(){  
 ObjectInCellEvent[] actualEvents = {**null**};  
  
 Headquarters headquarters = **new** Headquarters();  
  
 **bullet**.addListener(**new** IObjectInCellEventListener() {  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 Assertions.*assertSame*(**bullet**, event.getObject());  
 Assertions.*assertEquals*(ObjectInCellEvent.EventType.***DESTROYING***, event.getType());  
 actualEvents[0] = event;  
 }  
 });  
  
 **bullet**.faceWith(headquarters);  
  
 Assertions.*assertNotNull*(actualEvents[0]);  
 }  
  
 @Test  
 **public void** move\_movingInEmptyCell(){  
 ObjectInCellEvent[] actualEvents = {**null**};  
  
 **bullet**.addListener(**new** IObjectInCellEventListener() {  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 Assertions.*assertSame*(**bullet**, event.getObject());  
 Assertions.*assertEquals*(ObjectInCellEvent.EventType.***MOVING***, event.getType());  
 Assertions.*assertSame*(**targetCell**, **bullet**.getCell());  
 actualEvents[0] = event;  
 }  
 });  
  
 **bullet**.move();  
  
 Assertions.*assertNotNull*(actualEvents[0]);  
 }  
  
 @Test  
 **public void** move\_movingInCellWithWall(){  
 List<ObjectInCellEvent> events = **new** ArrayList<>();  
  
 **targetCell**.addObject(**new** Wall());  
  
 **bullet**.addListener(**new** IObjectInCellEventListener() {  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 Assertions.*assertSame*(**bullet**, event.getObject());  
 Assertions.*assertSame*(**targetCell**, **bullet**.getCell());  
 events.add(event);  
 }  
 });  
  
 **bullet**.move();  
  
 List<ObjectInCellEvent> expectedEvents = **new** ArrayList<>();  
 expectedEvents.add(**new** ObjectInCellEvent(**bullet**, ObjectInCellEvent.EventType.***DESTROYING***));  
  
 Assertions.*assertEquals*(expectedEvents.size(), events.size());  
 **for** (**int** i = 0; i < expectedEvents.size(); i++){  
 Assertions.*assertEquals*(expectedEvents.get(i).getType(), events.get(i).getType());  
 }  
 }  
  
 @Test **void** update\_moving(){  
 ObjectInCellEvent[] actualEvents = {**null**};  
  
 **bullet**.addListener(**new** IObjectInCellEventListener() {  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 Assertions.*assertSame*(**bullet**, event.getObject());  
 Assertions.*assertEquals*(ObjectInCellEvent.EventType.***MOVING***, event.getType());  
 Assertions.*assertSame*(**targetCell**, **bullet**.getCell());  
 actualEvents[0] = event;  
 }  
 });  
  
 **bullet**.update();  
  
 Assertions.*assertNotNull*(actualEvents[0]);  
 }  
  
 @Test  
 **public void** update\_detonating(){  
 ObjectInCellEvent[] actualEvents = {**null**};  
  
 **targetCell**.addObject(**new** Wall());  
  
 **bullet**.addListener(**new** IObjectInCellEventListener() {  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 Assertions.*assertSame*(**bullet**, event.getObject());  
 Assertions.*assertEquals*(ObjectInCellEvent.EventType.***DESTROYING***, event.getType());  
 Assertions.*assertSame*(**targetCell**, **bullet**.getCell());  
 actualEvents[0] = event;  
 }  
 });  
  
 **targetCell**.addObject(**bullet**);  
 **bullet**.update();  
  
 Assertions.*assertNotNull*(actualEvents[0]);  
 Assertions.*assertNull*(**bullet**.getCell());  
 }  
}

**public class** CellTest {  
  
 **private** Cell **cell**;  
  
 @BeforeEach  
 **public void** testConfiguration(){  
 **cell** = **new** Cell(**new** Position(0, 0));  
 }  
  
 @Test  
 **public void** constructor\_cellCreation(){  
 **cell** = **new** Cell(**new** Position(1,2));  
  
 Assertions.*assertEquals*(1, **cell**.getPosition().getX());  
 Assertions.*assertEquals*(2, **cell**.getPosition().getY());  
 Assertions.*assertEquals*(0, **cell**.getObjects().size());  
 Assertions.*assertNull*(**cell**.getNeighbour(Direction.***NORTH***));  
 Assertions.*assertNull*(**cell**.getNeighbour(Direction.***SOUTH***));  
 Assertions.*assertNull*(**cell**.getNeighbour(Direction.***EAST***));  
 Assertions.*assertNull*(**cell**.getNeighbour(Direction.***WEST***));  
 }  
  
 @Test  
 **public void** addObject\_addingInEmptyCell(){  
 ObjectInCell object = **new** Tank();  
  
 **boolean** result = **cell**.addObject(object);  
  
 Assertions.*assertTrue*(result);  
 Assertions.*assertTrue*(**cell**.getObjects().contains(object));  
 Assertions.*assertSame*(**cell**, object.getCell());  
 }  
  
 @Test  
 **public void** addObject\_addingInCellContainsObjectWithRightCollision(){  
 ObjectInCell tank = **new** Tank();  
 ObjectInCell bullet = **new** BulletForTest();  
  
 **cell**.addObject(tank);  
 **boolean** result = **cell**.addObject(bullet);  
  
 Assertions.*assertTrue*(result);  
 Assertions.*assertTrue*(**cell**.getObjects().contains(tank));  
 Assertions.*assertTrue*(**cell**.getObjects().contains(bullet));  
 Assertions.*assertSame*(**cell**, tank.getCell());  
 Assertions.*assertSame*(**cell**, bullet.getCell());  
 }  
  
 @Test  
 **public void** addObject\_addingInCellContainsObjectWithWrongCollision(){  
 ObjectInCell tank = **new** Tank();  
 ObjectInCell wall = **new** Wall();  
  
 **cell**.addObject(tank);  
 **boolean** result = **cell**.addObject(wall);  
  
 Assertions.*assertFalse*(result);  
 Assertions.*assertTrue*(**cell**.getObjects().contains(tank));  
 Assertions.*assertFalse*(**cell**.getObjects().contains(wall));  
 Assertions.*assertSame*(**cell**, tank.getCell());  
 Assertions.*assertNotSame*(**cell**, wall.getCell());  
 }  
  
 @Test  
 **public void** addObject\_addingInCellContainsSameObject(){  
 ObjectInCell tank = **new** Tank();  
  
 **cell**.addObject(tank);  
 **boolean** result = **cell**.addObject(tank);  
  
 Assertions.*assertFalse*(result);  
 Assertions.*assertTrue*(**cell**.getObjects().contains(tank));  
 Assertions.*assertSame*(**cell**, tank.getCell());  
 Assertions.*assertEquals*(1, **cell**.getObjects().size());  
 }  
  
 @Test  
 **public void** takeObject\_takingFromCellContainsThisObject(){  
 ObjectInCell object = **new** Tank();  
 **cell**.addObject(object);  
  
 ObjectInCell actualObject = **cell**.takeObject(object);  
  
 Assertions.*assertSame*(object, actualObject);  
 Assertions.*assertFalse*(**cell**.getObjects().contains(object));  
 Assertions.*assertEquals*(0, **cell**.getObjects().size());  
 Assertions.*assertNull*(object.getCell());  
 }  
  
 @Test  
 **public void** takeObject\_takingFromEmptyCell(){  
 ObjectInCell object = **new** Tank();  
  
 ObjectInCell actualObject = **cell**.takeObject(object);  
  
 Assertions.*assertNull*(actualObject);  
 Assertions.*assertFalse*(**cell**.getObjects().contains(object));  
 Assertions.*assertEquals*(0, **cell**.getObjects().size());  
 }  
  
 @Test  
 **public void** takeObject\_takingFromCellNotContainsThisObject(){  
 ObjectInCell tank = **new** Tank();  
 ObjectInCell wall = **new** Wall();  
  
 **cell**.addObject(tank);  
  
 ObjectInCell actualObject = **cell**.takeObject(wall);  
  
 Assertions.*assertNull*(actualObject);  
 Assertions.*assertTrue*(**cell**.getObjects().contains(tank));  
 Assertions.*assertEquals*(1, **cell**.getObjects().size());  
 Assertions.*assertNull*(wall.getCell());  
 }  
  
 @Test  
 **public void** setNeighbour\_northNeighbour(){  
 Cell neighbour = **new** Cell(**new** Position(0, -1));  
  
 **cell**.setNeighbour(neighbour);  
  
 Assertions.*assertSame*(neighbour, **cell**.getNeighbour(Direction.***NORTH***));  
 Assertions.*assertSame*(**cell**, neighbour.getNeighbour(Direction.***SOUTH***));  
 }  
  
 @Test  
 **public void** setNeighbour\_southNeighbour(){  
 Cell neighbour = **new** Cell(**new** Position(0, 1));  
  
 **cell**.setNeighbour(neighbour);  
  
 Assertions.*assertSame*(neighbour, **cell**.getNeighbour(Direction.***SOUTH***));  
 Assertions.*assertSame*(**cell**, neighbour.getNeighbour(Direction.***NORTH***));  
 }  
  
 @Test  
 **public void** setNeighbour\_eastNeighbour(){  
 Cell neighbour = **new** Cell(**new** Position(1, 0));  
  
 **cell**.setNeighbour(neighbour);  
  
 Assertions.*assertSame*(neighbour, **cell**.getNeighbour(Direction.***EAST***));  
 Assertions.*assertSame*(**cell**, neighbour.getNeighbour(Direction.***WEST***));  
 }  
  
 @Test  
 **public void** setNeighbour\_westNeighbour(){  
 Cell neighbour = **new** Cell(**new** Position(-1, 0));  
  
 **cell**.setNeighbour(neighbour);  
  
 Assertions.*assertSame*(neighbour, **cell**.getNeighbour(Direction.***WEST***));  
 Assertions.*assertSame*(**cell**, neighbour.getNeighbour(Direction.***EAST***));  
 }  
  
 @Test  
 **public void** setNeighbour\_FourNeighbours(){  
 Cell northNeighbour = **new** Cell(**new** Position(0, -1));  
 Cell southNeighbour = **new** Cell(**new** Position(0, 1));  
 Cell eastNeighbour = **new** Cell(**new** Position(1, 0));  
 Cell westNeighbour = **new** Cell(**new** Position(-1, 0));  
  
 **cell**.setNeighbour(northNeighbour);  
 **cell**.setNeighbour(southNeighbour);  
 **cell**.setNeighbour(eastNeighbour);  
 **cell**.setNeighbour(westNeighbour);  
  
 Assertions.*assertSame*(northNeighbour, **cell**.getNeighbour(Direction.***NORTH***));  
 Assertions.*assertSame*(southNeighbour, **cell**.getNeighbour(Direction.***SOUTH***));  
 Assertions.*assertSame*(eastNeighbour, **cell**.getNeighbour(Direction.***EAST***));  
 Assertions.*assertSame*(westNeighbour, **cell**.getNeighbour(Direction.***WEST***));  
 }  
  
 @Test  
 **public void** setNeighbour\_setSameCellAsNeighbour(){  
 Throwable exception = Assertions.*assertThrows*(RuntimeException.**class**,  
 () -> **cell**.setNeighbour(**cell**));  
  
 Assertions.*assertEquals*(**"Illegal neighbour"**, exception.getMessage());  
 }  
  
 @Test  
 **public void** setNeighbour\_setIllegalNeighbour(){  
 Cell neighbour = **new** Cell(**new** Position(1, 1));  
  
 Throwable exception = Assertions.*assertThrows*(RuntimeException.**class**,  
 () -> **cell**.setNeighbour(neighbour));  
  
 Assertions.*assertEquals*(**"Illegal neighbour"**, exception.getMessage());  
 }  
}

**public class** FieldTest {  
  
 @Test  
 **public void** constructor\_fieldCreation(){  
 Environment environment = **new** EnvironmentForTest();  
 Field field = **new** Field(environment);  
  
 **for** (**int** i = 0; i < environment.fieldSize().height(); i++){  
 **for** (**int** j = 0; j < environment.fieldSize().width(); j++){  
 Assertions.*assertNotNull*(field.getCell(**new** Position(j, i)));  
 }  
 }  
  
 Assertions.*assertEquals*(2, field.getTanks().size());  
 Assertions.*assertNotNull*(field.getTanks().get(0));  
 Assertions.*assertNotNull*(field.getTanks().get(1));  
 }  
}

**public class** GameTest {  
  
 **private** ArrayList<Game.State> **gameStates** = **new** ArrayList<>();  
  
 **private** ArrayList<ObjectInCellEvent> **objectInCellEvents** = **new** ArrayList<>();  
  
 **private** Game **game**;  
  
 **private class** GameListener **implements** IGameEventListener{  
  
 @Override  
 **public void** onObjectChanged(ObjectInCellEvent event) {  
 **objectInCellEvents**.add(event);  
 }  
  
 @Override  
 **public void** onGameStateChanged(Game.State state) {  
 **gameStates**.add(state);  
 }  
 }  
  
 **private void** waitGame(){  
 **try** {  
 **long** sleepTime = 2000L;  
 Thread.*sleep*(sleepTime);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
  
 @BeforeEach  
 **public void** testConfiguration(){  
 **gameStates**.clear();  
 **objectInCellEvents**.clear();  
  
 **game** = **new** Game(**new** EnvironmentForTest());  
 **game**.addListener(**new** GameListener());  
 **game**.start();  
 }  
  
 @Test  
 **public void** tankRotation\_success(){  
 Tank activeTank = **game**.activeTank();  
 activeTank.rotate(Direction.***WEST***);  
  
 ArrayList<ObjectInCellEvent> expectedObjectEvents = **new** ArrayList<>();  
 expectedObjectEvents.add(**new** ObjectInCellEvent(activeTank, ObjectInCellEvent.EventType.***MOVING***));  
  
 Assertions.*assertEquals*(expectedObjectEvents.size(), **objectInCellEvents**.size());  
 **for** (**int** i = 0; i < expectedObjectEvents.size(); i++){  
 Assertions.*assertEquals*(expectedObjectEvents.get(i).getObject().getClass(), **objectInCellEvents**.get(i).getObject().getClass());  
 Assertions.*assertEquals*(expectedObjectEvents.get(i).getType(), **objectInCellEvents**.get(i).getType());  
 }  
  
 Assertions.*assertEquals*(0, **gameStates**.size());  
 }  
  
 @Test  
 **public void** tankRotation\_notSuccess(){  
 Tank activeTank = **game**.activeTank();  
 activeTank.rotate(Direction.***NORTH***);  
  
 Assertions.*assertEquals*(0, **objectInCellEvents**.size());  
 Assertions.*assertEquals*(0, **gameStates**.size());  
 }  
  
 @Test  
 **public void** tankMoving\_success(){  
 Tank activeTank = **game**.activeTank();  
 activeTank.rotate(Direction.***EAST***);  
 activeTank.move();  
  
 ArrayList<ObjectInCellEvent> expectedObjectEvents = **new** ArrayList<>();  
 expectedObjectEvents.add(**new** ObjectInCellEvent(activeTank, ObjectInCellEvent.EventType.***MOVING***));  
 expectedObjectEvents.add(**new** ObjectInCellEvent(activeTank, ObjectInCellEvent.EventType.***MOVING***));  
  
 ArrayList<Game.State> expectedGameStates = **new** ArrayList<>();  
 expectedGameStates.add(Game.State.***GAME\_IS\_ON***);  
  
 Assertions.*assertEquals*(expectedObjectEvents.size(), **objectInCellEvents**.size());  
 **for** (**int** i = 0; i < expectedObjectEvents.size(); i++){  
 Assertions.*assertEquals*(expectedObjectEvents.get(i).getObject().getClass(), **objectInCellEvents**.get(i).getObject().getClass());  
 Assertions.*assertEquals*(expectedObjectEvents.get(i).getType(), **objectInCellEvents**.get(i).getType());  
 }  
  
 Assertions.*assertEquals*(expectedGameStates.size(), **gameStates**.size());  
 Assertions.*assertEquals*(expectedGameStates.get(0), **gameStates**.get(0));  
 Assertions.*assertNotSame*(**game**.activeTank(), activeTank);  
 }  
  
 @Test  
 **public void** tankMoving\_notSuccess(){  
 Tank activeTank = **game**.activeTank();  
 activeTank.move();  
  
 Assertions.*assertEquals*(0, **objectInCellEvents**.size());  
 Assertions.*assertEquals*(0, **gameStates**.size());  
 }  
  
 @Test  
 **public void** tankSkipStep(){  
 Tank activeTank = **game**.activeTank();  
 activeTank.pass();  
  
 ArrayList<Game.State> expectedGameStates = **new** ArrayList<>();  
 expectedGameStates.add(Game.State.***GAME\_IS\_ON***);  
  
 Assertions.*assertEquals*(0, **objectInCellEvents**.size());  
  
 Assertions.*assertEquals*(expectedGameStates.size(), **gameStates**.size());  
 Assertions.*assertEquals*(expectedGameStates.get(0), **gameStates**.get(0));  
 Assertions.*assertNotSame*(**game**.activeTank(), activeTank);  
 }  
  
 @Test  
 **public void** tankShootingEnemyTank(){  
 Bullet testBullet = **new** Bullet(Direction.***NORTH***, **new** Cell(**new** Position(0, 0)));  
 Tank tank1 = **game**.activeTank();  
 tank1.rotate(Direction.***EAST***);  
 tank1.move();  
  
 Tank tank2 = **game**.activeTank();  
 tank2.rotate(Direction.***WEST***);  
 tank2.move();  
  
 tank1.rotate(Direction.***SOUTH***);  
 tank1.shoot();  
 waitGame();  
  
 ArrayList<ObjectInCellEvent> expectedObjectEvents = **new** ArrayList<>();  
 expectedObjectEvents.add(**new** ObjectInCellEvent(tank1, ObjectInCellEvent.EventType.***MOVING***));  
 expectedObjectEvents.add(**new** ObjectInCellEvent(tank1, ObjectInCellEvent.EventType.***MOVING***));  
 expectedObjectEvents.add(**new** ObjectInCellEvent(tank2, ObjectInCellEvent.EventType.***MOVING***));  
 expectedObjectEvents.add(**new** ObjectInCellEvent(tank2, ObjectInCellEvent.EventType.***MOVING***));  
 expectedObjectEvents.add(**new** ObjectInCellEvent(tank1, ObjectInCellEvent.EventType.***MOVING***));  
 expectedObjectEvents.add(**new** ObjectInCellEvent(testBullet, ObjectInCellEvent.EventType.***MOVING***));  
 expectedObjectEvents.add(**new** ObjectInCellEvent(testBullet, ObjectInCellEvent.EventType.***MOVING***));  
 expectedObjectEvents.add(**new** ObjectInCellEvent(tank2, ObjectInCellEvent.EventType.***DAMAGED***));  
 expectedObjectEvents.add(**new** ObjectInCellEvent(testBullet, ObjectInCellEvent.EventType.***DESTROYING***));  
  
 ArrayList<Game.State> expectedGameStates = **new** ArrayList<>();  
 expectedGameStates.add(Game.State.***GAME\_IS\_ON***);  
 expectedGameStates.add(Game.State.***GAME\_IS\_ON***);  
 expectedGameStates.add(Game.State.***GAME\_IS\_ON***);  
  
 Assertions.*assertEquals*(expectedObjectEvents.size(), **objectInCellEvents**.size());  
 **for** (**int** i = 0; i < expectedObjectEvents.size(); i++){  
 Assertions.*assertEquals*(expectedObjectEvents.get(i).getObject().getClass(), **objectInCellEvents**.get(i).getObject().getClass());  
 Assertions.*assertEquals*(expectedObjectEvents.get(i).getType(), **objectInCellEvents**.get(i).getType());  
 }  
  
 Assertions.*assertEquals*(expectedGameStates.size(), **gameStates**.size());  
 **for** (**int** i = 0; i < expectedGameStates.size(); i++) {  
 Assertions.*assertEquals*(expectedGameStates.get(i), **gameStates**.get(i));  
 }  
 }  
  
 @Test  
 **public void** gameFinished\_tankDestroyEnemyHeadquarters(){  
 Bullet testBullet = **new** Bullet(Direction.***NORTH***, **new** Cell(**new** Position(0, 0)));  
 Tank activeTank = **game**.activeTank();  
 activeTank.rotate(Direction.***EAST***);  
 activeTank.shoot();  
 waitGame();  
  
 ArrayList<ObjectInCellEvent> expectedObjectEvents = **new** ArrayList<>();  
 expectedObjectEvents.add(**new** ObjectInCellEvent(activeTank, ObjectInCellEvent.EventType.***MOVING***));  
 expectedObjectEvents.add(**new** ObjectInCellEvent(testBullet, ObjectInCellEvent.EventType.***MOVING***));  
 expectedObjectEvents.add(**new** ObjectInCellEvent(testBullet, ObjectInCellEvent.EventType.***MOVING***));  
 expectedObjectEvents.add(**new** ObjectInCellEvent(**new** Headquarters(), ObjectInCellEvent.EventType.***DESTROYING***));  
 expectedObjectEvents.add(**new** ObjectInCellEvent(testBullet, ObjectInCellEvent.EventType.***DESTROYING***));  
  
 ArrayList<Game.State> expectedGameStates = **new** ArrayList<>();  
 expectedGameStates.add(Game.State.***WINNER\_FOUND***);  
  
 Assertions.*assertEquals*(expectedObjectEvents.size(), **objectInCellEvents**.size());  
 **for** (**int** i = 0; i < expectedObjectEvents.size(); i++){  
 Assertions.*assertEquals*(expectedObjectEvents.get(i).getObject().getClass(), **objectInCellEvents**.get(i).getObject().getClass());  
 Assertions.*assertEquals*(expectedObjectEvents.get(i).getType(), **objectInCellEvents**.get(i).getType());  
 }  
  
 Assertions.*assertEquals*(expectedGameStates.size(), **gameStates**.size());  
 Assertions.*assertEquals*(expectedGameStates.get(0), **gameStates**.get(0));  
 Assertions.*assertSame*(**game**.winner(), activeTank);  
 }  
}

**public class** HeadquartersTest {  
  
 **private** Headquarters **headquarters**;  
  
 @BeforeEach  
 **public void** testConfiguration(){  
 **headquarters** = **new** Headquarters();  
 }  
  
 @Test  
 **public void** constructor\_headquartersCreation(){  
 **headquarters** = **new** Headquarters();  
  
 Assertions.*assertNull*(**headquarters**.getCell());  
 Assertions.*assertNull*(**headquarters**.getTank());  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithTank(){  
 Tank tank = **new** Tank();  
  
 **boolean** result = **headquarters**.canFaceWith(tank);  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithObstacle(){  
 Obstacle obstacle = **new** Wall();  
  
 **boolean** result = **headquarters**.canFaceWith(obstacle);  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithSameObject(){  
 **boolean** result = **headquarters**.canFaceWith(**headquarters**);  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithBullet(){  
 Bullet bullet = **new** BulletForTest();  
  
 **boolean** result = **headquarters**.canFaceWith(bullet);  
  
 Assertions.*assertTrue*(result);  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithTank(){  
 Tank tank = **new** Tank();  
  
 Assertions.*assertThrows*(IllegalArgumentException.**class**, () ->  
 **headquarters**.faceWith(tank));  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithObstacle(){  
 Obstacle wall = **new** Wall();  
  
 Assertions.*assertThrows*(IllegalArgumentException.**class**, () ->  
 **headquarters**.faceWith(wall));  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithBullet(){  
 Bullet bullet = **new** BulletForTest();  
 ObjectInCellEvent[] actualEvents = {**null**};  
  
 **headquarters**.addListener(**new** IObjectInCellEventListener() {  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 Assertions.*assertSame*(**headquarters**, event.getObject());  
 Assertions.*assertEquals*(ObjectInCellEvent.EventType.***DESTROYING***, event.getType());  
 actualEvents[0] = event;  
 }  
 });  
  
 ObjectInCellEvent expectedEvent = **new** ObjectInCellEvent(**headquarters**, ObjectInCellEvent.EventType.***DESTROYING***);  
  
 **headquarters**.faceWith(bullet);  
  
 Assertions.*assertNull*(**headquarters**.getCell());  
 Assertions.*assertNotNull*(actualEvents[0]);  
 Assertions.*assertSame*(expectedEvent.getObject(), actualEvents[0].getObject());  
 }  
  
 @Test  
 **public void** update\_destroying(){  
 Cell cell = **new** Cell(**new** Position(0, 0));  
 Bullet bullet = **new** BulletForTest();  
 ObjectInCellEvent[] actualEvents = {**null**};  
  
 **headquarters**.addListener(**new** IObjectInCellEventListener() {  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 Assertions.*assertSame*(**headquarters**, event.getObject());  
 Assertions.*assertEquals*(ObjectInCellEvent.EventType.***DESTROYING***, event.getType());  
 Assertions.*assertSame*(cell, **headquarters**.getCell());  
 actualEvents[0] = event;  
 }  
 });  
  
 ObjectInCellEvent expectedEvent = **new** ObjectInCellEvent(**headquarters**, ObjectInCellEvent.EventType.***DESTROYING***);  
  
 cell.addObject(**headquarters**);  
 cell.addObject(bullet);  
 **headquarters**.update();  
  
 Assertions.*assertNull*(**headquarters**.getCell());  
 Assertions.*assertNotNull*(actualEvents[0]);  
 Assertions.*assertSame*(expectedEvent.getObject(), actualEvents[0].getObject());  
 }  
  
 @Test  
 **public void** setTank\_addingTank(){  
 Tank tank = **new** Tank();  
  
 **headquarters**.setTank(tank);  
  
 Assertions.*assertSame*(tank, **headquarters**.getTank());  
 }  
}

**class** PositionTest {  
  
 **private** Position **position** = **new** Position(0, 0);  
  
 @Test  
 **public void** positionCreation(){  
 **position** = **new** Position(1, 2);  
  
 Assertions.*assertEquals*(1, **position**.getX());  
 Assertions.*assertEquals*(2, **position**.getY());  
 }  
  
 @Test  
 **public void** shift\_shiftToNorthOnPositiveDelta(){  
 Position newPosition = **position**.shift(Direction.***NORTH***, 1);  
  
 Position expectedPosition = **new** Position(0, -1);  
  
 Assertions.*assertNotNull*(newPosition);  
  
 Assertions.*assertEquals*(expectedPosition.getX(), newPosition.getX());  
 Assertions.*assertEquals*(expectedPosition.getY(), newPosition.getY());  
  
 Assertions.*assertEquals*(0, **position**.getX());  
 Assertions.*assertEquals*(0, **position**.getY());  
 }  
  
 @Test  
 **public void** shift\_shiftToSouthOnPositiveDelta(){  
 Position newPosition = **position**.shift(Direction.***SOUTH***, 1);  
  
 Position expectedPosition = **new** Position(0, 1);  
  
 Assertions.*assertNotNull*(newPosition);  
  
 Assertions.*assertEquals*(expectedPosition.getX(), newPosition.getX());  
 Assertions.*assertEquals*(expectedPosition.getY(), newPosition.getY());  
  
 Assertions.*assertEquals*(0, **position**.getX());  
 Assertions.*assertEquals*(0, **position**.getY());  
 }  
  
 @Test  
 **public void** shift\_shiftToEastOnPositiveDelta(){  
 Position newPosition = **position**.shift(Direction.***EAST***, 1);  
  
 Position expectedPosition = **new** Position(1, 0);  
  
 Assertions.*assertNotNull*(newPosition);  
  
 Assertions.*assertEquals*(expectedPosition.getX(), newPosition.getX());  
 Assertions.*assertEquals*(expectedPosition.getY(), newPosition.getY());  
  
 Assertions.*assertEquals*(0, **position**.getX());  
 Assertions.*assertEquals*(0, **position**.getY());  
 }  
  
 @Test  
 **public void** shift\_shiftToWestOnPositiveDelta(){  
 Position newPosition = **position**.shift(Direction.***WEST***, 1);  
  
 Position expectedPosition = **new** Position(-1, 0);  
  
 Assertions.*assertNotNull*(newPosition);  
  
 Assertions.*assertEquals*(expectedPosition.getX(), newPosition.getX());  
 Assertions.*assertEquals*(expectedPosition.getY(), newPosition.getY());  
  
 Assertions.*assertEquals*(0, **position**.getX());  
 Assertions.*assertEquals*(0, **position**.getY());  
 }  
  
 @Test  
 **public void** shift\_shiftToNorthOnNegativeDelta(){  
 Position newPosition = **position**.shift(Direction.***NORTH***, -1);  
  
 Position expectedPosition = **new** Position(0, 1);  
  
 Assertions.*assertNotNull*(newPosition);  
  
 Assertions.*assertEquals*(expectedPosition.getX(), newPosition.getX());  
 Assertions.*assertEquals*(expectedPosition.getY(), newPosition.getY());  
  
 Assertions.*assertEquals*(0, **position**.getX());  
 Assertions.*assertEquals*(0, **position**.getY());  
 }  
  
 @Test  
 **public void** shift\_shiftToSouthOnNegativeDelta(){  
 Position newPosition = **position**.shift(Direction.***SOUTH***, -1);  
  
 Position expectedPosition = **new** Position(0, -1);  
  
 Assertions.*assertNotNull*(newPosition);  
  
 Assertions.*assertEquals*(expectedPosition.getX(), newPosition.getX());  
 Assertions.*assertEquals*(expectedPosition.getY(), newPosition.getY());  
  
 Assertions.*assertEquals*(0, **position**.getX());  
 Assertions.*assertEquals*(0, **position**.getY());  
 }  
  
 @Test  
 **public void** shift\_shiftToEastOnNegativeDelta(){  
 Position newPosition = **position**.shift(Direction.***EAST***, -1);  
  
 Position expectedPosition = **new** Position(-1, 0);  
  
 Assertions.*assertNotNull*(newPosition);  
  
 Assertions.*assertEquals*(expectedPosition.getX(), newPosition.getX());  
 Assertions.*assertEquals*(expectedPosition.getY(), newPosition.getY());  
  
 Assertions.*assertEquals*(0, **position**.getX());  
 Assertions.*assertEquals*(0, **position**.getY());  
 }  
  
 @Test  
 **public void** shift\_shiftToWestOnNegativeDelta(){  
 Position newPosition = **position**.shift(Direction.***WEST***, -1);  
  
 Position expectedPosition = **new** Position(1, 0);  
  
 Assertions.*assertNotNull*(newPosition);  
  
 Assertions.*assertEquals*(expectedPosition.getX(), newPosition.getX());  
 Assertions.*assertEquals*(expectedPosition.getY(), newPosition.getY());  
  
 Assertions.*assertEquals*(0, **position**.getX());  
 Assertions.*assertEquals*(0, **position**.getY());  
 }  
  
 @Test  
 **public void** shift\_shiftToZeroDelta(){  
 Position newPosition = **position**.shift(Direction.***NORTH***, 0);  
  
 Position expectedPosition = **new** Position(0, 0);  
  
 Assertions.*assertNotNull*(newPosition);  
  
 Assertions.*assertEquals*(expectedPosition.getX(), newPosition.getX());  
 Assertions.*assertEquals*(expectedPosition.getY(), newPosition.getY());  
  
 Assertions.*assertEquals*(0, **position**.getX());  
 Assertions.*assertEquals*(0, **position**.getY());  
 }  
}

**public class** TankTest {  
  
 **private** Tank **tank**;  
  
 @BeforeEach  
 **public void** testConfiguration(){  
 **tank** = **new** Tank();  
 }  
  
 @Test  
 **public void** constructor\_tankCreation(){  
 **tank** = **new** Tank();  
  
 Assertions.*assertNull*(**tank**.getCell());  
 Assertions.*assertNull*(**tank**.getHeadquarters());  
 Assertions.*assertEquals*(Direction.***NORTH***, **tank**.getDirection());  
 Assertions.*assertEquals*(3, **tank**.getHealth());  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithObstacle(){  
 Obstacle obstacle = **new** Wall();  
  
 **boolean** result = **tank**.canFaceWith(obstacle);  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithSameObject(){  
 **boolean** result = **tank**.canFaceWith(**tank**);  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithBullet(){  
 Bullet bullet = **new** BulletForTest();  
  
 **boolean** result = **tank**.canFaceWith(bullet);  
  
 Assertions.*assertTrue*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithTank(){  
 Tank otherTank = **new** Tank();  
  
 Assertions.*assertThrows*(IllegalArgumentException.**class**, () ->  
 **tank**.faceWith(otherTank));  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithObstacle(){  
 Obstacle wall = **new** Wall();  
  
 Assertions.*assertThrows*(IllegalArgumentException.**class**, () ->  
 **tank**.faceWith(wall));  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithBullet() {  
 Bullet bullet = **new** BulletForTest();  
 ObjectInCellEvent[] actualEvents = {**null**};  
  
 **tank**.addListener(**new** ITankEventListener() {  
 @Override  
 **public void** onTankMoved(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onTankShot(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onTankSkipStep(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 Assertions.*assertSame*(**tank**, event.getObject());  
 Assertions.*assertEquals*(ObjectInCellEvent.EventType.***DAMAGED***, event.getType());  
 actualEvents[0] = event;  
 }  
 });  
  
 ObjectInCellEvent expectedEvent = **new** ObjectInCellEvent(**tank**, ObjectInCellEvent.EventType.***DAMAGED***);  
  
 **tank**.faceWith(bullet);  
  
 Assertions.*assertNotNull*(actualEvents[0]);  
 Assertions.*assertSame*(expectedEvent.getObject(), actualEvents[0].getObject());  
 }  
  
 @Test  
 **public void** rotate\_rotateTank(){  
 ObjectInCellEvent[] actualEvents = {**null**};  
  
 **tank**.addListener(**new** ITankEventListener() {  
 @Override  
 **public void** onTankMoved(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onTankShot(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onTankSkipStep(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 Assertions.*assertSame*(**tank**, event.getObject());  
 Assertions.*assertEquals*(ObjectInCellEvent.EventType.***MOVING***, event.getType());  
 actualEvents[0] = event;  
 }  
 });  
  
 **tank**.rotate(Direction.***SOUTH***);  
  
 Assertions.*assertNotNull*(actualEvents[0]);  
 Assertions.*assertEquals*(Direction.***SOUTH***, **tank**.getDirection());  
 }  
  
 @Test  
 **public void** rotate\_rotateTankOnSameDirection(){  
 **tank**.addListener(**new** ITankEventListener() {  
 @Override  
 **public void** onTankMoved(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onTankShot(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onTankSkipStep(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
 });  
  
 **tank**.rotate(Direction.***NORTH***);  
  
 Assertions.*assertEquals*(Direction.***NORTH***, **tank**.getDirection());  
 }  
  
 @Test  
 **public void** move\_movingInEmptyCell(){  
 Cell startCell = **new** Cell(**new** Position(0, 1));  
 Cell targetCell = **new** Cell(**new** Position(0, 0));  
 startCell.setNeighbour(targetCell);  
 ObjectInCellEvent[] actualEvents = {**null**};  
  
 startCell.addObject(**tank**);  
 **tank**.rotate(Direction.***NORTH***);  
  
 **tank**.addListener(**new** ITankEventListener() {  
 @Override  
 **public void** onTankMoved(ObjectInCellEvent event) {  
 Assertions.*assertSame*(**tank**, event.getObject());  
 Assertions.*assertEquals*(ObjectInCellEvent.EventType.***MOVING***, event.getType());  
 Assertions.*assertSame*(targetCell, **tank**.getCell());  
 actualEvents[0] = event;  
 }  
  
 @Override  
 **public void** onTankShot(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onTankSkipStep(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
 });  
  
 **boolean** result = **tank**.move();  
  
 Assertions.*assertTrue*(result);  
 Assertions.*assertNotNull*(actualEvents[0]);  
 }  
  
 @Test  
 **public void** move\_movingInNotEmptyCell(){  
 Cell startCell = **new** Cell(**new** Position(0, 0));  
 Cell targetCell = **new** Cell(**new** Position(0, 1));  
 startCell.setNeighbour(targetCell);  
 Wall wall = **new** Wall();  
  
 startCell.addObject(**tank**);  
 targetCell.addObject(wall);  
 **tank**.rotate(Direction.***NORTH***);  
  
 **tank**.addListener(**new** ITankEventListener() {  
 @Override  
 **public void** onTankMoved(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onTankShot(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onTankSkipStep(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
 });  
  
 **boolean** result = **tank**.move();  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** move\_noCellInMovingDirection(){  
 Cell startCell = **new** Cell(**new** Position(0, 0));  
  
 startCell.addObject(**tank**);  
 **tank**.rotate(Direction.***NORTH***);  
  
 **tank**.addListener(**new** ITankEventListener() {  
 @Override  
 **public void** onTankMoved(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onTankShot(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onTankSkipStep(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
 });  
  
 **boolean** result = **tank**.move();  
  
 Assertions.*assertFalse*(result);  
 Assertions.*assertSame*(startCell, **tank**.getCell());  
 }  
  
 @Test  
 **public void** shoot\_shootInEmptyCell(){  
 Cell startCell = **new** Cell(**new** Position(0, 1));  
 Cell targetCell = **new** Cell(**new** Position(0, 0));  
 startCell.setNeighbour(targetCell);  
 ObjectInCellEvent[] actualEvents = {**null**};  
  
 startCell.addObject(**tank**);  
 **tank**.rotate(Direction.***NORTH***);  
  
 **tank**.addListener(**new** ITankEventListener() {  
 @Override  
 **public void** onTankMoved(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onTankShot(ObjectInCellEvent event) {  
 Assertions.*assertEquals*(Bullet.**class**, event.getObject().getClass());  
 Assertions.*assertEquals*(ObjectInCellEvent.EventType.***MOVING***, event.getType());  
 actualEvents[0] = event;  
 }  
  
 @Override  
 **public void** onTankSkipStep(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
 });  
  
 **boolean** result = **tank**.shoot();  
  
 Assertions.*assertTrue*(result);  
 Assertions.*assertNotNull*(actualEvents[0]);  
 }  
  
 @Test  
 **public void** shoot\_noCellInShootingDirection(){  
 Cell startCell = **new** Cell(**new** Position(0, 0));  
  
 startCell.addObject(**tank**);  
 **tank**.rotate(Direction.***NORTH***);  
  
 **tank**.addListener(**new** ITankEventListener() {  
 @Override  
 **public void** onTankMoved(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onTankShot(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onTankSkipStep(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
 });  
  
 **boolean** result = **tank**.shoot();  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** pass\_skipStep(){  
 ObjectInCellEvent[] actualEvents = {**null**};  
  
 **tank**.addListener(**new** ITankEventListener() {  
 @Override  
 **public void** onTankMoved(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onTankShot(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onTankSkipStep(ObjectInCellEvent event) {  
 Assertions.*assertSame*(**tank**, event.getObject());  
 actualEvents[0] = event;  
 }  
  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
 });  
  
 **tank**.pass();  
  
 Assertions.*assertNotNull*(actualEvents[0]);  
 }  
  
 @Test  
 **public void** update\_getDamage() {  
 Cell cell = **new** Cell(**new** Position(0, 0));  
 Bullet bullet = **new** BulletForTest();  
 ObjectInCellEvent[] actualEvents = {**null**};  
  
 **tank**.addListener(**new** ITankEventListener() {  
 @Override  
 **public void** onTankMoved(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onTankShot(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onTankSkipStep(ObjectInCellEvent event) {  
 **throw new** RuntimeException(**"Unexpected event"**);  
 }  
  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 Assertions.*assertSame*(**tank**, event.getObject());  
 Assertions.*assertEquals*(ObjectInCellEvent.EventType.***DAMAGED***, event.getType());  
 Assertions.*assertSame*(cell, **tank**.getCell());  
 actualEvents[0] = event;  
 }  
 });  
  
 ObjectInCellEvent expectedEvent = **new** ObjectInCellEvent(**tank**, ObjectInCellEvent.EventType.***DAMAGED***);  
 **int** expectedHealth = **tank**.getHealth() - 1;  
  
 cell.addObject(**tank**);  
 cell.addObject(bullet);  
 **tank**.update();  
  
 Assertions.*assertNotNull*(actualEvents[0]);  
 Assertions.*assertEquals*(expectedHealth, **tank**.getHealth());  
 Assertions.*assertSame*(expectedEvent.getObject(), actualEvents[0].getObject());  
 }  
}

**public class** WallTest {  
  
 **private** Wall **wall**;  
  
 @BeforeEach  
 **public void** testConfiguration(){  
 **wall** = **new** Wall();  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithTank(){  
 Tank tank = **new** Tank();  
  
 **boolean** result = **wall**.canFaceWith(tank);  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithObstacle(){  
 Obstacle obstacle = **new** Wall();  
  
 **boolean** result = **wall**.canFaceWith(obstacle);  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithSameObject(){  
 **boolean** result = **wall**.canFaceWith(**wall**);  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithBullet(){  
 Bullet bullet = **new** BulletForTest();  
  
 **boolean** result = **wall**.canFaceWith(bullet);  
  
 Assertions.*assertTrue*(result);  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithTank(){  
 Tank tank = **new** Tank();  
  
 Assertions.*assertThrows*(IllegalArgumentException.**class**, () ->  
 **wall**.faceWith(tank));  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithObstacle(){  
 Obstacle wall = **new** Wall();  
  
 Assertions.*assertThrows*(IllegalArgumentException.**class**, () ->  
 wall.faceWith(wall));  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithBullet() {  
 Bullet bullet = **new** BulletForTest();  
 ObjectInCellEvent[] actualEvents = {**null**};  
  
 **wall**.addListener(**new** IObjectInCellEventListener() {  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 Assertions.*assertSame*(**wall**, event.getObject());  
 Assertions.*assertEquals*(ObjectInCellEvent.EventType.***DESTROYING***, event.getType());  
 actualEvents[0] = event;  
 }  
 });  
  
 ObjectInCellEvent expectedEvent = **new** ObjectInCellEvent(**wall**, ObjectInCellEvent.EventType.***DESTROYING***);  
  
 **wall**.faceWith(bullet);  
  
 Assertions.*assertNull*(**wall**.getCell());  
 Assertions.*assertNotNull*(actualEvents[0]);  
 Assertions.*assertSame*(expectedEvent.getObject(), actualEvents[0].getObject());  
 }  
  
 @Test  
 **public void** update\_destroying() {  
 Cell cell = **new** Cell(**new** Position(0, 0));  
 Bullet bullet = **new** BulletForTest();  
 ObjectInCellEvent[] actualEvents = {**null**};  
  
 **wall**.addListener(**new** IObjectInCellEventListener() {  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 Assertions.*assertSame*(**wall**, event.getObject());  
 Assertions.*assertEquals*(ObjectInCellEvent.EventType.***DESTROYING***, event.getType());  
 Assertions.*assertSame*(cell, **wall**.getCell());  
 actualEvents[0] = event;  
 }  
 });  
  
 ObjectInCellEvent expectedEvent = **new** ObjectInCellEvent(**wall**, ObjectInCellEvent.EventType.***DESTROYING***);  
  
 cell.addObject(**wall**);  
 cell.addObject(bullet);  
 **wall**.update();  
  
 Assertions.*assertNull*(**wall**.getCell());  
 Assertions.*assertNotNull*(actualEvents[0]);  
 Assertions.*assertSame*(expectedEvent.getObject(), actualEvents[0].getObject());  
 }  
}

**public class** WaterTest {  
  
 **private** Water **water**;  
  
 @BeforeEach  
 **public void** testConfiguration(){  
 **water** = **new** Water();  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithTank(){  
 Tank tank = **new** Tank();  
  
 **boolean** result = **water**.canFaceWith(tank);  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithObstacle(){  
 Obstacle obstacle = **new** Wall();  
  
 **boolean** result = **water**.canFaceWith(obstacle);  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithSameObject(){  
 **boolean** result = **water**.canFaceWith(**water**);  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithBullet(){  
 Bullet bullet = **new** BulletForTest();  
  
 **boolean** result = **water**.canFaceWith(bullet);  
  
 Assertions.*assertTrue*(result);  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithTank(){  
 Tank tank = **new** Tank();  
  
 Assertions.*assertThrows*(IllegalArgumentException.**class**, () ->  
 **water**.faceWith(tank));  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithObstacle(){  
 Obstacle wall = **new** Wall();  
  
 Assertions.*assertThrows*(IllegalArgumentException.**class**, () ->  
 **water**.faceWith(wall));  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithBullet(){  
 Bullet bullet = **new** BulletForTest();  
 ObjectInCellEvent[] actualEvents = {**null**};  
  
 **water**.addListener(**new** IObjectInCellEventListener() {  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 actualEvents[0] = event;  
 }  
 });  
  
 **water**.faceWith(bullet);  
  
 Assertions.*assertNull*(actualEvents[0]);  
 }  
}

# 4 Вторая итерация разработки

## 4.1 Формулировка модифицированного варианта задания.

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

**Вариативность**: предусмотреть возможность создания других "препятствий", имеющих различное поведение при попадании снаряда: собственное разрушение, разрушение других объектов, возможность перемещения танка и снаряда.

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

* препятствие "заросли" - танки могут заезжать в заросли, это делает их невидимыми для противника - снаряд пролетает насквозь зарослей, не уничтожая танки и заросли.
* препятствие "бочка мазута" - при попадании она взрывается и наносит урон рядом стоящим объектам (возможна цепная реакция, которая должна анимироваться)

## 4.2 Функциональные требования (сценарии)

**1) Сценарий** «Играть»:

1. **По указанию пользователя** Игра стартует.

2. **По указанию Игры** Поле создаёт Ячейки и формирует из них себя.

3. **По указанию Игры** Окружение создает и размещает на Поле Препятствия, Танки и Штабы.

4. **В ответ на запрос Игры** Поле сообщает о Танках, которые

находятся на нём.

5. Игра случайным образом выбирает один из Танков и делает его

активным.

6. **Делать**

6.1. **По указанию пользователя** активному Танку задается направление хода.

6.2. **По указанию пользователя** Танк перемещается в соседнюю Ячейку в заданном направлении и сообщает об этом Игре.

6.3. Игра делает активным другой Танк.

**Пока** оба Танка живы и ни один Штаб не уничтожен.

7. Игра считает победителем выживший Танк с целым Штабом.

8. **Сценарий завершается.**

**2) Дочерний сценарий** «Окружение создает и размещает на Поле Препятствия, Танки и Штабы».

1. Окружение создает и помещает Штабы в Ячейки на противоположных сторонах Поля.

2. Окружение создает Препятствия и помешает их в свободные Ячейки произвольным образом.

3. Окружение создает Танки и помещает их в свободные Ячейки.

4. **Сценарий завершается.**

3) **Дочерний сценарий** «Танку задается направление хода».

1. **Пока** пользователь не определится с направление хода **делать**

1.1. **По указанию пользователя** Танк меняет направление хода на указанное.

2. **Сценарий завершается.**

4) **Дочерний сценарий** «Танк перемещается в соседнюю Ячейку в заданном направлении».

1. **Пока** ход не будет завершен **делать**

1.1. **По указанию пользователя** Танк решает переместиться в соседнюю Ячейку в направлении хода.

1.2. **В ответ на запрос Танка** Ячейка сообщает о соседней Ячейке в направлении хода.

1.3. **По указанию Танка** соседняя Ячейка помещает его в себя, так как в ней больше нет объектов.

1.4. **По указанию Танка** Ячейка извлекает его из себя.

1.5. Танк обновляет счетчик перезарядки орудия.

1.6. Танк сообщает Игре о завершении хода.

2. **Сценарий завершается.**

4.1) **Альтернативный сценарий** «В соседней Ячейке находится объект, препятствующий перемещению Танка». Сценарий выполняется с пункта 1.3 сценария 4.

1. **В ответ на запрос Танка** соседняя Ячейка сообщает о том, что она не может поместить в себя Танк, т. к. в ней находится объект, с которым Танк не может сталкиваться (Танк противника, Штаб, Препятствие).

2. Танк остается в текущей Ячейке.

3. **Сценарий продолжается с п. 1 сценария 4.**

4.2) Альтернативный сценарий «В соседней Ячейке находятся Заросли». Сценарий выполняется с пункта 1.3 сценария 4.

1. **По указанию Танка** соседняя Ячейка помещает его в себя, и сталкивает с Зарослями.

2. **При столкновении с Танком** Заросли перемещают его в себя.

3. **Сценарий продолжается с п. 1.4 сценария 4.**

4.3) **Альтернативный сценарий** «Танк находится на краю поля». Сценарий выполняется с пункта п. 1.2 сценария 4.

1. **В ответ на запрос Танка** Ячейка сообщает, что у нее нет соседа в направлении хода.

2. Танк решает, что не он не может передвигаться в указанном направлении, и остается в текущей Ячейке.

3. **Сценарий продолжается с п. 1 сценария 4.**

4.4) **Альтернативный сценарий** «Танк совершает выстрел». Сценарий выполняется с п. 1.1. сценария 4.

1. **По указанию пользователя** Танк решает совершить выстрел в направлении хода.

2. **В ответ на запрос Танка** Ячейка сообщает о соседней Ячейке в направлении хода.

3. **Если** оружие Танка не на перезарядке **делать**

3.1. Танк размещает Снаряд в соседней Ячейке и сообщает ему направление выстрела.

3.2. Снаряд движется по полю в направлении выстрела.

3.3. Танк начинает перезарядку орудия.

4. **Сценарий продолжается с п. 1.7 сценария 4.**

4.5) **Альтернативный сценарий** «Пользователь решает пропустить ход». Сценарий выполняется с п. 1.1. сценария 4.

1. **По указанию пользователя** Танк завершает ход.

2. **Сценарий продолжается с п. 1 сценария 4.**

5) Дополнительный сценарий «Снаряд движется по полю в направлении выстрела».

1. **Делать**

1.1. **В ответ на запрос Снаряда** Ячейка сообщает ему о своем соседе в направлении движения.

1.2. **По указанию Снаряда** текущая Ячейка извлекает его из себя.

1.3**. По указанию Снаряда** соседняя Ячейка помещает его в себя и сталкивает его с Объектами, которые уже находится в ячейке.

**Пока** Снаряд находится на Поле.

2. **Сценарий завершается**.

5.1) Альтернативный сценарий «Снаряд достигает края Поля». Сценарий выполняется с п. 1.1. сценария 5.

1. **По запросу Снаряда** текущая Ячейка сообщает о том, что у нее нет соседа в направлении выстрела.

2. **По запросу Снаряда** текущая Ячейка извлекает его из себя.

3. **Сценарий завершается.**

6) Дополнительный сценарий «Поражающий\_Объект сталкивается с другим Объектом».

1. **Если** Объект\_В\_Ячейке является Поражаемым\_Объектом, **делать**.

1.1. **При столкновении с Поражающим\_Объектом** Поражаемый\_Объект получает урон.

1.2. **При столкновении с Поражаемым\_Объектом** Поражающий\_Объект принимает решение уничтожиться.

1.3. **По указанию Поражающего\_Объекта** Ячейка, в которой находится объект, извлекает его из себя.

2. Сценарий завершается.

7) Дополнительный сценарий «Поражаемый\_Объект получает урон».

1. **При получении урона** Поражаемый\_Объект решает уничтожится.

2. **По указанию Поражаемого\_Объекта** Ячейка, в которой он находится, извлекает его из себя.

3. Сценарий завершается.

7.1) Альтернативный сценарий «Танк получает урон». Сценарий выполняется с п. 1. сценария 7.

1. **При получении урона** Танк уменьшает количество своего здоровья на 1.

2. Сценарий завершается.

7.2) Альтернативный сценарий «Бочка\_Мазута получает урон». Сценарий выполняется с п. 1. сценария 7.

1. **При получении урона** Бочка\_Мазута решает взорваться.

2. **В ответ на запрос Бочки\_Мазута** Ячейка, в которой она находится, сообщает ей обо всех ее соседях.

3 **Для** каждого соседа Ячейки **делать**.

3.1. Бочка\_Мазута создает Взрыв.

3.2. **По указанию Бочки\_Мазута** соседняя Ячеек помещает в себя Взрыв и сталкивает его с другими Объектами в ячейке.

3.3. **По указанию Взрыва** соседняя Ячейка извлекает его из себя.

4. **По указанию Бочки\_Мазута** Ячейка извлекает ее из себя.

8) **Дополнительный сценарий** «Игра считает победителем выживший Танк».

1. **По запросу Игры** Поле сообщает ей об одном Танке, который на нем расположен.

2. Игра считает Танк победителем, т. к. его противник уничтожен.

3. **Сценарий завершается.**

8.1) **Альтернативный сценарий** «Игра считает победителем выживший Танк с целым Штабом». Сценарий выполняется с п. 1 сценария 8.

1. **По запросу Игры** Поле сообщает Игре о двух Танках, которые на нем расположены.

2. **По запросу Игры** Танки сообщают ей о своих Штабах.

3. Игра узнает об одном Штабе.

4. Игра считает победителем Танк, Штаб которого цел.

5. **Сценарий завершается.**

8.1.1) **Альтернативный сценарий** «Игра завершается вничью т. к. уничтожены оба штаба». Сценарий выполняется с п. 3 сценария 8.1.

1. Игра не узнает ни об одном Штабе.

2. Игра завершается в ничью, т. к. оба Штаба уничтожены.

3. **Сценарий завершается.**

8.2) **Альтернативный сценарий** «Игра завершается вничью т. к. оба танка уничтожены». Сценарий выполняется с п. 1 сценария 8.

1. **По запросу Игры** Поле не сообщает Игре ни об одном Танке.

2. Игра завершается в ничью, т. к. оба танка уничтожены.

3. **Сценарий завершается.**

9) **Альтернативный сценарий** «Досрочное завершение игры».

Сценарий выполняется в любой точке главного сценария

1. **По указанию пользователя** программа завершается без

определения победителя.

2. **Сценарий завершается.**

## 4.3 Словарь предметной области.

**Игра –** знает о Поле и активном Танке. Инициирует создание и заполнение Поля. Игра определяет следующего активного игрока и окончание игры.

**Поле** – прямоугольная область, состоящая из Ячеек. Знает обо всех Ячейках, из которых состоит, а также о Танках, которые на нем расположены.

**Окружение** – умеет создавать Танки, Штабы, Препятствия и размещать их в Ячейках на Поле.

**Ячейка –** квадратная часть Поля. Знает о четырех соседних Ячейках. На ней может располагаться Танк, Снаряд, Препятствие или Штаб.

**Объект\_В\_Ячейке (Объект)** – игровой объект, который может находиться в ячейки и сталкиваться с другими объектами. Знает о ячейке, в которой находится.

**Столкновение** – процесс взаимодействий двух Объектов при помещении в одну Ячейку.

**Поражаемый\_Объект** – Объект\_В\_Ячейке, который получает урон при столкновении с Поражающим\_Объектом.

**Поражающий\_Объект** – Объект\_В\_Ячейке, который наносит урон Поражаемому\_Объекту при столкновении с ним.

**Снаряд** – Поражающий\_Объект, который движется по Полю, пока не покинет его или не уничтожится. Принадлежит Танку, который его выпускает.

**Танк –** Поражаемый\_Объект, которыйумеет поворачиваться в одном из четырех направлений и однократно перемещаться или стрелять в выбранном направлении, а также пропускать ход. Знает о своем Штабе.

**Штаб –** Поражаемый\_Объект, которыйпривязывается к Танку, также влияет на исход игры**.**

**Препятствие –** прочие игровые Объекты, которые имеют разное поведение при столкновении с Танком и Снарядом.

**Стена** – Препятствие, являющееся Поражаемым\_Объектом.

**Вода** – Непреодолимое Препятствие.

**Заросли** – Препятствие, способное скрывать Танк внутри себя.

**Бочка\_Мазута** – Препятствие, являющееся Поражаемым\_Объектом и создающая Взрывы при получении урона.

**Взрыв** – недвижимый Поражающий\_Объект, всегда уничтожающийся после помещения в Ячейку.

## 4.4 Структура программы на уровне классов

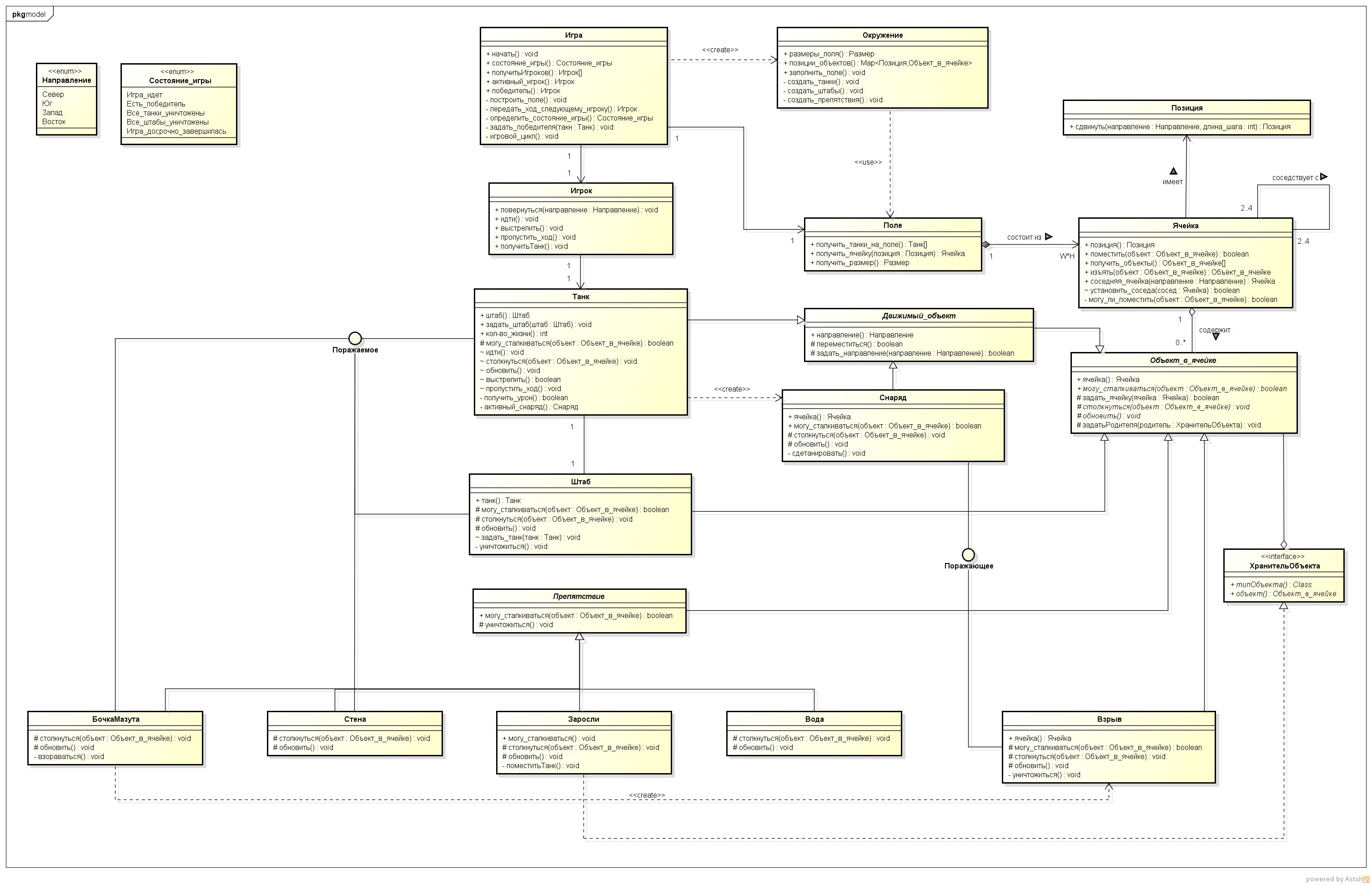


Диаграмма классов вычислительной модели

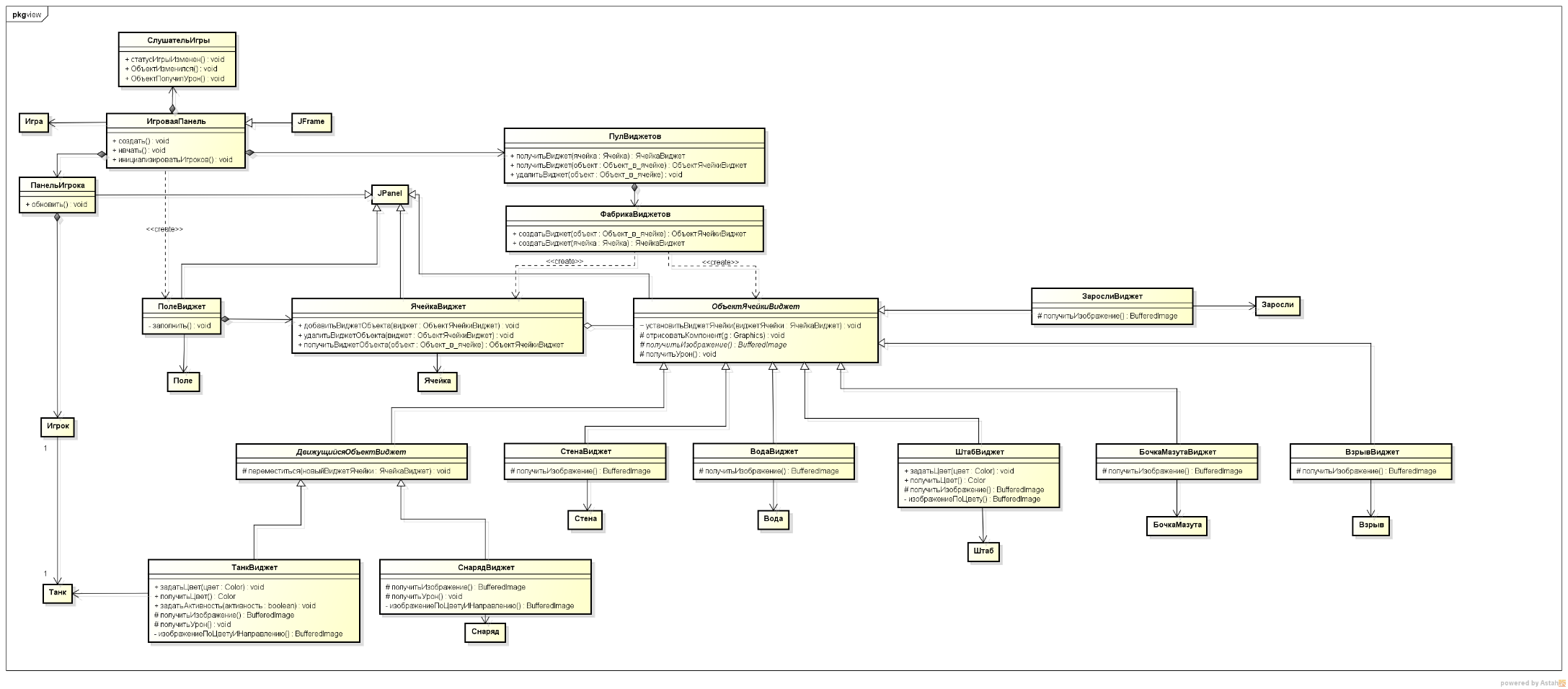
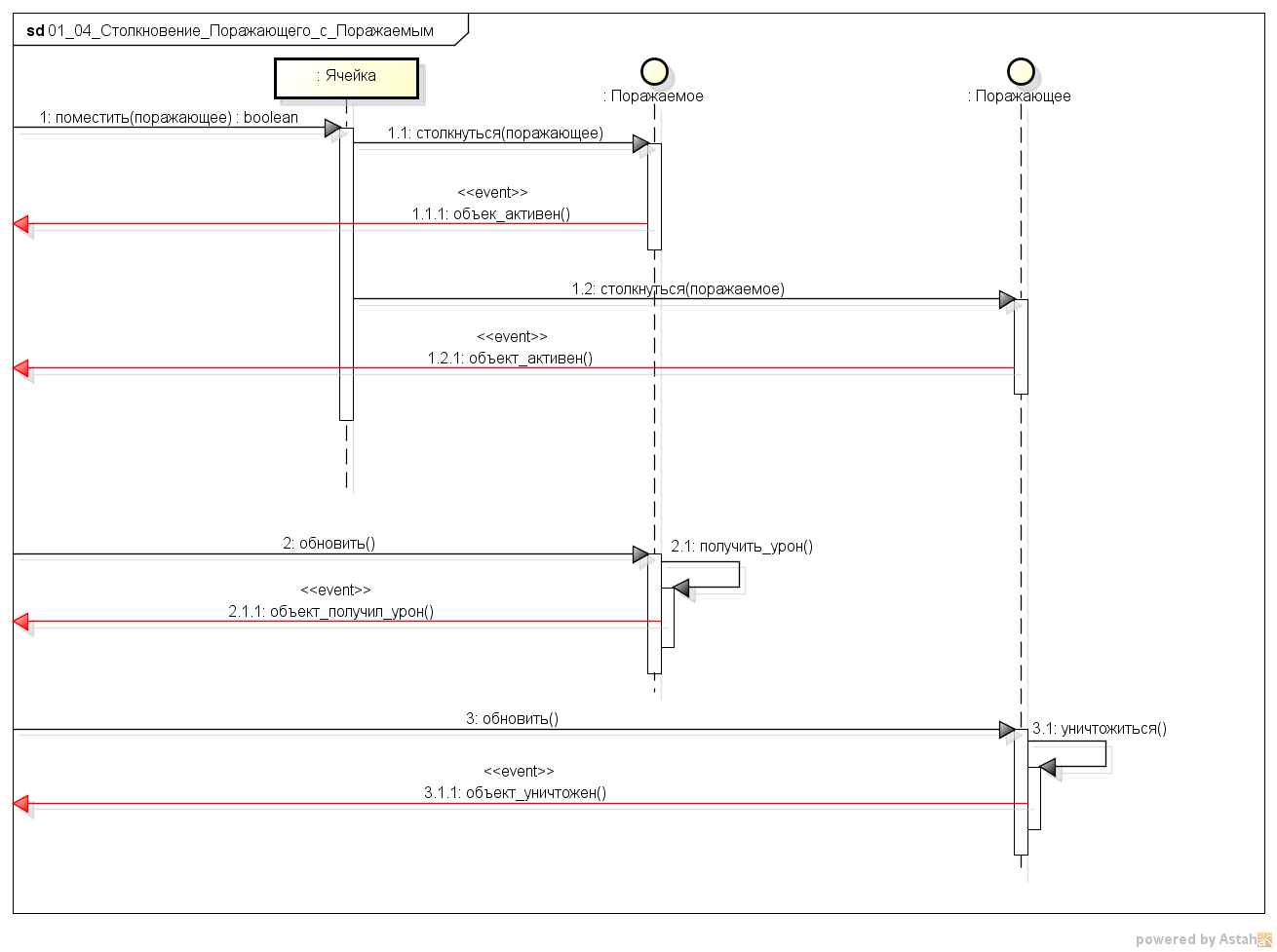
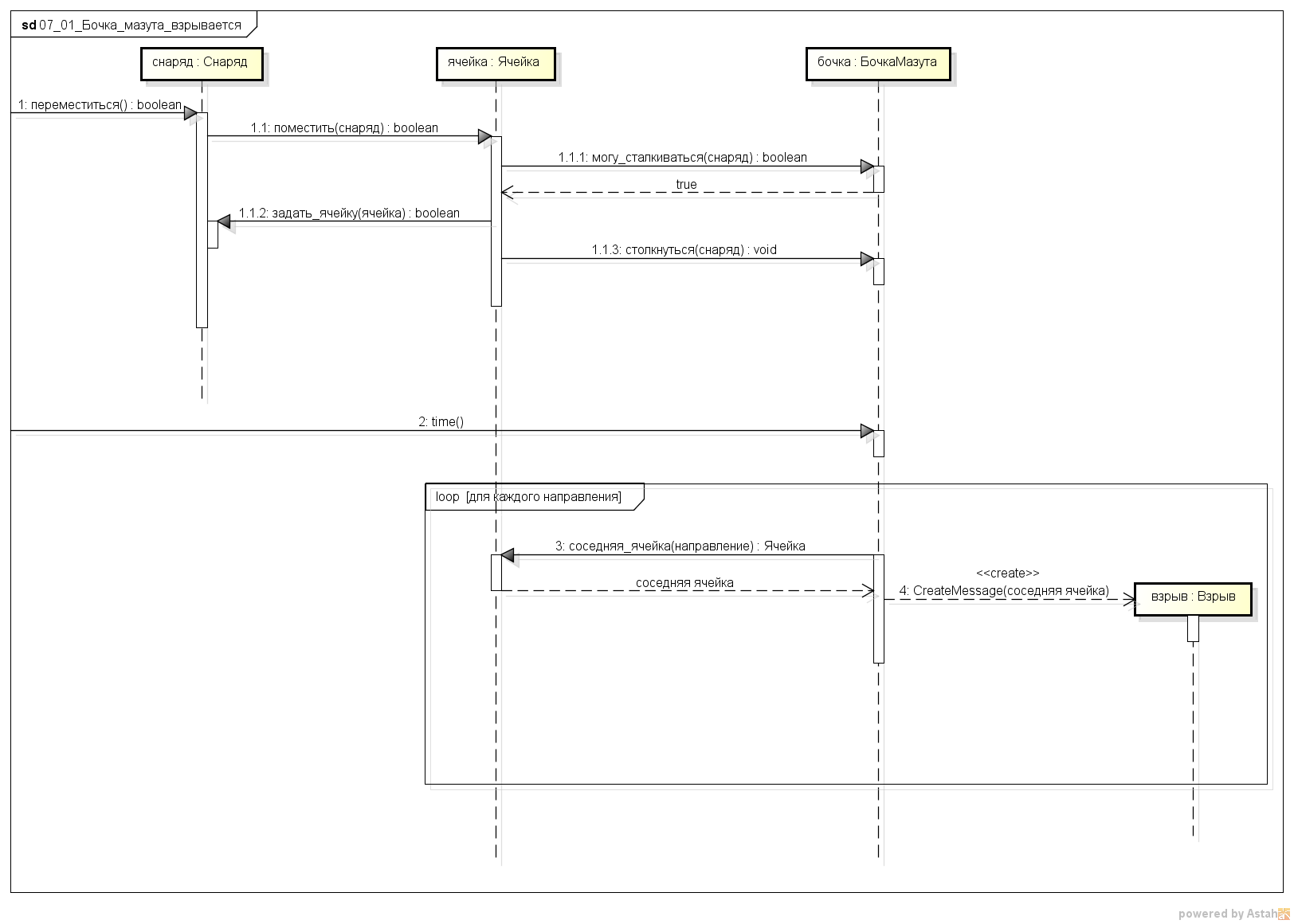


Диаграмма классов представления

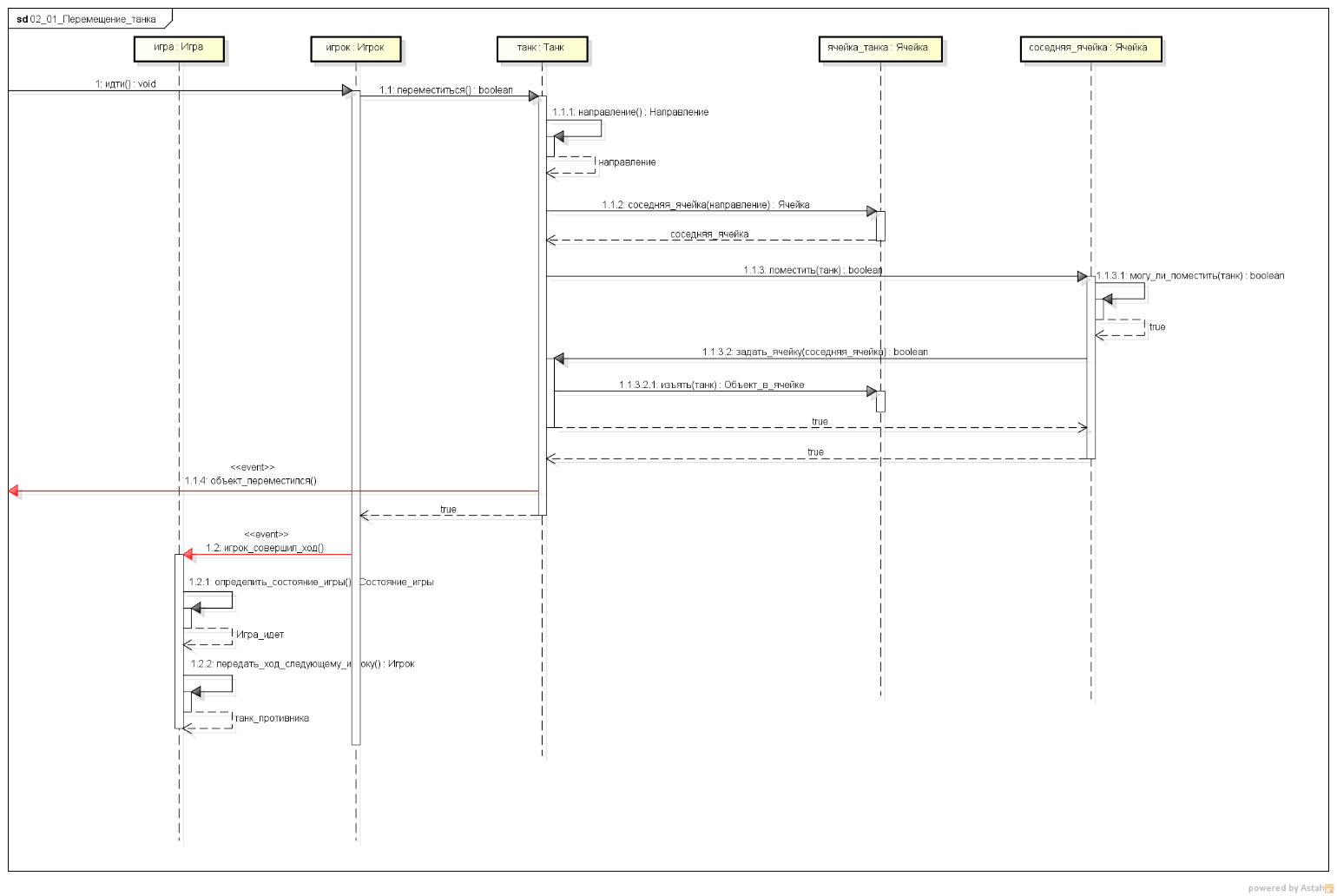
## 4.5 Типовые процессы в программе



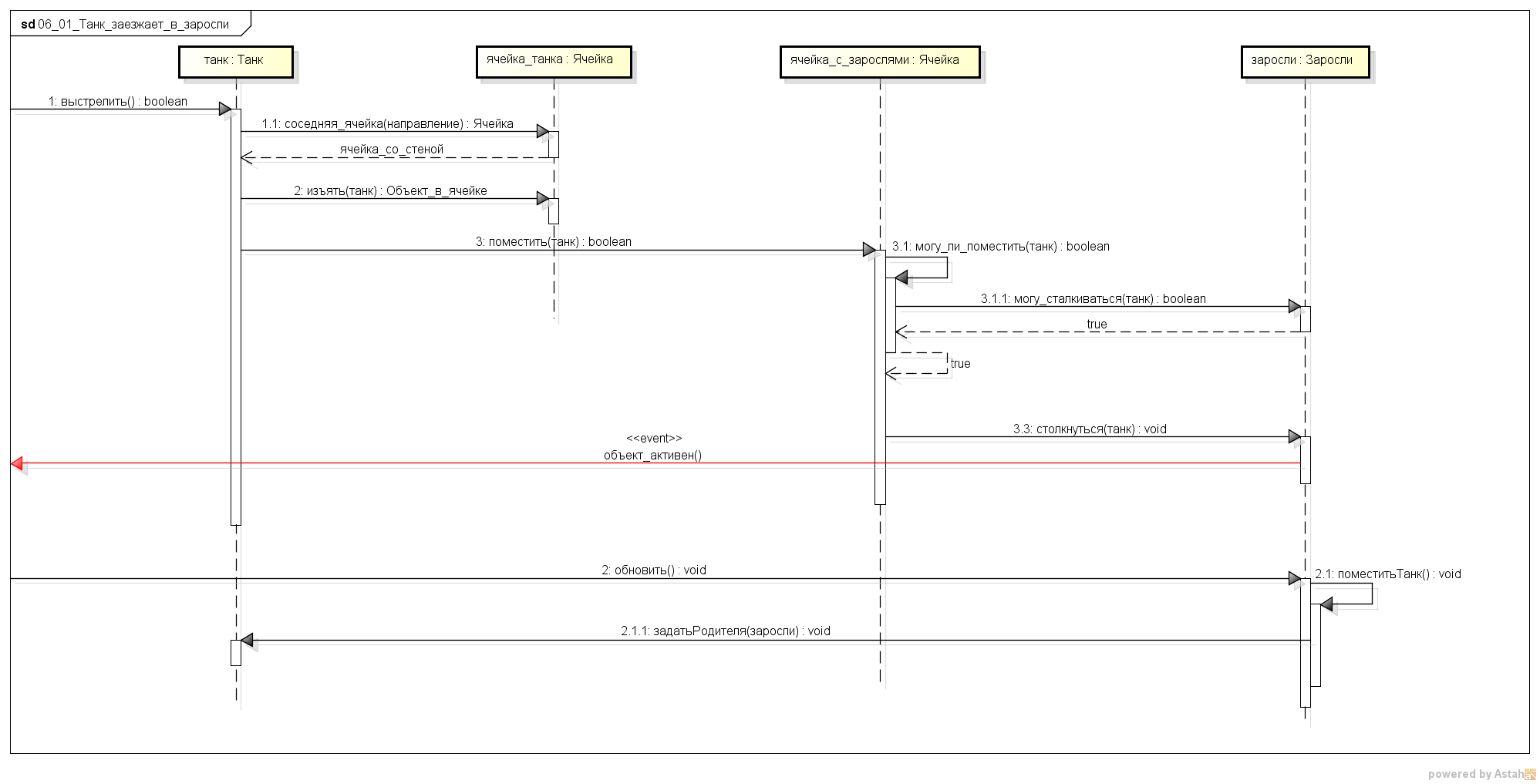
Столкновения поражающего объекта с поражаемым



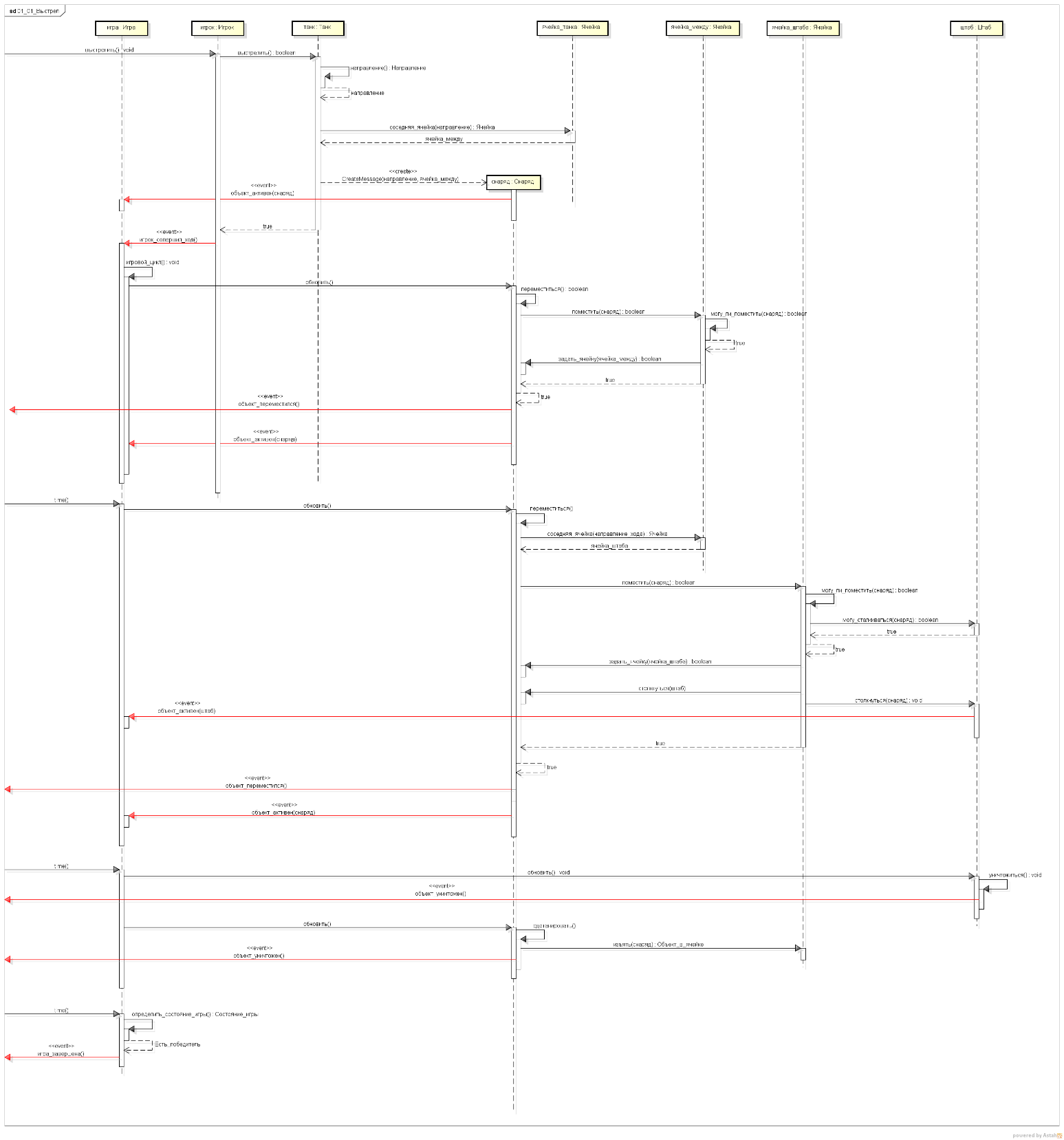
Бочка мазута взрывается



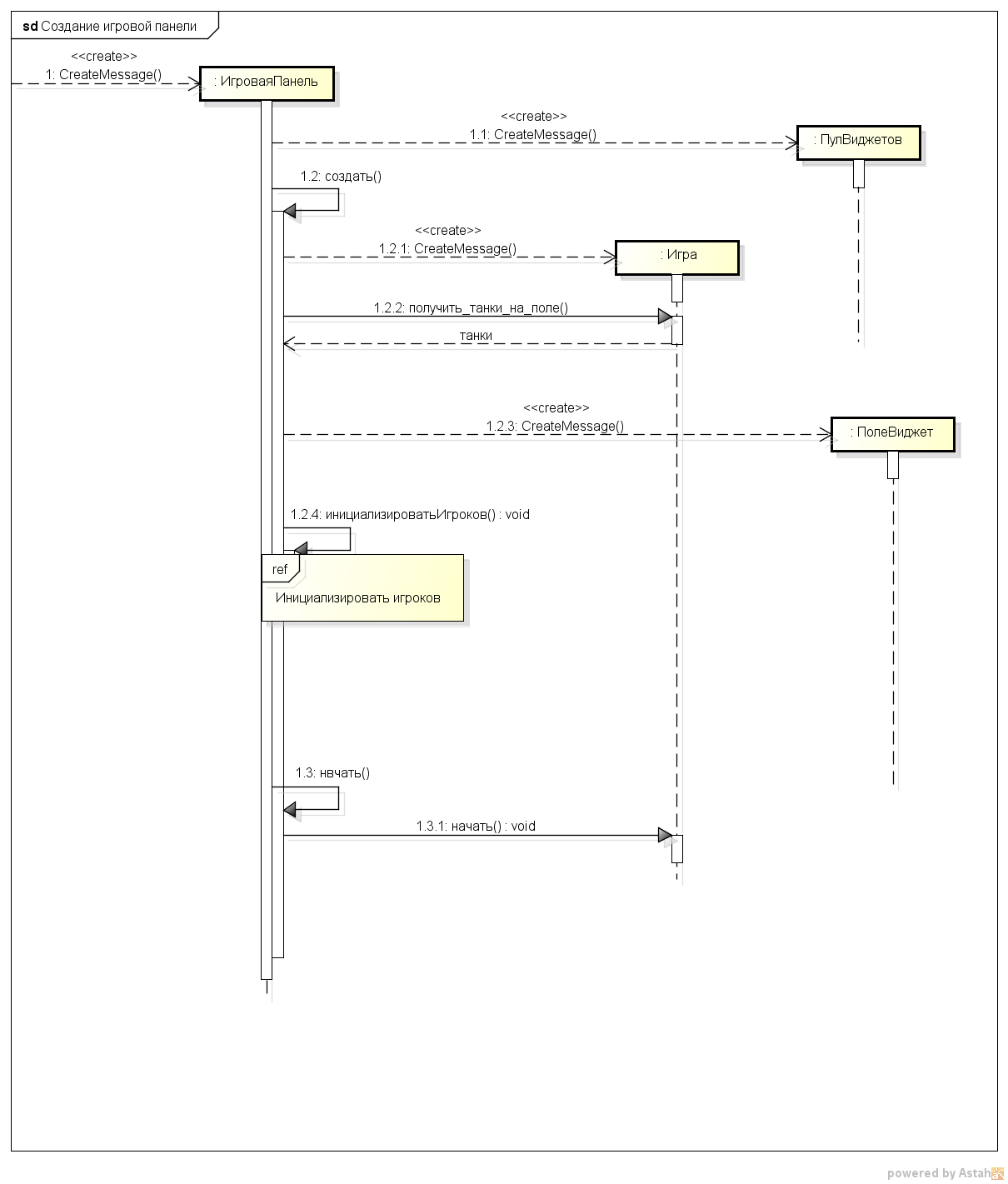
Перемещение танка



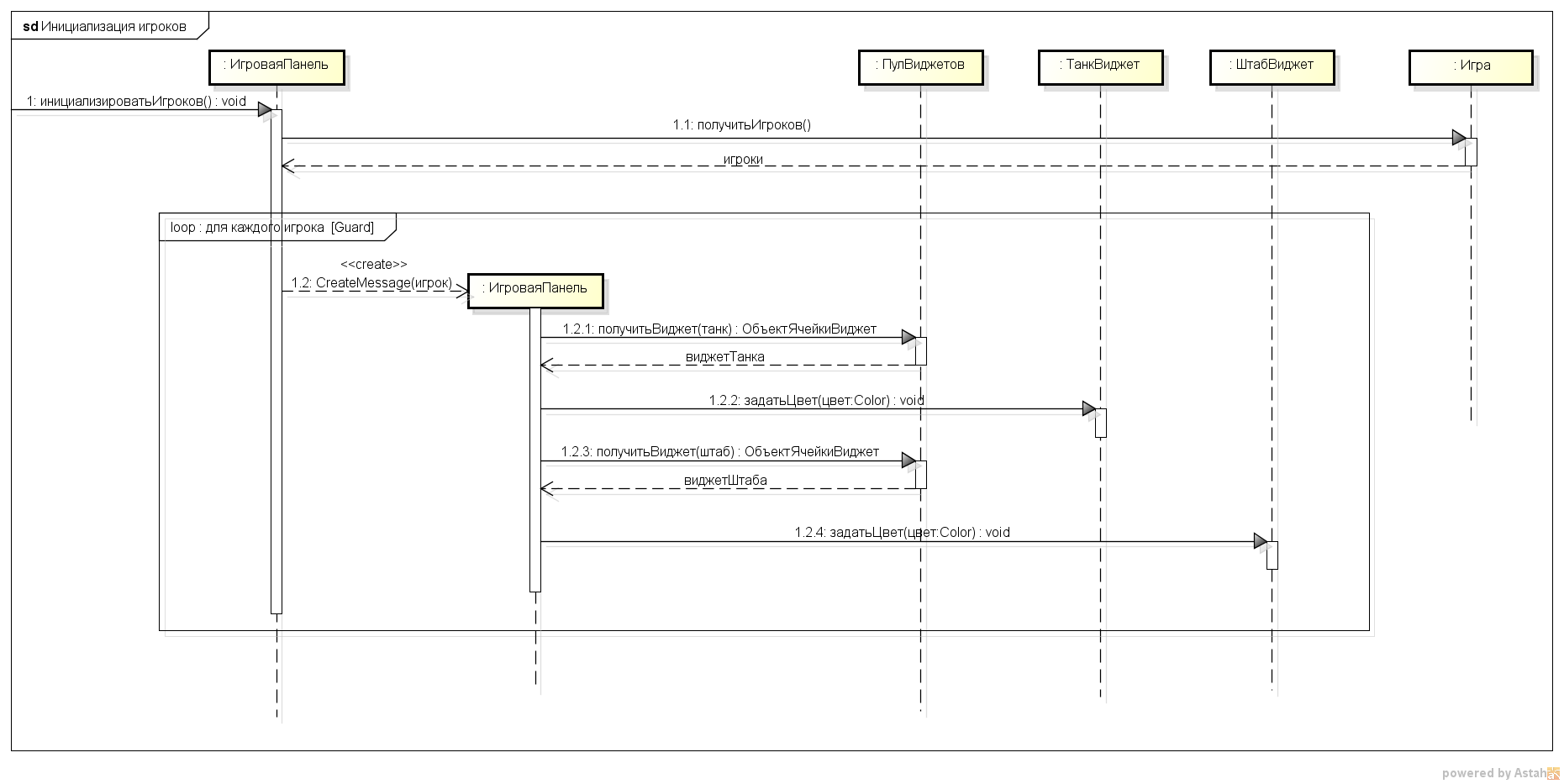
Танк заезжает в заросли



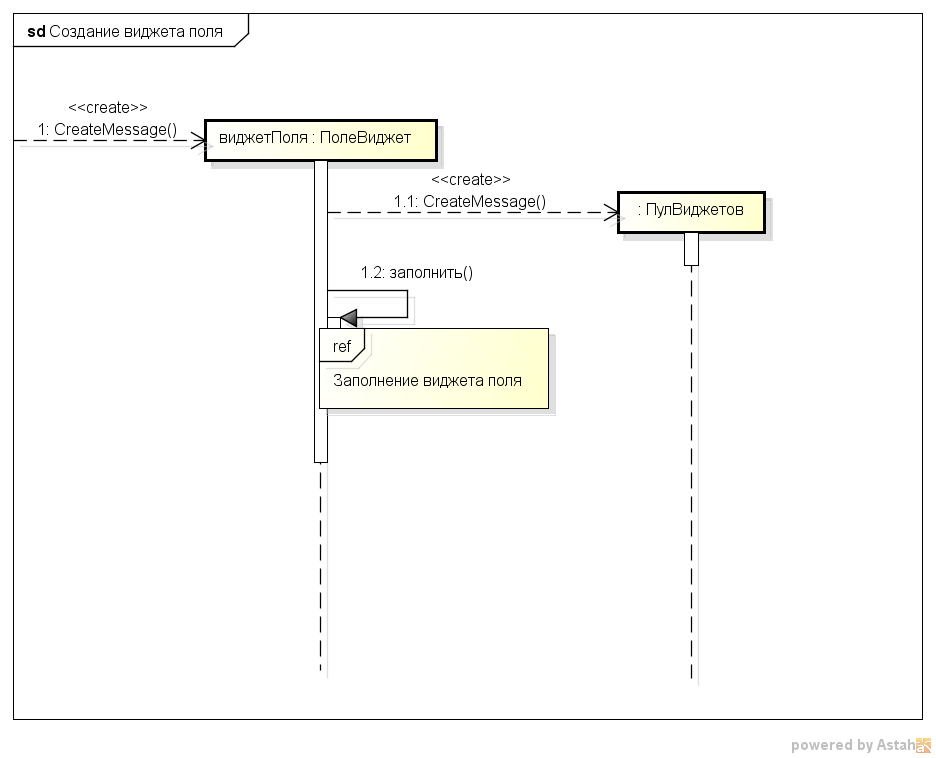
Выстрел



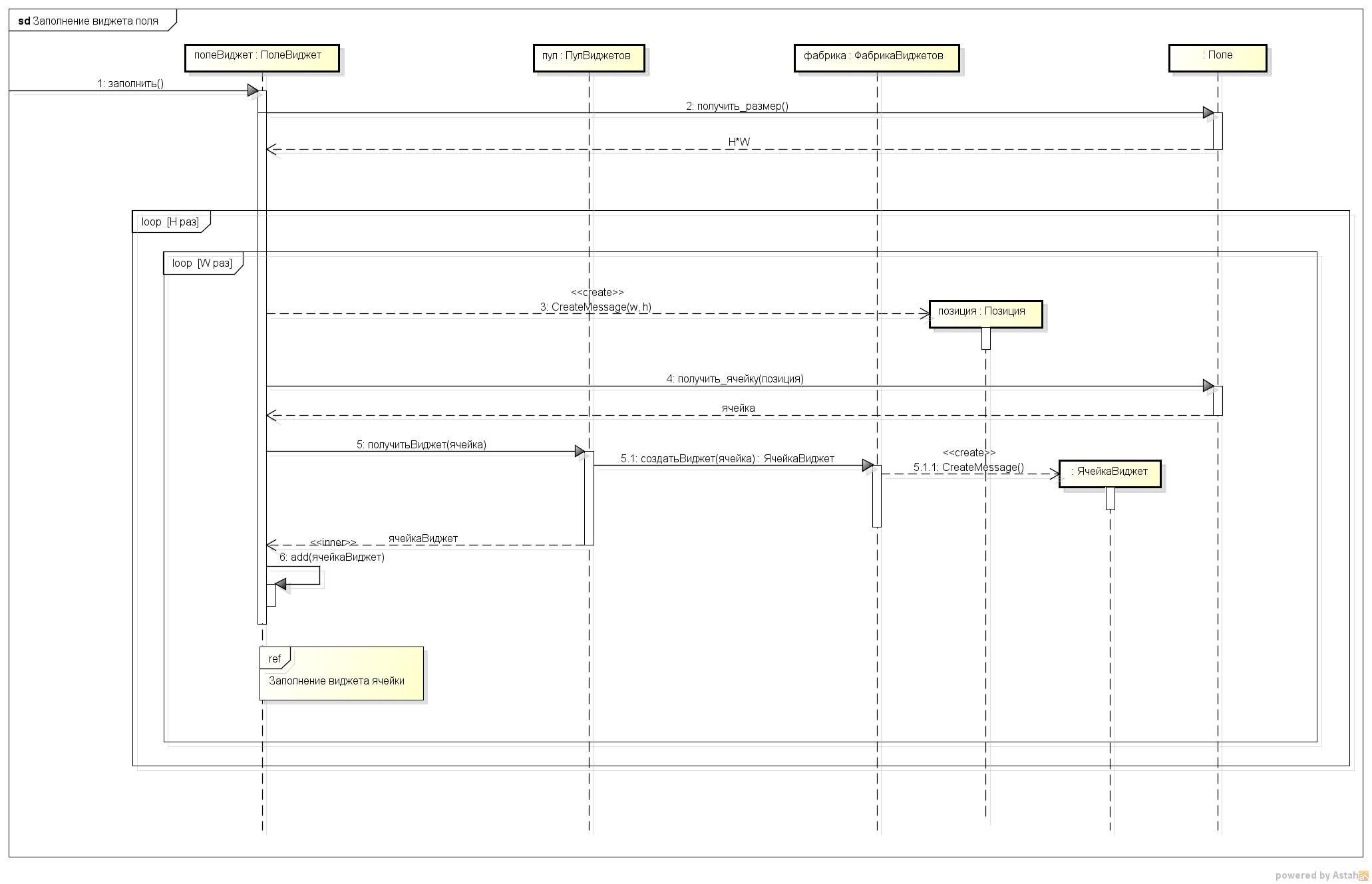
Создание игровой панели



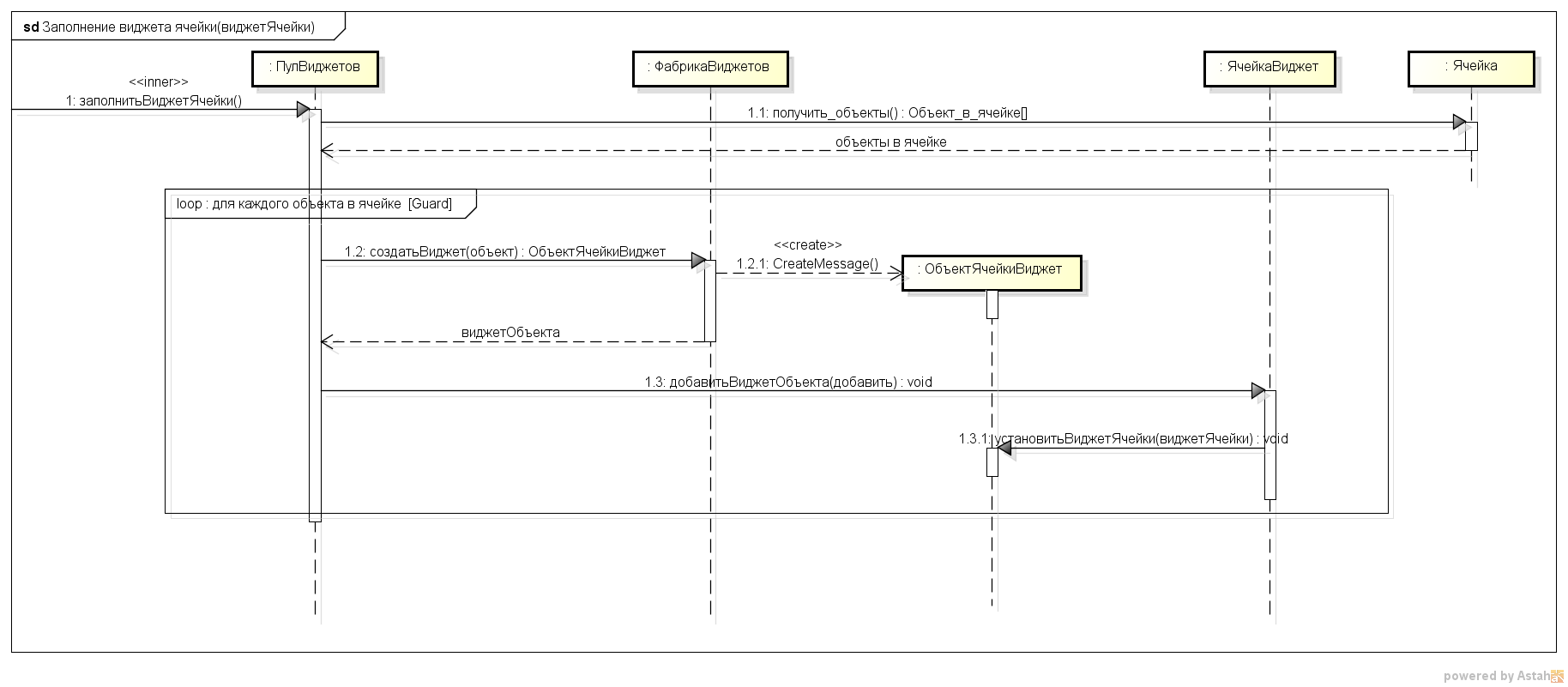
Инициализация игроков



Создание виджета поля



Заполнение виджета поля



Заполнение виджета ячейки

## 4.6 Человеко машинное взаимодействие

Изображение зарослей представлено ниже. Когда танк заезжает в заросли, их изображение накладывается поверх изображения танка.

Изображение выглядит как пиксель, зеленый, снимок экрана

Автоматически созданное описание Изображение выглядит как пиксель, снимок экрана, Красочность, зеленый

Автоматически созданное описание

Изображение зарослей

Изображение бочки с мазутом представлено на рисунке ниже. При взрыве бочки ее изображение меняется на взорванную бочку.

Изображение выглядит как снимок экрана, желтый, пиксель, Прямоугольник

Автоматически созданное описание Изображение выглядит как снимок экрана, пиксель, Красочность, апельсин

Автоматически созданное описание

Изображение бочки с мазутом

Когда бочка взрывается, она создает вокруг себя взрыв, который отображается следующим образом.

Изображение выглядит как снимок экрана, Красочность, пиксель, мозаика

Автоматически созданное описание

Отображение взрыва бочки

## 4.7 Реализация ключевых классов

*/\*\*  
 \* Объект, располагающийся в ячейке на поле  
 \*/***public abstract class** ObjectInCell {  
  
 */\* --------------- Ячейка ---------------------- \*/  
  
 /\*\*  
 \* Ячейка в которой находится объект  
 \*/* **private** Cell **\_cell**;  
  
 **public** Cell getCell(){  
 Cell cell = **\_cell**;  
 **if** (**\_parent** != **null**){  
 cell = **\_parent**.getCell();  
 }  
 **return** cell;  
 }  
  
 **protected void** setCell(Cell cell){  
 **if** (**\_cell** != **null** && **\_cell** != cell){  
 **\_cell**.takeObject(**this**);  
 }  
 **\_parent** = **null**;  
 **\_cell** = cell;  
 }  
  
 **protected void** unsetCell(){  
 setCell(**null**);  
 }  
  
 */\* ------------------- Родитель ------------- \*/  
  
 /\*\*  
 \* Родитель объекта  
 \*/* **private** ObjectKeeper<? **extends** ObjectInCell> **\_parent**;  
  
 **protected void** setParent(ObjectKeeper<? **extends** ObjectInCell> parent){  
 **if** (!canBeParent(parent)) {  
 **return**;  
 }  
  
 unsetCell();  
 **\_parent** = parent;  
 }  
  
 */\*\*  
 \* Может ли объект быть родителем объекта  
 \** ***@param object*** *проверяемы объект  
 \** ***@return*** *true - если объект может быть родителем  
 \*/* **protected boolean** canBeParent(ObjectInCell object){  
 **boolean** isObjectKeeper = **false**;  
 **if** (object **instanceof** ObjectKeeper<?> objectKeeper){  
 isObjectKeeper = canBeParent(objectKeeper);  
 }  
 **return** isObjectKeeper;  
 }  
  
 */\*\*  
 \* Может ли хранитель быть родителем объекта  
 \** ***@param keeper*** *проверяемый хранитель  
 \** ***@return*** *true - если хранитель может быть родителем  
 \*/* **private boolean** canBeParent(ObjectKeeper<? **extends** ObjectInCell> keeper){  
 **return** keeper.getObjectClass().isAssignableFrom(**this**.getClass());  
 }  
  
 */\* -------------------------- Взаимодействие с другими объектами --------------- \*/  
  
 /\*\*  
 \* Состояние объекта  
 \** ***@return*** *true - если объект уничтожен  
 \*/* **public abstract boolean** isDestroying();  
  
 */\*\*  
 \* Может ли объект находится в одной ячейке с указанным объектом  
 \** ***@param object*** *объект, с котоым проверяется столкновение  
 \** ***@return*** *true если столькновение возможно  
 \*/* **public abstract boolean** canFaceWith(ObjectInCell object);  
  
 */\*\*  
 \* Столкнуть объект с переданным  
 \** ***@param object*** *объект для столкновения  
 \** ***@throws*** *IllegalArgumentException если столкновение невозможно  
 \*/* **protected void** faceWith(ObjectInCell object){  
 **if** (!canFaceWith(object)){  
 **throw new** IllegalArgumentException();  
 }  
 }  
  
 */\*\*  
 \* Обновление состояния объекта, вызываемое из игрового цикла  
 \*/* **protected void** update(){  
 **if** (isDestroying() && getCell() != **null**){  
 getCell().takeObject(**this**);  
 fireEvent(**new** ObjectInCellEvent(**this**, ObjectInCellEvent.EventType.***DESTROYED***));  
 }  
 }  
  
 */\* -------------------- Слушатели и события ------------------- \*/  
  
 /\*\*  
 \* Слушатели игрового объекта  
 \*/* **private final** Set<IObjectInCellEventListener> **\_listeners** = **new** HashSet<>();  
  
 **public void** addListener(IObjectInCellEventListener listener) {  
 **\_listeners**.add(listener);  
 }  
  
 **public void** removeListener(IObjectInCellEventListener listener) {  
 **\_listeners**.remove(listener);  
 }  
  
 */\*\*  
 \* Сообщить об изменение игрового объекта  
 \** ***@param event*** *событие с информацией об объекте  
 \*/* **protected void** fireEvent(ObjectInCellEvent event){  
 **for** (IObjectInCellEventListener listener : **\_listeners**){  
 listener.onObjectInCellAction(event);  
 }  
 }  
}

**public abstract class** MovableObject **extends** ObjectInCell {  
  
 */\*\*  
 \* Направление, в котором движется объект  
 \*/* **private** Direction **\_direction**;  
  
 **public** Direction getDirection(){  
 **return \_direction**;  
 }  
  
 **protected void** setDirection(Direction direction){  
 **\_direction** = direction;  
 }  
  
 */\*\*  
 \* Перемещение объекта в соседнюю ячейку в текущем направлении  
 \** ***@return*** *true если объект перемещен в новую ячейку  
 \*/* **protected boolean** move(){  
 Cell nextCell = getCell().getNeighbour(getDirection());  
 **return** moveAt(nextCell);  
 }  
  
 */\*\*  
 \* Перемещение объекта в указанную ячейку  
 \** ***@param newCell*** *новая ячейка  
 \** ***@return*** *true если объект перемещен в новую ячейку  
 \*/* **protected boolean** moveAt(Cell newCell){  
 **if** (newCell == **null**){  
 **return false**;  
 }  
 **return** newCell.addObject(**this**);  
 }  
  
 @Override  
 **protected void** setParent(ObjectKeeper<? **extends** ObjectInCell> parent){  
 **super**.setParent(parent);  
 }  
}

**public class** Tank **extends** MovableObject **implements** Damageable {  
  
 **public** Tank(Direction direction, **int** health){  
 **if** (health <= 0){  
 **throw new** IllegalArgumentException(**"negative health amount"**);  
 }  
  
 **\_health** = health;  
 setDirection(direction);  
 }  
  
 */\* --------------------- Состояние танка ------------------ \*/  
  
 /\*\*  
 \* Состояние танка, true если танк получает урон  
 \*/* **private boolean \_isDamaged** = **false**;  
  
 */\* -------------------------- Штаб ----------------- \*/  
  
 /\*\*  
 \* Штаб, к которому относиться танк  
 \*/* **private** Headquarters **\_headquarters**;  
  
 **public** Headquarters getHeadquarters(){  
 **return \_headquarters**;  
 }  
  
 **public void** setHeadquarters(Headquarters headquarters){  
 **if** (headquarters == **\_headquarters**){  
 **return**;  
 }  
  
 **\_headquarters** = headquarters;  
 headquarters.setTank(**this**);  
 }  
  
 */\* ----------------- Управление танком ------------ \*/  
  
 /\*\*  
 \* Поворот танка в указанном направлении  
 \** ***@param direction*** *направление поворота  
 \*/* **void** rotate(Direction direction){  
 **if** (direction == getDirection()){  
 **return**;  
 }  
  
 setDirection(direction);  
 fireEvent(**new** ObjectInCellEvent(**this**, ObjectInCellEvent.EventType.***MOVED***));  
 }  
  
 */\*\*  
 \* Время перезарядки орудия  
 \*/* **private final int RELOAD\_TIME** = 3;  
  
 */\*\*  
 \* Текущее время перезарядки  
 \*/* **private int \_currentReloadTime** = 0;  
  
 **public int** getCurrentReloadTime() { **return \_currentReloadTime**; }  
  
 */\*\*  
 \* Соверщить выстрел  
 \** ***@return*** *true если выстрел совершен успешно  
 \*/* **boolean** shoot(){  
 **if** (getCurrentReloadTime() > 0){  
 **return false**;  
 }  
  
 Cell targetCell = getCell().getNeighbour(getDirection());  
 **if** (targetCell == **null**){  
 **return false**;  
 }  
  
 Bullet bullet = **new** Bullet(getDirection(), getCell());  
 bullet.addListener(**new** BulletListener());  
 fireEvent(**new** ObjectInCellEvent(bullet, ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
 **\_currentReloadTime** = **RELOAD\_TIME**;  
 **return true**;  
 }  
  
 */\*\*  
 \* Пропустить ход  
 \*/* **void** pass(){  
 **\_currentReloadTime** -= 1;  
 }  
  
 @Override  
 **protected boolean** move() {  
 **boolean** isMoved = **super**.move();  
 **if** (isMoved){  
 **\_currentReloadTime** -= 1;  
 }  
 **return** isMoved;  
 }  
  
 @Override  
 **protected boolean** moveAt(Cell newCell){  
 **boolean** isMoved = **super**.moveAt(newCell);  
 **if** (isMoved){  
 fireEvent(**new** ObjectInCellEvent(**this**, ObjectInCellEvent.EventType.***MOVED***));  
 }  
 **return** isMoved;  
 }  
  
 */\* -------------------- Взаимодействие с другими объектами ------------------ \*/  
  
 /\*\*  
 \* Здоровье танка  
 \*/* **private int \_health**;  
  
 **public int** getHealth(){  
 **return \_health**;  
 }  
  
 */\*\*  
 \* Полученный танком урон  
 \*/* **private int takenDamage** = 0;  
  
 @Override  
 **public boolean** isDestroying() {  
 **return false**;  
 }  
  
 @Override  
 **public boolean** canFaceWith(ObjectInCell object) {  
 **return** object **instanceof** Damaging || canBeParent(object);  
 }  
  
 @Override  
 **protected void** faceWith(ObjectInCell object) {  
 **super**.faceWith(object);  
  
 **if** (object **instanceof** Damaging){  
 **takenDamage** += 1;  
 **if** (!**\_isDamaged**){  
 **\_isDamaged** = **true**;  
 fireEvent(**new** ObjectInCellEvent(**this**, ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
 }  
 }  
 }  
  
 @Override  
 **protected void** update() {  
 **if** (**\_isDamaged** && getHealth() > 0){  
 **\_health** = getHealth() - **takenDamage**;  
 **\_isDamaged** = **false**;  
 **takenDamage** = 0;  
 fireEvent(**new** ObjectInCellEvent(**this**, ObjectInCellEvent.EventType.***DAMAGED***));  
 }  
 }  
  
 */\* -------------------------- Слушатели и события ------------------- \*/  
  
 /\*\*  
 \* Слушатель снаряда  
 \*/* **private class** BulletListener **implements** IObjectInCellEventListener{  
  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 fireEvent(event);  
 }  
 }  
}

*/\*\*  
 \* Класс игрока, необходимый для управления танком извне  
 \*/***public class** Player {  
  
 */\* ------------------ Состояние игрока ----------------- \*/  
  
 /\*\*  
 \* Состояние игрока, true - если игрок активен  
 \*/* **private boolean \_isActive** = **false**;  
  
 **public void** setActive(**boolean** active){  
 **\_isActive** = active;  
 }  
  
 **public boolean** isActive(){  
 **return \_isActive**;  
 }  
  
 */\* ----------------- Танк игрока ------------ \*/  
  
 /\*\*  
 \* Танк, принадлежащий игроку  
 \*/* **private final** Tank **\_tank**;  
  
 **public** Player(Tank tank){  
 **\_tank** = tank;  
 }  
  
 **public** Tank getTank(){  
 **return \_tank**;  
 }  
  
 */\*\*  
 \* Узнать жив ли игрок  
 \** ***@return*** *true - если танк и штаб игрока целы  
 \*/* **public boolean** isAlive(){  
 **return \_tank**.getHealth() > 0 && **\_tank**.getHeadquarters().getCell() != **null**;  
 }  
  
 */\* --------------------- Управление танком ---------------- \*/  
  
 /\*\*  
 \* Повернуть танк в указанном направлении  
 \** ***@param direction*** *направление поворота  
 \*/* **public void** rotate(Direction direction){  
 **if** (!isActive()){  
 **return**;  
 }  
 **\_tank**.rotate(direction);  
 }  
  
 */\*\*  
 \* Переместить танк  
 \*/* **public void** move(){  
 **if** (!isActive()){  
 **return**;  
 }  
  
 **if** (**\_tank**.move()){  
 firePlayerMadeMove();  
 }  
 }  
  
 */\*\*  
 \* Совершить выстрел  
 \*/* **public void** shoot(){  
 **if** (!isActive()){  
 **return**;  
 }  
  
 **if** (**\_tank**.shoot()){  
 firePlayerMadeMove();  
 }  
 }  
  
 */\*\*  
 \* Пропустить ход  
 \*/* **public void** pass(){  
 **if** (!isActive()){  
 **return**;  
 }  
  
 **\_tank**.pass();  
 firePlayerMadeMove();  
 }  
  
 */\* ------------------- Слушатели и события --------------- \*/* **private final** Set<IPlayerListener> **\_listeners** = **new** HashSet<>();  
  
 **public void** addListener(IPlayerListener listener){  
 **\_listeners**.add(listener);  
 }  
  
 **public void** removeListener(IPlayerListener listener){  
 **\_listeners**.remove(listener);  
 }  
  
 **private void** firePlayerMadeMove(){  
 **for** (IPlayerListener listener : **\_listeners**){  
 listener.onPlayerMadeMove();  
 }  
 }  
}

*/\*\*  
 \* Снаряд выпускаемый танком  
 \*/***public class** Bullet **extends** MovableObject **implements** Damaging {  
  
 */\*\*  
 \* Сосотояние снаряда, true - если мнаряд уничтожен  
 \*/* **private boolean \_isDestroying** = **false**;  
  
 @Override  
 **public boolean** isDestroying(){  
 **return \_isDestroying**;  
 }  
  
 */\*\*  
 \* Состояние снаряда, true - если снаряд самостоятельно не движется  
 \*/* **private boolean \_isFrozen** = **false**;  
  
 **public** Bullet(Direction direction, Cell startCell){  
 setDirection(direction);  
 setCell(startCell);  
 }  
  
 @Override  
 **public boolean** canFaceWith(ObjectInCell object) {  
 **return** !(**this** == object);  
 }  
  
 @Override  
 **protected void** faceWith(ObjectInCell object) {  
 **super**.faceWith(object);  
  
 **if** (object **instanceof** Damageable){  
 **\_isDestroying** = **true**;  
 }  
  
 **if** (canBeParent(object)){  
 **\_isFrozen** = **true**;  
 }  
 }  
  
 @Override  
 **protected void** update() {  
 **if** (**\_isDestroying**){  
 destroy();  
 } **else** {  
 move();  
 }  
 }  
  
 @Override  
 **protected boolean** move(){  
 **return super**.move();  
 }  
  
 @Override  
 **protected boolean** moveAt(Cell newCell){  
 **boolean** isMoved = **super**.moveAt(newCell);  
 **if** (isMoved){  
 fireMoved();  
 } **else** {  
 destroy();  
 }  
 **return** isMoved;  
 }  
  
 */\*\*  
 \* Испустить сообщение о перемещении  
 \*/* **private void** fireMoved(){  
 fireEvent(**new** ObjectInCellEvent(**this**, ObjectInCellEvent.EventType.***MOVED***));  
 **if** (!**\_isFrozen**){  
 fireEvent(**new** ObjectInCellEvent(**this**, ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
 } **else** {  
 **\_isFrozen** = **false**;  
 }  
 }  
  
 */\*\*  
 \* Уничтожить снаряд  
 \*/* **private void** destroy(){  
 getCell().takeObject(**this**);  
 fireEvent(**new** ObjectInCellEvent(**this**, ObjectInCellEvent.EventType.***DESTROYED***));  
 }  
}

*/\*\*  
 \* Взрыв - поражающий объект, который поражает только одну ячейку  
 \*/***public class** Explosion **extends** ObjectInCell **implements** Damaging {  
  
 */\*\*  
 \* Состояние взрыва, true - если взрыв уничтожен  
 \*/* **private boolean \_isDestroying** = **false**;  
  
 @Override  
 **public boolean** isDestroying() {  
 **return \_isDestroying**;  
 }  
  
 @Override  
 **public boolean** canFaceWith(ObjectInCell object) {  
 **return this** != object;  
 }  
  
 @Override  
 **protected void** setCell(Cell cell){  
 **super**.setCell(cell);  
 **if** (cell != **null**) {  
 **\_isDestroying** = **true**;  
 fireEvent(**new** ObjectInCellEvent(**this**, ObjectInCellEvent.EventType.***DAMAGED***));  
 fireEvent(**new** ObjectInCellEvent(**this**, ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
 }  
 }  
  
 @Override  
 **protected void** update(){  
 **super**.update();  
 }  
}

*/\*\*  
 \* Бочка мазута  
 \*/***public class** FuelOilBarrel **extends** Obstacle **implements** Damageable {  
  
 */\*\*  
 \* Состояние бочки, true - если бочка взорвана  
 \*/* **private boolean \_isDestroying** = **false**;  
  
 @Override  
 **public boolean** isDestroying() {  
 **return \_isDestroying**;  
 }  
  
 @Override  
 **protected void** faceWith(ObjectInCell object){  
 **super**.faceWith(object);  
  
 **if** (object **instanceof** Damaging){  
 fireEvent(**new** ObjectInCellEvent(**this**, ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
 }  
 }  
  
 @Override  
 **protected void** update(){  
 **if** (!isDestroying()){  
 detonate();  
 **\_isDestroying** = **true**;  
 fireEvent(**new** ObjectInCellEvent(**this**, ObjectInCellEvent.EventType.***DAMAGED***));  
 fireEvent(**new** ObjectInCellEvent(**this**, ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
 } **else** {  
 **super**.update();  
 }  
 }  
  
 */\*\*  
 \* Взорваться  
 \*/* **private void** detonate(){  
 ExplosionListener listener = **new** ExplosionListener();  
 **for** (Direction direction : Direction.*values*()){  
 Cell neighbour = getCell().getNeighbour(direction);  
 **if** (neighbour != **null**){  
 Explosion explosion = **new** Explosion();  
 explosion.addListener(listener);  
 neighbour.addObject(explosion);  
 }  
 }  
 }  
  
 **private class** ExplosionListener **implements** IObjectInCellEventListener{  
  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 fireEvent(event);  
 }  
 }  
}

*/\*\*  
 \* Заросли  
 \*/***public class** Thicket **extends** Obstacle **implements** ObjectKeeper<Tank> {  
  
 */\*\*  
 \* Танк, который находится в заросли  
 \*/* **private** Tank **\_tank**;  
  
 @Override  
 **public** Tank getObject(){  
 **return \_tank**;  
 }  
  
 @Override  
 **public** Class<Tank> getObjectClass() {  
 **return** Tank.**class**;  
 }  
  
 @Override  
 **public boolean** isDestroying() {  
 **return false**;  
 }  
  
 @Override  
 **public boolean** canFaceWith(ObjectInCell object){  
 **return super**.canFaceWith(object) || (object **instanceof** Tank && getObject() == **null**);  
 }  
  
 @Override  
 **protected void** faceWith(ObjectInCell object) {  
 **super**.faceWith(object);  
  
 **if** (object **instanceof** Tank tank){  
 **\_tank** = tank;  
 getObject().addListener(**\_tankListener**);  
 fireNeedUpdate();  
 }  
 }

@Override  
 **protected void** update(){  
 **if** (getObject() == **null** || getCell() == **null**){  
 **return**;  
 }  
  
 **if** (getObject().getCell() == getCell()){  
 getCell().takeObject(getObject());  
 getObject().setParent(**this**);  
 } **else** {  
 getObject().removeListener(**\_tankListener**);  
 **\_tank** = **null**;  
 }  
 }  
  
 **private void** fireNeedUpdate(){  
 fireEvent(**new** ObjectInCellEvent(**this**, ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
 }  
  
 **private final** TankListener **\_tankListener** = **new** TankListener();  
  
 **private class** TankListener **implements** IObjectInCellEventListener{  
  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 **if** (event.getObject() != getObject()) {  
 **return**;  
 }  
  
 **if** (event.getType() == ObjectInCellEvent.EventType.***MOVED*** && event.getObject().getCell() != getCell()){  
 fireNeedUpdate();  
 }  
 }  
 }  
}

*/\*\*  
 \* Главный класс игровой модели  
 \*/***public class** Game {  
  
 */\*\*  
 \* Состояние игры  
 \*/* **public enum** State {  
 ***GAME\_IS\_ON***,  
 ***WINNER\_FOUND***,  
 ***DRAW*** }  
  
 **public** Game(Environment environment){  
 **\_field** = **new** Field(environment);  
 **\_field**.addObjectInCellListener(**new** ObjectListener());  
  
 ArrayList<Tank> tanks = **\_field**.getTanks();  
 PlayerListener playerListener = **new** PlayerListener();  
 **for** (Tank tank : tanks){  
 Player player = **new** Player(tank);  
 player.addListener(playerListener);  
 **\_players**.add(player);  
 }  
 }  
  
 */\* ----------------- Игровое поле ----------------- \*/  
  
 /\*\*  
 \* Игровое поле  
 \*/* **private final** Field **\_field**;  
  
 **public** Field getField(){  
 **return \_field**;  
 }  
  
 */\*\*  
 \* Запуск игры  
 \*/* **public void** start() {  
 **\_activePlayer** = **\_players**.get(0);  
 activePlayer().setActive(**true**);  
 }  
  
 */\* ---------------- Танки ----------------- \*/* **private final** ArrayList<Player> **\_players** = **new** ArrayList<>();  
  
 **public** List<Player> getPlayers(){  
 **return** Collections.*unmodifiableList*(**\_players**);  
 }  
  
 */\*\*  
 \* Текущий активный танк, которым можно управлять  
 \*/* **private** Player **\_activePlayer** = **null**;  
  
 **public** Player activePlayer() {  
 **return \_activePlayer**;  
 }  
  
 */\*\*  
 \* Танк - победитель, определенный после окончания игры  
 \*/* **private** Player **\_winner** = **null**;  
  
 **public** Player winner() {  
 **return \_winner**;  
 }  
  
 */\*\*  
 \* Получить танк-противник, для данного танка  
 \*  
 \** ***@param player*** *танк, для которого нужно определить противника  
 \** ***@return*** *танк-противник  
 \*/* **private** Player getEnemy(Player player){  
 **return \_players**.stream().filter(p -> p != player).findAny().orElse(player);  
 }  
  
 */\* ----------------- Игровой цикл ------------ \*/  
  
 /\*\*  
 \* События требующие обработки в игровом цикле  
 \*/* **private final** Queue<ObjectInCellEvent> **\_activeEvents** = **new** ArrayDeque<>();  
  
 **private final** Timer **\_timer** = **new** Timer();  
  
 */\*\*  
 \* Игровой цикл  
 \*/* **private void** gameLoop(){  
 **int** eventsCount = **\_activeEvents**.size();  
 **if** (eventsCount > 0){  
 **for** (**int** i = 0; i < eventsCount; i++){  
 ObjectInCellEvent event = **\_activeEvents**.remove();  
 event.getObject().update();  
 }  
 waitRendering();  
 } **else** {  
 State state = checkState();  
 **if** (state == State.***GAME\_IS\_ON***){  
 **\_activePlayer** = getEnemy(activePlayer());  
 activePlayer().setActive(**true**);  
 }  
 fireGameStateGanged(state);  
 }  
 }  
  
 */\*\*  
 \* Ожидание отрисовки ереданных событий  
 \*/* **private void** waitRendering() {  
 **\_timer**.schedule(**new** TimerTask() {  
 @Override  
 **public void** run() {  
 gameLoop();  
 }  
 }, 500L);  
 }  
  
 */\*\*  
 \* Определить текущее состояние игры  
 \** ***@return*** *текущее состояние  
 \*/* **private** State checkState(){  
 Player winner = **null**;  
 **int** alivePlayersCount = 0;  
 **for** (Player player : **\_players**) {  
 **if** (!player.isAlive()) {  
 winner = getEnemy(player);  
 } **else** {  
 alivePlayersCount++;  
 }  
 }  
  
 State state = State.***GAME\_IS\_ON***;  
 **if** (alivePlayersCount == 0) {  
 state = State.***DRAW***;  
 } **else if** (winner != **null**){  
 **\_winner** = winner;  
 state = State.***WINNER\_FOUND***;  
 }  
 **return** state;  
 }  
  
 */\* --------------- Игровые слушатели ---------------- \*/  
  
 /\*\*  
 \* Слушатели игровых событий  
 \*/* **private final** Set<IGameEventListener> **\_listeners** = **new** HashSet<>();  
  
 **public void** addListener(IGameEventListener listener){  
 **\_listeners**.add(listener);  
 }  
  
 */\* ---------------- События -------------------- \*/  
  
 /\*\*  
 \* Сообщить о том, что объект на поле изменился  
 \** ***@param event*** *событие, содержащий информацию об объекте  
 \*/* **private void** fireObjectChanged(ObjectInCellEvent event){  
 **for** (IGameEventListener listener : **\_listeners**){  
 listener.onObjectChanged(event);  
 }  
 }  
  
 */\*\*  
 \* Сообщить о том что состояние игры изиенились  
 \** ***@param state*** *новое состояние игры  
 \*/* **private void** fireGameStateGanged(State state){  
 **for** (IGameEventListener listener : **\_listeners**){  
 listener.onGameStateChanged(state);  
 }  
 }  
  
 */\*\*  
 \* Слушатель игровых объектов нв поле  
 \*/* **private class** ObjectListener **implements** IObjectInCellEventListener{  
  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 **if** (event.getType() == ObjectInCellEvent.EventType.***NEED\_UPDATE***) {  
 **\_activeEvents**.add(event);  
 } **else** {  
 fireObjectChanged(event);  
 }  
 }  
 }  
  
 **private class** PlayerListener **implements** IPlayerListener {  
  
 @Override  
 **public void** onPlayerMadeMove() {  
 activePlayer().setActive(**false**);  
 gameLoop();  
 }  
 }  
}

**public class** RandomEnvironment **extends** Environment{  
  
 **private final** List<Class<? **extends** Obstacle>> **obstacles** = List.*of*(  
 FuelOilBarrel.**class**,  
 Thicket.**class**,  
 Wall.**class**,  
 Water.**class** );  
  
 **private** Obstacle getObstacle(**int** index){  
 Obstacle obstacle = **null**;  
 **try** {  
 obstacle = **obstacles**.get(index).getConstructor().newInstance();  
 } **catch** (NoSuchMethodException | InstantiationException | IllegalAccessException | InvocationTargetException e) {  
 e.printStackTrace();  
 }  
 **return** obstacle;  
 }  
  
 @Override  
 **public** Size fieldSize() {  
 **return new** Size(8, 8);  
 }  
  
 @Override  
 **public void** fillField(Field field) {  
 setTanksWithHeadquarters(field);  
  
 **for** (**int** y = 0; y < fieldSize().height(); y++){  
 **for** (**int** x = 0; x < fieldSize().width(); x++){  
 **int** objectIndex = ThreadLocalRandom.*current*().nextInt(0, 6);  
 Cell cell = field.getCell(**new** Position(x, y));  
 **if** (objectIndex < **obstacles**.size() && cell.getObjects().isEmpty()){  
 field.getCell(**new** Position(x, y)).addObject(getObstacle(objectIndex));  
 }  
 }  
 }  
 }  
  
 **private void** setTanksWithHeadquarters(Field field){  
 Tank firstTank = **new** Tank(Direction.***SOUTH***, 3);  
 Headquarters firstHeadquarters = **new** Headquarters();  
 firstTank.setHeadquarters(firstHeadquarters);  
  
 **int** firstHeadquartersPosition = ThreadLocalRandom.*current*().nextInt(0, fieldSize().width());  
 field.getCell(**new** Position(firstHeadquartersPosition, 0)).addObject(firstHeadquarters);  
  
 **int** firstTankPosition = ThreadLocalRandom.*current*().nextInt(0, fieldSize().width());  
 field.getCell(**new** Position(firstTankPosition, 1)).addObject(firstTank);  
  
 Tank secondTank = **new** Tank(Direction.***NORTH***, 3);  
 Headquarters secondHeadquarters = **new** Headquarters();  
 secondTank.setHeadquarters(secondHeadquarters);  
  
 **int** secondHeadquartersPosition = ThreadLocalRandom.*current*().nextInt(0, fieldSize().width());  
 field.getCell(**new** Position(secondHeadquartersPosition, fieldSize().height() - 1)).addObject(secondHeadquarters);  
  
 **int** secondTankPosition = ThreadLocalRandom.*current*().nextInt(0, fieldSize().width());  
 field.getCell(**new** Position(secondTankPosition, fieldSize().height() - 2)).addObject(secondTank);  
 }  
}

**public class** ExplosionWidget **extends** ObjectInCellWidget{  
  
 **public** ExplosionWidget(Explosion explosion) {  
 **super**(explosion);  
 }  
  
 @Override  
 **public** Dimension getDimension(){  
 **return new** Dimension(60, 60);  
 }  
  
 @Override  
 **protected** File getImageFile() {  
 **return new** File(**"resources/explosion.png"**);  
 }  
  
 @Override  
 **public** Layer getLayer(){  
 **return** Layer.***TOP***;  
 }  
  
 @Override  
 **protected void** getDamage(){  
 repaint();  
 revalidate();  
 }  
}

**public class** FuelOilBarrelWidget **extends** ObjectInCellWidget{  
  
 **public** FuelOilBarrelWidget(FuelOilBarrel barrel) {  
 **super**(barrel);  
 }  
  
 @Override  
 **protected void** getDamage(){  
 repaint();  
 }  
  
 @Override  
 **public** Dimension getDimension(){  
 **return new** Dimension(60, 60);  
 }  
  
 @Override  
 **protected** File getImageFile() {  
 **return new** File(getImageFileName());  
 }  
  
 **private** String getImageFileName(){  
 String filename = **"resources/fuel\_oil\_barrel"**;  
 **if** (getObject().isDestroying()){  
 filename += **"\_detonating"**;  
 }  
 filename += **".png"**;  
  
 **return** filename;  
 }  
}

*/\*\*  
 \* Представление панели игрока  
 \*/***public class** PlayerPanel **extends** JPanel {  
  
 */\*\*  
 \* Игрок  
 \*/* **private final** Player **\_player**;  
  
 **private final** TankWidget **\_tankWidget**;  
  
 */\*\*  
 \* Максимальное кол-во жизней игрока  
 \*/* **private final int \_maxHeath**;  
  
 */\*\*  
 \* Цвет игрока  
 \*/* **private final** Color **\_color**;  
  
 **public** PlayerPanel(Player player, Color color, WidgetPool widgetPool){  
 **\_player** = player;  
 **\_tankWidget** = (TankWidget) widgetPool.getWidget(**\_player**.getTank());  
 HeadquartersWidget headquartersWidget = (HeadquartersWidget) widgetPool.getWidget(**\_player**.getTank().getHeadquarters());  
  
 **\_color** = color;  
 **\_tankWidget**.setColor(color);  
 headquartersWidget.setColor(color);  
  
 **\_maxHeath** = **\_player**.getTank().getHealth();  
  
 setPreferredSize(**new** Dimension(150, 600));  
 setLayout(**new** GridLayout(3, 1, 0, 50));  
 setBackground(Color.***BLACK***);  
 addKeyListener(**new** KeyListener());  
  
 update();  
 }  
  
 **public void** setActive(**boolean** active){  
 **\_tankWidget**.setActive(active);  
 setFocusable(active);  
 requestFocus();  
 }  
  
 */\*\*  
 \* Обновить информацию об игроке  
 \*/* **public void** update(){  
 removeAll();  
 add(getFlagBar());  
 add(getHeathBar());  
 add(getReloadGunBar());  
 repaint();  
 revalidate();  
 }  
  
 **private final static** Map<Color, String> ***names*** = Map.*ofEntries*(  
 Map.*entry*(Color.***BLUE***, **"Синий"**),  
 Map.*entry*(Color.***ORANGE***, **"Оранжевый"**)  
 );  
  
 **public** String getName(){  
 **return *names***.get(**\_color**);  
 }  
  
 */\*\*  
 \* Получить панель с жизнями игрока  
 \** ***@return*** *панель с жизнями игрока  
 \*/* **private** JPanel getHeathBar(){  
 JPanel heathBar = **new** JPanel();  
 heathBar.setLayout(**new** GridLayout(**\_maxHeath**, 1));  
 heathBar.setBackground(Color.***BLACK***);  
  
 **for** (**int** i = 1; i <= **\_maxHeath**; i++){  
 **boolean** heathActive = i <= **\_player**.getTank().getHealth();  
 heathBar.add(**new** JLabel(**new** ImageIcon(getHeartImage(heathActive))));  
 }  
  
 **return** heathBar;  
 }  
  
 */\*\*  
 \* Получить изображение сердца  
 \** ***@param isActive*** *активное сердце - красное, неактивное - серое  
 \** ***@return*** *изображения сердца  
 \*/* **private** BufferedImage getHeartImage(**boolean** isActive){  
 String path = isActive ? **"resources/red\_heart.png"** : **"resources/grey\_heart.png"**;  
  
 **return** getImage(**new** File(path), **new** Dimension(50, 50));  
 }  
  
 */\*\*  
 \* Получить панель с флагом игрока  
 \** ***@return*** *панель с флагом  
 \*/* **private** JPanel getFlagBar(){  
 JPanel flagBar = **new** JPanel();  
 flagBar.setLayout(**new** GridBagLayout());  
 flagBar.setBackground(Color.***BLACK***);  
 flagBar.add(**new** JLabel(**new** ImageIcon(getFlagImage())));  
 **return** flagBar;  
 }  
  
 */\*\*  
 \* Получить изображение флага игрока, соотыетсвующее его цвету  
 \** ***@return*** *изображение флага  
 \*/* **private** BufferedImage getFlagImage(){  
 String path = **"resources/"** + ColorUtil.*ColorName*(**\_color**) + **"\_flag.png"**;  
  
 **return** getImage(**new** File(path), **new** Dimension(100, 100));  
 }  
  
 */\*\*  
 \* Получить панель перезарядки орудия  
 \** ***@return*** *панель перезарядки  
 \*/* **private** JPanel getReloadGunBar(){  
 JPanel reloadPanel = **new** JPanel();  
 reloadPanel.setLayout(**new** GridBagLayout());  
 reloadPanel.setBackground(Color.***BLACK***);  
  
 JLabel label;  
 **if** (**\_player**.getTank().getCurrentReloadTime() > 0){  
 label = **new** JLabel(Integer.*toString*(**\_player**.getTank().getCurrentReloadTime()),  
 **new** ImageIcon(getReloadImage(**false**)),  
 JLabel.***CENTER***);  
 label.setHorizontalTextPosition(JLabel.***RIGHT***);  
 label.setVerticalTextPosition(JLabel.***CENTER***);  
 label.setFont(**new** Font(**"Serif"**, Font.***PLAIN***, 36));  
 label.setForeground(Color.*decode*(**"#6d6d6d"**));  
 } **else** {  
 label = **new** JLabel(**new** ImageIcon(getReloadImage(**true**)));  
 }  
  
 reloadPanel.add(label);  
 **return** reloadPanel;  
 }  
  
 */\*\*  
 \* Получить изображение перезарядки орудия  
 \** ***@param isActive*** *активность орудия  
 \** ***@return*** *изображение перезарядки  
 \*/* **private** BufferedImage getReloadImage(**boolean** isActive){  
 String path = isActive ? **"resources/bullet.png"** : **"resources/grey\_bullet.png"**;  
  
 **return** getImage(**new** File(path), **new** Dimension(50, 50));  
 }  
  
 */\*\*  
 \* Получить изображение из файла  
 \** ***@param file*** *файл с изображением  
 \** ***@param dimension*** *размер изображения  
 \** ***@return*** *изображение из файла  
 \*/* **private** BufferedImage getImage(File file, Dimension dimension){  
 BufferedImage image = **null**;  
 **try** {  
 image = ImageIO.*read*(file);  
 image = ImageUtils.*resizeImage*(image, dimension.**width**, dimension.**height**);  
 } **catch** (IOException e){  
 e.printStackTrace();  
 }  
 **return** image;  
 }  
  
 */\*\*  
 \* Слушатель пользовательского ввода  
 \*/* **private class** KeyListener **implements** java.awt.event.KeyListener{  
  
 @Override  
 **public void** keyTyped(KeyEvent e) {  
  
 }  
  
 @Override  
 **public void** keyPressed(KeyEvent e) {  
 setActive(**false**);  
 **int** key = e.getKeyCode();  
 move(key);  
 pass(key);  
 shoot(key);  
 rotate(key);  
 setActive(**\_player**.isActive());  
 }  
  
 **private void** move(**int** keyCode){  
 **if** (keyCode == KeyEvent.***VK\_SPACE***){  
 **\_player**.move();  
 }  
 }  
  
 **private void** shoot(**int** keyCode){  
 **if** (keyCode == KeyEvent.***VK\_ENTER***){  
 **\_player**.shoot();  
 }  
 }  
  
 **private void** pass(**int** keyCode){  
 **if** (keyCode == KeyEvent.***VK\_BACK\_SPACE***){  
 **\_player**.pass();  
 }  
 }  
  
 **private void** rotate(**int** keyCode){  
 Map<Integer, Direction> keyCodeToDirection = **new** HashMap<>();  
 keyCodeToDirection.put(KeyEvent.***VK\_W***, Direction.***NORTH***);  
 keyCodeToDirection.put(KeyEvent.***VK\_A***, Direction.***WEST***);  
 keyCodeToDirection.put(KeyEvent.***VK\_S***, Direction.***SOUTH***);  
 keyCodeToDirection.put(KeyEvent.***VK\_D***, Direction.***EAST***);  
  
 **if** (keyCodeToDirection.containsKey(keyCode)){  
 **\_player**.rotate(keyCodeToDirection.get(keyCode));  
 }  
 }  
  
 @Override  
 **public void** keyReleased(KeyEvent e) {  
  
 }  
 }  
}

**public class** ThicketWidget **extends** ObjectInCellWidget{  
  
 **public** ThicketWidget(Thicket object) {  
 **super**(object);  
 }  
  
 @Override  
 **protected** File getImageFile() {  
 **return new** File(**"resources/thicket.png"**);  
 }  
  
 @Override  
 **public** Layer getLayer(){  
 **return** Layer.***MIDDLE***;  
 }  
  
 @Override  
 **public** Dimension getDimension(){  
 **return new** Dimension(60, 60);  
 }  
}

**public class** WidgetFactory {  
  
 **private static final** Map<Class<? **extends** ObjectInCell>, Class<? **extends** ObjectInCellWidget>> ***widgetMap*** = Map.*ofEntries*(  
 Map.*entry*(Bullet.**class**, BulletWidget.**class**),  
 Map.*entry*(Explosion.**class**, ExplosionWidget.**class**),  
 Map.*entry*(FuelOilBarrel.**class**, FuelOilBarrelWidget.**class**),  
 Map.*entry*(Headquarters.**class**, HeadquartersWidget.**class**),  
 Map.*entry*(Tank.**class**, TankWidget.**class**),  
 Map.*entry*(Thicket.**class**, ThicketWidget.**class**),  
 Map.*entry*(Wall.**class**, WallWidget.**class**),  
 Map.*entry*(Water.**class**, WaterWidget.**class**)  
 );  
  
 **public** ObjectInCellWidget createWidget(ObjectInCell objectInCell){  
 ObjectInCellWidget widget = **null**;  
 **try** {  
 Constructor<?> constructor = ***widgetMap***.get(objectInCell.getClass()).getConstructor(objectInCell.getClass());  
 widget = (ObjectInCellWidget) constructor.newInstance(objectInCell);  
 } **catch** (NoSuchMethodException | InstantiationException | IllegalAccessException | InvocationTargetException e) {  
 e.printStackTrace();  
 }  
 **return** widget;  
 }  
  
 **public** CellWidget createWidget(Cell cell){  
 **return new** CellWidget(cell);  
 }  
}

*/\*\*  
 \* Пул виджетов, расположенных на поле  
 \*/***public class** WidgetPool {  
  
 **private final** WidgetFactory **\_factory** = **new** WidgetFactory();  
  
 **private final** Map<ObjectInCell, ObjectInCellWidget> **\_objectWidgets** = **new** HashMap<>();  
  
 **private final** Map<Cell, CellWidget> **\_cellWidgets** = **new** HashMap<>();  
  
 **public** ObjectInCellWidget getWidget(ObjectInCell object){  
 ObjectInCellWidget widget = **\_objectWidgets**.get(object);  
 **if** (widget == **null**){  
 widget = createWidget(object);  
 }  
 **return** widget;  
 }  
  
 **public** ObjectInCellWidget removeWidget(ObjectInCell object){  
 **return \_objectWidgets**.remove(object);  
 }  
  
 **private** ObjectInCellWidget createWidget(ObjectInCell object){  
 **if** (**\_objectWidgets**.containsKey(object)){  
 **throw new** IllegalArgumentException();  
 }  
  
 ObjectInCellWidget widget = **\_factory**.createWidget(object);  
 getWidget(object.getCell()).addObjectWidget(widget);  
 **\_objectWidgets**.put(object, widget);  
 **return** widget;  
 }  
  
 **public** CellWidget getWidget(Cell cell){  
 CellWidget widget = **\_cellWidgets**.get(cell);  
 **if** (widget == **null**){  
 widget = createWidget(cell);  
 }  
 **return** widget;  
 }  
  
 **private** CellWidget createWidget(Cell cell){  
 **if** (**\_cellWidgets**.containsKey(cell)){  
 **throw new** IllegalArgumentException();  
 }  
  
 CellWidget widget = **\_factory**.createWidget(cell);  
 **\_cellWidgets**.put(cell, widget);  
 **for** (ObjectInCell object : cell.getObjects()) {  
 createWidget(object);  
 }  
 **return** widget;  
 }  
}

*/\*\*  
 \* Представление игры  
 \*/***public class** GamePanel **extends** JFrame {  
  
 */\*\*  
 \* Пул виджетов, содержащий виджеты ячеек и объектов на поле  
 \*/* **private final** WidgetPool **\_pool** = **new** WidgetPool();  
  
 */\*\*  
 \* Панели игроков  
 \*/* **private final** Map<Player, PlayerPanel> **\_playerPanels** = **new** HashMap<>();  
  
 */\*\*  
 \* Модель игры  
 \*/* **private** Game **\_game**;  
  
 **public** GamePanel(){  
 create();  
 pack();  
 setDefaultCloseOperation(***EXIT\_ON\_CLOSE***);  
 setResizable(**false**);  
 setVisible(**true**);  
 start();  
 }  
  
 */\*\*  
 \* Создать игру и ее представление  
 \*/* **private void** create(){  
 **\_game** = **new** Game(**new** RandomEnvironment());  
 FieldWidget fieldWidget = **new** FieldWidget(**\_game**.getField(), **\_pool**);  
 **\_game**.addListener(**new** GameListener());  
  
 JPanel content = (JPanel) getContentPane();  
 content.removeAll();  
 content.setLayout(**new** BoxLayout(content, BoxLayout.***X\_AXIS***));  
  
 **\_playerPanels**.clear();  
 initializePlayers();  
 **for**(PlayerPanel playerPanel : **\_playerPanels**.values()){  
 content.add(playerPanel);  
 }  
 content.add(fieldWidget, 1);  
  
 repaint();  
 revalidate();  
 }  
  
 */\*\*  
 \* Присвоить игрокам их цвета  
 \*/* **private void** initializePlayers(){  
 Color[] colors = **new** Color[] {Color.***BLUE***, Color.***ORANGE***};  
 **for** (**int** i = 0; i < **\_game**.getPlayers().size(); i++){  
 Player player = **\_game**.getPlayers().get(i);  
 PlayerPanel playerPanel = **new** PlayerPanel(player, colors[i], **\_pool**);  
 **\_playerPanels**.put(player, playerPanel);  
 }  
 }  
  
 */\*\*  
 \* Запустить игру  
 \*/* **private void** start(){  
 **\_game**.start();  
 **\_playerPanels**.get(**\_game**.activePlayer()).setActive(**true**);  
 }  
  
 */\*\*  
 \* Перезапустить игру  
 \*/* **private void** restart(){  
 create();  
 start();  
 }  
  
 */\*\*  
 \* Слушатель игровых событий  
 \*/* **private class** GameListener **implements** IGameEventListener {  
  
 @Override  
 **public void** onObjectChanged(ObjectInCellEvent event) {  
 ObjectInCellWidget widget = **\_pool**.getWidget(event.getObject());  
 **if** (event.getType() == ObjectInCellEvent.EventType.***MOVED*** && widget **instanceof** MovableObjectWidget movableWidget){  
 CellWidget cellWidget = event.getObject().getCell() != **null** ? **\_pool**.getWidget(event.getObject().getCell()) : **null**;  
 movableWidget.moveTo(cellWidget);  
 } **else if** (event.getType() == ObjectInCellEvent.EventType.***DAMAGED***) {  
 widget.getDamage();  
 } **else** {  
 widget.destroy();  
 **\_pool**.removeWidget(event.getObject());  
 }  
 }  
  
 @Override  
 **public void** onGameStateChanged(Game.State state) {  
 **for** (PlayerPanel panel : **\_playerPanels**.values()) {  
 panel.update();  
 }  
  
 **if** (state == Game.State.***GAME\_IS\_ON***){  
 **\_playerPanels**.get(**\_game**.activePlayer()).setActive(**true**);  
 } **else if** (state == Game.State.***WINNER\_FOUND***){  
 String winnerName = **\_playerPanels**.get(**\_game**.winner()).getName();  
 JOptionPane.*showMessageDialog*(GamePanel.**this**, winnerName + **" танк победил"**);  
 restart();  
 } **else if** (state == Game.State.***DRAW***) {  
 JOptionPane.*showMessageDialog*(GamePanel.**this**, **"Игра завершилась вничью"**);  
 restart();  
 }  
 }  
 }  
}

## 4.8 Реализация ключевых тестовых случаев

**public class** BulletTest {  
  
 Bullet **bullet**;  
 Cell **startCell**;  
 Cell **targetCell**;  
  
 **private final** ArrayList<ObjectInCellEvent> **events** = **new** ArrayList<>();  
  
 **private class** BulletListener **implements** IObjectInCellEventListener{  
  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 **events**.add(event);  
 }  
 }  
  
 @BeforeEach  
 **public void** createTestConfiguration(){  
 Field field = **new** Field(**new** Environment() {  
 @Override  
 **public** Size fieldSize() {  
 **return new** Size(1, 3);  
 }  
  
 @Override  
 **public void** fillField(Field field) {  
  
 }  
 });  
  
 **startCell** = field.getCell(**new** Position(0, 0));  
 **targetCell** = field.getCell(**new** Position(0, 1));  
 **bullet** = **new** Bullet(Direction.***SOUTH***, **startCell**);  
 **events**.clear();  
 }  
  
 @Test  
 **public void** constructor\_creationBullet(){  
 **bullet** = **new** Bullet(Direction.***NORTH***, **startCell**);  
  
 Assertions.*assertSame*(**startCell**, **bullet**.getCell());  
 Assertions.*assertEquals*(Direction.***NORTH***, **bullet**.getDirection());  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithTank(){  
 Tank tank = **new** TankForTest();  
  
 **boolean** result = **bullet**.canFaceWith(tank);  
  
 Assertions.*assertTrue*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithWall(){  
 Obstacle wall = **new** Wall();  
  
 **boolean** result = **bullet**.canFaceWith(wall);  
  
 Assertions.*assertTrue*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithWater(){  
 Obstacle water = **new** Water();  
  
 **boolean** result = **bullet**.canFaceWith(water);  
  
 Assertions.*assertTrue*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithBullet(){  
 Bullet otherBullet = **new** BulletForTest();  
  
 **boolean** result = **bullet**.canFaceWith(otherBullet);  
  
 Assertions.*assertTrue*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithSameObject(){  
 **boolean** result = **bullet**.canFaceWith(**bullet**);  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithTank(){  
 Tank tank = **new** TankForTest();  
  
 **bullet**.addListener(**new** BulletListener());  
  
 **bullet**.faceWith(tank);  
  
 Assertions.*assertTrue*(**bullet**.isDestroying());  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithWall(){  
 Wall wall = **new** Wall();  
  
 **bullet**.addListener(**new** BulletListener());  
  
 **bullet**.faceWith(wall);  
  
 Assertions.*assertTrue*(**bullet**.isDestroying());  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithWater(){  
 Water water = **new** Water();  
  
 **bullet**.addListener(**new** BulletListener());  
  
 **bullet**.faceWith(water);  
  
 Assertions.*assertFalse*(**bullet**.isDestroying());  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithHeadquarters(){  
 Headquarters headquarters = **new** Headquarters();  
  
 **bullet**.addListener(**new** BulletListener());  
  
 **bullet**.faceWith(headquarters);  
  
 Assertions.*assertTrue*(**bullet**.isDestroying());  
 }  
  
 @Test  
 **public void** move\_movingInEmptyCell(){  
 **bullet**.addListener(**new** BulletListener());  
  
 **bullet**.move();  
  
 ArrayList<ObjectInCellEvent> expectedEvents = **new** ArrayList<>();  
 expectedEvents.add(**new** ObjectInCellEvent(**bullet**, ObjectInCellEvent.EventType.***MOVED***));  
 expectedEvents.add(**new** ObjectInCellEvent(**bullet**, ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
  
 Assertions.*assertEquals*(expectedEvents.size(), **events**.size());  
 **for**(**int** i = 0; i < **events**.size(); i++){  
 Assertions.*assertEquals*(expectedEvents.get(i).getType(), **events**.get(i).getType());  
 Assertions.*assertEquals*(expectedEvents.get(i).getObject(), **events**.get(i).getObject());  
 }  
 }  
  
 @Test  
 **public void** move\_movingInCellWithWall(){  
 **targetCell**.addObject(**new** Wall());  
  
 **bullet**.addListener(**new** BulletListener());  
  
 **bullet**.move();  
  
 ArrayList<ObjectInCellEvent> expectedEvents = **new** ArrayList<>();  
 expectedEvents.add(**new** ObjectInCellEvent(**bullet**, ObjectInCellEvent.EventType.***MOVED***));  
 expectedEvents.add(**new** ObjectInCellEvent(**bullet**, ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
  
 Assertions.*assertEquals*(expectedEvents.size(), **events**.size());  
 **for**(**int** i = 0; i < **events**.size(); i++){  
 Assertions.*assertEquals*(expectedEvents.get(i).getType(), **events**.get(i).getType());  
 Assertions.*assertEquals*(expectedEvents.get(i).getObject(), **events**.get(i).getObject());  
 }  
 Assertions.*assertTrue*(**bullet**.isDestroying());  
 }  
  
 @Test  
 **public void** move\_movingInBulletKeeper(){  
 **targetCell**.addObject(**new** BulletKeeper());  
  
 **bullet**.addListener(**new** BulletListener());  
  
 **bullet**.move();  
  
 ArrayList<ObjectInCellEvent> expectedEvents = **new** ArrayList<>();  
 expectedEvents.add(**new** ObjectInCellEvent(**bullet**, ObjectInCellEvent.EventType.***MOVED***));  
  
 Assertions.*assertEquals*(expectedEvents.size(), **events**.size());  
 **for**(**int** i = 0; i < **events**.size(); i++){  
 Assertions.*assertEquals*(expectedEvents.get(i).getType(), **events**.get(i).getType());  
 Assertions.*assertEquals*(expectedEvents.get(i).getObject(), **events**.get(i).getObject());  
 }  
 Assertions.*assertFalse*(**bullet**.isDestroying());  
 }  
  
 @Test  
 **public void** move\_movingFromBulletKeeper(){  
 **targetCell**.addObject(**new** BulletKeeper());  
  
 **bullet**.move();  
  
 **bullet**.addListener(**new** BulletListener());  
  
 **bullet**.move();  
  
 ArrayList<ObjectInCellEvent> expectedEvents = **new** ArrayList<>();  
 expectedEvents.add(**new** ObjectInCellEvent(**bullet**, ObjectInCellEvent.EventType.***MOVED***));  
 expectedEvents.add(**new** ObjectInCellEvent(**bullet**, ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
  
 Assertions.*assertEquals*(expectedEvents.size(), **events**.size());  
 **for**(**int** i = 0; i < **events**.size(); i++){  
 Assertions.*assertEquals*(expectedEvents.get(i).getType(), **events**.get(i).getType());  
 Assertions.*assertEquals*(expectedEvents.get(i).getObject(), **events**.get(i).getObject());  
 }  
 Assertions.*assertFalse*(**bullet**.isDestroying());  
 }  
  
 @Test **void** update\_moving(){  
 **bullet**.addListener(**new** BulletListener());  
  
 **bullet**.update();  
  
 ArrayList<ObjectInCellEvent> expectedEvents = **new** ArrayList<>();  
 expectedEvents.add(**new** ObjectInCellEvent(**bullet**, ObjectInCellEvent.EventType.***MOVED***));  
 expectedEvents.add(**new** ObjectInCellEvent(**bullet**, ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
  
 Assertions.*assertEquals*(expectedEvents.size(), **events**.size());  
 **for**(**int** i = 0; i < **events**.size(); i++){  
 Assertions.*assertEquals*(expectedEvents.get(i).getType(), **events**.get(i).getType());  
 Assertions.*assertEquals*(expectedEvents.get(i).getObject(), **events**.get(i).getObject());  
 }  
 }  
  
 @Test  
 **public void** update\_detonating(){  
 **targetCell**.addObject(**new** Wall());  
  
 **bullet**.addListener(**new** BulletListener());  
  
 **targetCell**.addObject(**bullet**);  
 **bullet**.update();  
  
 ArrayList<ObjectInCellEvent> expectedEvents = **new** ArrayList<>();  
 expectedEvents.add(**new** ObjectInCellEvent(**bullet**, ObjectInCellEvent.EventType.***DESTROYED***));  
  
 Assertions.*assertEquals*(expectedEvents.size(), **events**.size());  
 **for**(**int** i = 0; i < **events**.size(); i++){  
 Assertions.*assertEquals*(expectedEvents.get(i).getType(), **events**.get(i).getType());  
 Assertions.*assertEquals*(expectedEvents.get(i).getObject(), **events**.get(i).getObject());  
 }  
 Assertions.*assertNull*(**bullet**.getCell());  
 }  
}

**public class** ExplosionTest {  
  
 **private** Explosion **explosion**;  
  
 **private final** ArrayList<ObjectInCellEvent> **events** = **new** ArrayList<>();  
  
 **private class** ObjectListener **implements** IObjectInCellEventListener {  
  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 **events**.add(event);  
 }  
 }  
  
 @BeforeEach  
 **public void** createTestConfiguration(){  
 **explosion** = **new** Explosion();  
 **events**.clear();  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithTank(){  
 Tank tank = **new** TankForTest();  
  
 **boolean** result = **explosion**.canFaceWith(tank);  
  
 Assertions.*assertTrue*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithWall(){  
 Obstacle wall = **new** Wall();  
  
 **boolean** result = **explosion**.canFaceWith(wall);  
  
 Assertions.*assertTrue*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithWater(){  
 Obstacle water = **new** Water();  
  
 **boolean** result = **explosion**.canFaceWith(water);  
  
 Assertions.*assertTrue*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithBullet(){  
 Explosion otherExplosion = **new** Explosion();  
  
 **boolean** result = **explosion**.canFaceWith(otherExplosion);  
  
 Assertions.*assertTrue*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithSameObject(){  
 **boolean** result = **explosion**.canFaceWith(**explosion**);  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** setCell\_addingToCell(){  
 Cell cell = **new** Cell(**new** Position(0, 0));  
 **explosion**.addListener(**new** ObjectListener());  
  
 **explosion**.setCell(cell);  
  
 ArrayList<ObjectInCellEvent> expectedEvents = **new** ArrayList<>();  
 expectedEvents.add(**new** ObjectInCellEvent(**explosion**, ObjectInCellEvent.EventType.***DAMAGED***));  
 expectedEvents.add(**new** ObjectInCellEvent(**explosion**, ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
  
 Assertions.*assertEquals*(expectedEvents.size(), **events**.size());  
 **for**(**int** i = 0; i < **events**.size(); i++){  
 Assertions.*assertEquals*(expectedEvents.get(i).getType(), **events**.get(i).getType());  
 Assertions.*assertEquals*(expectedEvents.get(i).getObject(), **events**.get(i).getObject());  
 }  
 Assertions.*assertTrue*(**explosion**.isDestroying());  
 }  
  
 @Test  
 **public void** update\_destroying(){  
 Cell cell = **new** Cell(**new** Position(0, 0));  
 **explosion**.addListener(**new** ObjectListener());  
  
 cell.addObject(**explosion**);  
 **explosion**.update();  
  
 ArrayList<ObjectInCellEvent> expectedEvents = **new** ArrayList<>();  
 expectedEvents.add(**new** ObjectInCellEvent(**explosion**, ObjectInCellEvent.EventType.***DAMAGED***));  
 expectedEvents.add(**new** ObjectInCellEvent(**explosion**, ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
 expectedEvents.add(**new** ObjectInCellEvent(**explosion**, ObjectInCellEvent.EventType.***DESTROYED***));  
  
 Assertions.*assertEquals*(expectedEvents.size(), **events**.size());  
 **for**(**int** i = 0; i < **events**.size(); i++){  
 Assertions.*assertEquals*(expectedEvents.get(i).getType(), **events**.get(i).getType());  
 Assertions.*assertEquals*(expectedEvents.get(i).getObject(), **events**.get(i).getObject());  
 }  
 Assertions.*assertTrue*(**explosion**.isDestroying());  
 Assertions.*assertNull*(**explosion**.getCell());  
 }  
}

**public class** TankTest {  
  
 **private** Tank **tank**;  
  
 **protected** Field **field**;  
  
 **private class** TankListener **implements** IObjectInCellEventListener {  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 **events**.add(event);  
 }  
 }  
  
 **private final** ArrayList<ObjectInCellEvent> **events** = **new** ArrayList<>();  
  
 @BeforeEach  
 **public void** testConfiguration(){  
 **field** = **new** Field(**new** Environment() {  
 @Override  
 **public** Size fieldSize() {  
 **return new** Size(1, 2);  
 }  
  
 @Override  
 **public void** fillField(Field field) {  
  
 }  
 });  
  
 **tank** = **new** Tank(Direction.***NORTH***, 3);  
 }  
  
 @Test  
 **public void** constructor\_tankCreation(){  
 **tank** = **new** Tank(Direction.***NORTH***, 3);  
  
 Assertions.*assertNull*(**tank**.getCell());  
 Assertions.*assertNull*(**tank**.getHeadquarters());  
 Assertions.*assertEquals*(Direction.***NORTH***, **tank**.getDirection());  
 Assertions.*assertEquals*(3, **tank**.getHealth());  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithObstacle(){  
 Obstacle obstacle = **new** Wall();  
  
 **boolean** result = **tank**.canFaceWith(obstacle);  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithSameObject(){  
 **boolean** result = **tank**.canFaceWith(**tank**);  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithBullet(){  
 Bullet bullet = **new** BulletForTest();  
  
 **boolean** result = **tank**.canFaceWith(bullet);  
  
 Assertions.*assertTrue*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithTank(){  
 Tank otherTank = **new** TankForTest();  
  
 Assertions.*assertThrows*(IllegalArgumentException.**class**, () ->  
 **tank**.faceWith(otherTank));  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithThicket(){  
 Thicket thicket = **new** Thicket();  
  
 **boolean** result = **tank**.canFaceWith(thicket);  
  
 Assertions.*assertTrue*(result);  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithObstacle(){  
 Obstacle wall = **new** Wall();  
  
 Assertions.*assertThrows*(IllegalArgumentException.**class**, () ->  
 **tank**.faceWith(wall));  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithBullet() {  
 Bullet bullet = **new** BulletForTest();  
  
 **tank**.addListener(**new** TankListener());  
  
 **tank**.faceWith(bullet);  
  
 ArrayList<ObjectInCellEvent> expectedEvents = **new** ArrayList<>();  
 expectedEvents.add(**new** ObjectInCellEvent(**tank**, ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
  
 Assertions.*assertEquals*(expectedEvents.size(), **events**.size());  
 **for**(**int** i = 0; i < **events**.size(); i++){  
 Assertions.*assertEquals*(expectedEvents.get(i).getType(), **events**.get(i).getType());  
 Assertions.*assertEquals*(expectedEvents.get(i).getObject(), **events**.get(i).getObject());  
 }  
 }  
  
 @Test  
 **public void** rotate\_rotateTank(){  
 **tank**.addListener(**new** TankListener());  
  
 **tank**.rotate(Direction.***SOUTH***);  
  
 ArrayList<ObjectInCellEvent> expectedEvents = **new** ArrayList<>();  
 expectedEvents.add(**new** ObjectInCellEvent(**tank**, ObjectInCellEvent.EventType.***MOVED***));  
  
 Assertions.*assertEquals*(expectedEvents.size(), **events**.size());  
 **for**(**int** i = 0; i < **events**.size(); i++){  
 Assertions.*assertEquals*(expectedEvents.get(i).getType(), **events**.get(i).getType());  
 Assertions.*assertEquals*(expectedEvents.get(i).getObject(), **events**.get(i).getObject());  
 }  
  
 Assertions.*assertEquals*(Direction.***SOUTH***, **tank**.getDirection());  
 }  
  
 @Test  
 **public void** rotate\_rotateTankOnSameDirection(){  
 **tank**.addListener(**new** TankListener());  
  
 **tank**.rotate(Direction.***NORTH***);  
  
 Assertions.*assertEquals*(Direction.***NORTH***, **tank**.getDirection());  
 Assertions.*assertEquals*(0, **events**.size());  
 }  
  
 @Test  
 **public void** move\_movingInEmptyCell(){  
 Cell startCell = **field**.getCell(**new** Position(0, 1));  
  
 startCell.addObject(**tank**);  
 **tank**.rotate(Direction.***NORTH***);  
  
 **tank**.addListener(**new** TankListener());  
  
 **boolean** result = **tank**.move();  
  
 ArrayList<ObjectInCellEvent> expectedEvents = **new** ArrayList<>();  
 expectedEvents.add(**new** ObjectInCellEvent(**tank**, ObjectInCellEvent.EventType.***MOVED***));  
  
 Assertions.*assertEquals*(expectedEvents.size(), **events**.size());  
 **for**(**int** i = 0; i < **events**.size(); i++){  
 Assertions.*assertEquals*(expectedEvents.get(i).getType(), **events**.get(i).getType());  
 Assertions.*assertEquals*(expectedEvents.get(i).getObject(), **events**.get(i).getObject());  
 }  
 Assertions.*assertTrue*(result);  
 }  
  
 @Test  
 **public void** move\_movingInNotEmptyCell(){  
 Cell startCell = **field**.getCell(**new** Position(0, 1));  
 Cell targetCell = **field**.getCell(**new** Position(0, 0));  
 Wall wall = **new** Wall();  
  
 startCell.addObject(**tank**);  
 targetCell.addObject(wall);  
 **tank**.rotate(Direction.***NORTH***);  
  
 **tank**.addListener(**new** TankListener());  
  
 **boolean** result = **tank**.move();  
  
 Assertions.*assertFalse*(result);  
 Assertions.*assertSame*(startCell, **tank**.getCell());  
 Assertions.*assertEquals*(0, **events**.size());  
 }  
  
 @Test  
 **public void** move\_noCellInMovingDirection(){  
 Cell startCell = **new** Cell(**new** Position(0, 0));  
  
 startCell.addObject(**tank**);  
 **tank**.rotate(Direction.***NORTH***);  
  
 **tank**.addListener(**new** TankListener());  
  
 **boolean** result = **tank**.move();  
  
 Assertions.*assertFalse*(result);  
 Assertions.*assertSame*(startCell, **tank**.getCell());  
 Assertions.*assertEquals*(0, **events**.size());  
 }  
  
 @Test  
 **public void** shoot\_shootInEmptyCell(){  
 Cell startCell = **field**.getCell(**new** Position(0, 1));  
  
 startCell.addObject(**tank**);  
 **tank**.rotate(Direction.***NORTH***);  
  
 **tank**.addListener(**new** TankListener());  
  
 **boolean** result = **tank**.shoot();  
  
 ArrayList<ObjectInCellEvent> expectedEvents = **new** ArrayList<>();  
 expectedEvents.add(**new** ObjectInCellEvent(**null**, ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
  
 Assertions.*assertEquals*(expectedEvents.size(), **events**.size());  
 **for**(**int** i = 0; i < **events**.size(); i++){  
 Assertions.*assertEquals*(expectedEvents.get(i).getType(), **events**.get(i).getType());  
 Assertions.*assertEquals*(Bullet.**class**, **events**.get(i).getObject().getClass());  
 }  
  
 Assertions.*assertTrue*(result);  
 }  
  
 @Test  
 **public void** shoot\_noCellInShootingDirection(){  
 Cell startCell = **new** Cell(**new** Position(0, 0));  
  
 startCell.addObject(**tank**);  
 **tank**.rotate(Direction.***NORTH***);  
  
 **tank**.addListener(**new** TankListener());  
  
 **boolean** result = **tank**.shoot();  
  
 Assertions.*assertFalse*(result);  
 Assertions.*assertEquals*(0, **events**.size());  
 }  
  
 @Test  
 **public void** pass\_skipStep(){  
 **tank**.addListener(**new** TankListener());  
  
 **tank**.pass();  
  
 Assertions.*assertEquals*(0, **events**.size());  
 }  
  
 @Test  
 **public void** update\_getDamage() {  
 Cell cell = **new** Cell(**new** Position(0, 0));  
 Bullet bullet = **new** BulletForTest();  
  
 **tank**.addListener(**new** TankListener());  
  
 **int** expectedHealth = **tank**.getHealth() - 1;  
  
 cell.addObject(**tank**);  
 cell.addObject(bullet);  
 **tank**.update();  
  
 ArrayList<ObjectInCellEvent> expectedEvents = **new** ArrayList<>();  
 expectedEvents.add(**new** ObjectInCellEvent(**tank**, ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
 expectedEvents.add(**new** ObjectInCellEvent(**tank**, ObjectInCellEvent.EventType.***DAMAGED***));  
  
 Assertions.*assertEquals*(expectedEvents.size(), **events**.size());  
 **for**(**int** i = 0; i < **events**.size(); i++){  
 Assertions.*assertEquals*(expectedEvents.get(i).getType(), **events**.get(i).getType());  
 Assertions.*assertEquals*(expectedEvents.get(i).getObject(), **events**.get(i).getObject());  
 }  
 Assertions.*assertEquals*(expectedHealth, **tank**.getHealth());  
 }  
}

**public class** FuelOilBarrelTest {  
  
 **private** FuelOilBarrel **barrel**;  
  
 **private class** ObjectObserver **implements** IObjectInCellEventListener {  
  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 **events**.add(event);  
 }  
 }  
  
 **private final** ArrayList<ObjectInCellEvent> **events** = **new** ArrayList<>();  
  
 @BeforeEach  
 **public void** testConfiguration(){  
 **barrel** = **new** FuelOilBarrel();  
 **events**.clear();  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithTank(){  
 Tank tank = **new** TankForTest();  
  
 **boolean** result = **barrel**.canFaceWith(tank);  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithObstacle(){  
 Obstacle obstacle = **new** Wall();  
  
 **boolean** result = **barrel**.canFaceWith(obstacle);  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithSameObject(){  
 **boolean** result = **barrel**.canFaceWith(**barrel**);  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithBullet(){  
 Bullet bullet = **new** BulletForTest();  
  
 **boolean** result = **barrel**.canFaceWith(bullet);  
  
 Assertions.*assertTrue*(result);  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithTank(){  
 Tank tank = **new** TankForTest();  
  
 Assertions.*assertThrows*(IllegalArgumentException.**class**, () ->  
 **barrel**.faceWith(tank));  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithObstacle(){  
 Obstacle wall = **new** Wall();  
  
 Assertions.*assertThrows*(IllegalArgumentException.**class**, () ->  
 **barrel**.faceWith(wall));  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithBullet() {  
 Bullet bullet = **new** BulletForTest();  
  
 **barrel**.addListener(**new** ObjectObserver());  
  
 **barrel**.faceWith(bullet);  
  
 ArrayList<ObjectInCellEvent> expectedEvents = **new** ArrayList<>();  
 expectedEvents.add(**new** ObjectInCellEvent(**barrel**, ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
  
 Assertions.*assertEquals*(expectedEvents.size(), **events**.size());  
 **for**(**int** i = 0; i < **events**.size(); i++){  
 Assertions.*assertEquals*(expectedEvents.get(i).getType(), **events**.get(i).getType());  
 Assertions.*assertEquals*(expectedEvents.get(i).getObject(), **events**.get(i).getObject());  
 }  
 }  
  
 @Test  
 **public void** update\_detonating(){  
 Field field = **new** Field(**new** Environment() {  
 @Override  
 **public** Size fieldSize() {  
 **return new** Size(3, 3);  
 }  
  
 @Override  
 **public void** fillField(Field field) {  
  
 }  
 });  
  
 Cell cell = field.getCell(**new** Position(1, 1));  
  
 **barrel**.addListener(**new** ObjectObserver());  
  
 cell.addObject(**barrel**);  
 cell.addObject(**new** BulletForTest());  
 **barrel**.update();  
  
 ArrayList<ObjectInCellEvent> expectedEvents = **new** ArrayList<>();  
 expectedEvents.add(**new** ObjectInCellEvent(**barrel**, ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
 expectedEvents.add(**new** ObjectInCellEvent(**new** Explosion(), ObjectInCellEvent.EventType.***DAMAGED***));  
 expectedEvents.add(**new** ObjectInCellEvent(**new** Explosion(), ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
 expectedEvents.add(**new** ObjectInCellEvent(**new** Explosion(), ObjectInCellEvent.EventType.***DAMAGED***));  
 expectedEvents.add(**new** ObjectInCellEvent(**new** Explosion(), ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
 expectedEvents.add(**new** ObjectInCellEvent(**new** Explosion(), ObjectInCellEvent.EventType.***DAMAGED***));  
 expectedEvents.add(**new** ObjectInCellEvent(**new** Explosion(), ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
 expectedEvents.add(**new** ObjectInCellEvent(**new** Explosion(), ObjectInCellEvent.EventType.***DAMAGED***));  
 expectedEvents.add(**new** ObjectInCellEvent(**new** Explosion(), ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
 expectedEvents.add(**new** ObjectInCellEvent(**barrel**, ObjectInCellEvent.EventType.***DAMAGED***));  
 expectedEvents.add(**new** ObjectInCellEvent(**barrel**, ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
  
 ArrayList<Explosion> explosions = **new** ArrayList<>();  
  
 Assertions.*assertEquals*(expectedEvents.size(), **events**.size());  
 **for**(**int** i = 0; i < **events**.size(); i++){  
 **if** (**events**.get(i).getObject() **instanceof** Explosion explosion){  
 explosions.add(explosion);  
 } **else** {  
 Assertions.*assertEquals*(expectedEvents.get(i).getObject(), **events**.get(i).getObject());  
 }  
 Assertions.*assertEquals*(expectedEvents.get(i).getType(), **events**.get(i).getType());  
 }  
 Assertions.*assertTrue*(**barrel**.isDestroying());  
  
 **for** (Direction direction : Direction.*values*()){  
 Cell neighbour = cell.getNeighbour(direction);  
 Assertions.*assertTrue*(neighbour.getObjects().contains(explosions.get(direction.ordinal() \* 2)));  
 Assertions.*assertEquals*(neighbour, explosions.get(direction.ordinal() \* 2).getCell());  
 }  
 }  
}

**public class** GameTest {  
  
 **private** ArrayList<Game.State> **gameStates** = **new** ArrayList<>();  
  
 **private** ArrayList<ObjectInCellEvent> **objectInCellEvents** = **new** ArrayList<>();  
  
 **private** Game **game**;  
  
 **private class** GameListener **implements** IGameEventListener{  
  
 @Override  
 **public void** onObjectChanged(ObjectInCellEvent event) {  
 **objectInCellEvents**.add(event);  
 }  
  
 @Override  
 **public void** onGameStateChanged(Game.State state) {  
 **gameStates**.add(state);  
 }  
 }  
  
 **private void** waitGame(){  
 **try** {  
 **long** sleepTime = 2000L;  
 Thread.*sleep*(sleepTime);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
  
 @BeforeEach  
 **public void** testConfiguration(){  
 **gameStates**.clear();  
 **objectInCellEvents**.clear();  
  
 **game** = **new** Game(**new** EnvironmentForTest());  
 **game**.addListener(**new** GameListener());  
 **game**.start();  
 }  
  
 @Test  
 **public void** tankRotation\_success(){  
 Player activePlayer = **game**.activePlayer();  
 activePlayer.rotate(Direction.***WEST***);  
  
 ArrayList<ObjectInCellEvent> expectedObjectEvents = **new** ArrayList<>();  
 expectedObjectEvents.add(**new** ObjectInCellEvent(activePlayer.getTank(), ObjectInCellEvent.EventType.***MOVED***));  
  
 Assertions.*assertEquals*(expectedObjectEvents.size(), **objectInCellEvents**.size());  
 **for** (**int** i = 0; i < expectedObjectEvents.size(); i++){  
 Assertions.*assertEquals*(expectedObjectEvents.get(i).getObject().getClass(), **objectInCellEvents**.get(i).getObject().getClass());  
 Assertions.*assertEquals*(expectedObjectEvents.get(i).getType(), **objectInCellEvents**.get(i).getType());  
 }  
  
 Assertions.*assertEquals*(0, **gameStates**.size());  
 }  
  
 @Test  
 **public void** tankRotation\_notSuccess(){  
 Player activePlayer = **game**.activePlayer();  
 activePlayer.rotate(Direction.***NORTH***);  
  
 Assertions.*assertEquals*(0, **objectInCellEvents**.size());  
 Assertions.*assertEquals*(0, **gameStates**.size());  
 }  
  
 @Test  
 **public void** tankMoving\_success(){  
 Player activePlayer = **game**.activePlayer();  
 activePlayer.rotate(Direction.***EAST***);  
 activePlayer.move();  
  
 ArrayList<ObjectInCellEvent> expectedObjectEvents = **new** ArrayList<>();  
 expectedObjectEvents.add(**new** ObjectInCellEvent(activePlayer.getTank(), ObjectInCellEvent.EventType.***MOVED***));  
 expectedObjectEvents.add(**new** ObjectInCellEvent(activePlayer.getTank(), ObjectInCellEvent.EventType.***MOVED***));  
  
 ArrayList<Game.State> expectedGameStates = **new** ArrayList<>();  
 expectedGameStates.add(Game.State.***GAME\_IS\_ON***);  
  
 Assertions.*assertEquals*(expectedObjectEvents.size(), **objectInCellEvents**.size());  
 **for** (**int** i = 0; i < expectedObjectEvents.size(); i++){  
 Assertions.*assertEquals*(expectedObjectEvents.get(i).getObject().getClass(), **objectInCellEvents**.get(i).getObject().getClass());  
 Assertions.*assertEquals*(expectedObjectEvents.get(i).getType(), **objectInCellEvents**.get(i).getType());  
 }  
  
 Assertions.*assertEquals*(expectedGameStates.size(), **gameStates**.size());  
 Assertions.*assertEquals*(expectedGameStates.get(0), **gameStates**.get(0));  
 Assertions.*assertNotSame*(**game**.activePlayer(), activePlayer);  
 }  
  
 @Test  
 **public void** tankMoving\_notSuccess(){  
 Player activePlayer = **game**.activePlayer();  
 activePlayer.move();  
  
 Assertions.*assertEquals*(0, **objectInCellEvents**.size());  
 Assertions.*assertEquals*(0, **gameStates**.size());  
 }  
  
 @Test  
 **public void** tankSkipStep(){  
 Player activePlayer = **game**.activePlayer();  
 activePlayer.pass();  
  
 ArrayList<Game.State> expectedGameStates = **new** ArrayList<>();  
 expectedGameStates.add(Game.State.***GAME\_IS\_ON***);  
  
 Assertions.*assertEquals*(0, **objectInCellEvents**.size());  
  
 Assertions.*assertEquals*(expectedGameStates.size(), **gameStates**.size());  
 Assertions.*assertEquals*(expectedGameStates.get(0), **gameStates**.get(0));  
 Assertions.*assertNotSame*(**game**.activePlayer(), activePlayer);  
 }  
  
 @Test  
 **public void** tankShootingEnemyTank(){  
 Bullet testBullet = **new** Bullet(Direction.***NORTH***, **new** Cell(**new** Position(0, 0)));  
 Player player1 = **game**.activePlayer();  
 Tank tank1 = player1.getTank();  
 player1.rotate(Direction.***SOUTH***);  
 player1.move();  
  
 Player player2 = **game**.activePlayer();  
 Tank tank2 = player2.getTank();  
 player2.move();  
  
 player1.rotate(Direction.***EAST***);  
 player1.shoot();  
 waitGame();  
  
 ArrayList<ObjectInCellEvent> expectedObjectEvents = **new** ArrayList<>();  
 expectedObjectEvents.add(**new** ObjectInCellEvent(tank1, ObjectInCellEvent.EventType.***MOVED***));  
 expectedObjectEvents.add(**new** ObjectInCellEvent(tank1, ObjectInCellEvent.EventType.***MOVED***));  
 expectedObjectEvents.add(**new** ObjectInCellEvent(tank2, ObjectInCellEvent.EventType.***MOVED***));  
 expectedObjectEvents.add(**new** ObjectInCellEvent(tank1, ObjectInCellEvent.EventType.***MOVED***));  
 expectedObjectEvents.add(**new** ObjectInCellEvent(testBullet, ObjectInCellEvent.EventType.***MOVED***));  
 expectedObjectEvents.add(**new** ObjectInCellEvent(testBullet, ObjectInCellEvent.EventType.***MOVED***));  
 expectedObjectEvents.add(**new** ObjectInCellEvent(tank2, ObjectInCellEvent.EventType.***DAMAGED***));  
 expectedObjectEvents.add(**new** ObjectInCellEvent(testBullet, ObjectInCellEvent.EventType.***DESTROYED***));  
  
 ArrayList<Game.State> expectedGameStates = **new** ArrayList<>();  
 expectedGameStates.add(Game.State.***GAME\_IS\_ON***);  
 expectedGameStates.add(Game.State.***GAME\_IS\_ON***);  
 expectedGameStates.add(Game.State.***WINNER\_FOUND***);  
  
 Assertions.*assertEquals*(expectedObjectEvents.size(), **objectInCellEvents**.size());  
 **for** (**int** i = 0; i < expectedObjectEvents.size(); i++){  
 Assertions.*assertEquals*(expectedObjectEvents.get(i).getObject().getClass(), **objectInCellEvents**.get(i).getObject().getClass());  
 Assertions.*assertEquals*(expectedObjectEvents.get(i).getType(), **objectInCellEvents**.get(i).getType());  
 }  
  
 Assertions.*assertEquals*(expectedGameStates.size(), **gameStates**.size());  
 **for** (**int** i = 0; i < expectedGameStates.size(); i++) {  
 Assertions.*assertEquals*(expectedGameStates.get(i), **gameStates**.get(i));  
 }  
 }  
  
 @Test  
 **public void** gameFinished\_tankDestroyEnemyHeadquarters(){  
 Bullet testBullet = **new** Bullet(Direction.***NORTH***, **new** Cell(**new** Position(0, 0)));  
 Player activePlayer = **game**.activePlayer();  
 activePlayer.rotate(Direction.***EAST***);  
 activePlayer.shoot();  
 waitGame();  
  
 ArrayList<ObjectInCellEvent> expectedObjectEvents = **new** ArrayList<>();  
 expectedObjectEvents.add(**new** ObjectInCellEvent(activePlayer.getTank(), ObjectInCellEvent.EventType.***MOVED***));  
 expectedObjectEvents.add(**new** ObjectInCellEvent(testBullet, ObjectInCellEvent.EventType.***MOVED***));  
 expectedObjectEvents.add(**new** ObjectInCellEvent(testBullet, ObjectInCellEvent.EventType.***MOVED***));  
 expectedObjectEvents.add(**new** ObjectInCellEvent(**new** Headquarters(), ObjectInCellEvent.EventType.***DESTROYED***));  
 expectedObjectEvents.add(**new** ObjectInCellEvent(testBullet, ObjectInCellEvent.EventType.***DESTROYED***));  
  
 ArrayList<Game.State> expectedGameStates = **new** ArrayList<>();  
 expectedGameStates.add(Game.State.***WINNER\_FOUND***);  
  
 Assertions.*assertEquals*(expectedObjectEvents.size(), **objectInCellEvents**.size());  
 **for** (**int** i = 0; i < expectedObjectEvents.size(); i++){  
 Assertions.*assertEquals*(expectedObjectEvents.get(i).getObject().getClass(), **objectInCellEvents**.get(i).getObject().getClass());  
 Assertions.*assertEquals*(expectedObjectEvents.get(i).getType(), **objectInCellEvents**.get(i).getType());  
 }  
  
 Assertions.*assertEquals*(expectedGameStates.size(), **gameStates**.size());  
 Assertions.*assertEquals*(expectedGameStates.get(0), **gameStates**.get(0));  
 Assertions.*assertSame*(**game**.winner(), activePlayer);  
 }  
  
 @Test  
 **public void** gameFinished\_draw(){  
 Player player1 = **game**.activePlayer();  
 player1.pass();  
  
 Player player2 = **game**.activePlayer();  
 player2.rotate(Direction.***WEST***);  
 player2.shoot();  
 waitGame();  
  
 ArrayList<Game.State> expectedGameStates = **new** ArrayList<>();  
 expectedGameStates.add(Game.State.***GAME\_IS\_ON***);  
 expectedGameStates.add(Game.State.***DRAW***);  
  
 Assertions.*assertEquals*(expectedGameStates.size(), **gameStates**.size());  
 **for** (**int** i = 0; i < **gameStates**.size(); i++){  
 Assertions.*assertEquals*(expectedGameStates.get(i), **gameStates**.get(i));  
 }  
 Assertions.*assertNull*(**game**.winner());  
 }  
}

**public class** ThicketTest {  
  
 **private class** ThicketListener **implements** IObjectInCellEventListener {  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 **events**.add(event);  
 }  
 }  
  
 **private final** ArrayList<ObjectInCellEvent> **events** = **new** ArrayList<>();  
  
 **private** Thicket **\_thicket**;  
  
 @BeforeEach  
 **public void** testConfiguration(){  
 **\_thicket** = **new** Thicket();  
 **\_thicket**.addListener(**new** ThicketListener());  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithTank(){  
 Tank tank = **new** TankForTest();  
  
 **boolean** result = **\_thicket**.canFaceWith(tank);  
  
 Assertions.*assertTrue*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithObstacle(){  
 Obstacle obstacle = **new** Wall();  
  
 **boolean** result = **\_thicket**.canFaceWith(obstacle);  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithSameObject(){  
 **boolean** result = **\_thicket**.canFaceWith(**\_thicket**);  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithBullet(){  
 Bullet bullet = **new** BulletForTest();  
  
 **boolean** result = **\_thicket**.canFaceWith(bullet);  
  
 Assertions.*assertTrue*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_thicketWithTank(){  
 Tank tank = **new** TankForTest();  
 Tank anotherTank = **new** TankForTest();  
 Cell cell = **new** Cell(**new** Position(0, 0));  
  
 cell.addObject(**\_thicket**);  
 cell.addObject(tank);  
 **\_thicket**.update();  
  
 **boolean** result = **\_thicket**.canFaceWith(anotherTank);  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithTank(){  
 Tank tank = **new** TankForTest();  
  
 **\_thicket**.faceWith(tank);  
  
 Assertions.*assertEquals*(tank, **\_thicket**.getObject());  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithObstacle(){  
 Obstacle wall = **new** Wall();  
  
 Assertions.*assertThrows*(IllegalArgumentException.**class**, () ->  
 **\_thicket**.faceWith(wall));  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithBullet(){  
 Bullet bullet = **new** BulletForTest();  
 ObjectInCellEvent[] actualEvents = {**null**};  
  
 **\_thicket**.addListener(event -> actualEvents[0] = event);  
  
 **\_thicket**.faceWith(bullet);  
  
 Assertions.*assertNull*(actualEvents[0]);  
 }  
  
 @Test  
 **public void** faceWith\_addingTankInCellWithThicket(){  
 Tank tank = **new** TankForTest();  
 Cell cell = **new** Cell(**new** Position(0, 0));  
  
 cell.addObject(**\_thicket**);  
 cell.addObject(tank);  
  
 ArrayList<ObjectInCellEvent> expectedEvents = **new** ArrayList<>();  
 expectedEvents.add(**new** ObjectInCellEvent(**\_thicket**, ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
  
 Assertions.*assertEquals*(expectedEvents.size(), **events**.size());  
 **for**(**int** i = 0; i < **events**.size(); i++){  
 Assertions.*assertEquals*(expectedEvents.get(i).getType(), **events**.get(i).getType());  
 Assertions.*assertEquals*(expectedEvents.get(i).getObject(), **events**.get(i).getObject());  
 }  
 Assertions.*assertEquals*(tank, **\_thicket**.getObject());  
  
 }  
  
 @Test  
 **public void** update\_addingTankInCellWithThicket(){  
 Tank tank = **new** TankForTest();  
 Cell cell = **new** Cell(**new** Position(0, 0));  
  
 cell.addObject(**\_thicket**);  
 cell.addObject(tank);  
 **\_thicket**.update();  
  
 ArrayList<ObjectInCellEvent> expectedEvents = **new** ArrayList<>();  
 expectedEvents.add(**new** ObjectInCellEvent(**\_thicket**, ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
  
 Assertions.*assertEquals*(expectedEvents.size(), **events**.size());  
 **for**(**int** i = 0; i < **events**.size(); i++){  
 Assertions.*assertEquals*(expectedEvents.get(i).getType(), **events**.get(i).getType());  
 Assertions.*assertEquals*(expectedEvents.get(i).getObject(), **events**.get(i).getObject());  
 }  
 Assertions.*assertEquals*(tank, **\_thicket**.getObject());  
 Assertions.*assertFalse*(cell.getObjects().contains(tank));  
 }  
  
 @Test  
 **public void** saveTankFromBullet(){  
 Tank tank = **new** TankForTest();  
 Cell cell = **new** Cell(**new** Position(0, 0));  
 Bullet bullet = **new** BulletForTest();  
  
 cell.addObject(**\_thicket**);  
 cell.addObject(tank);  
 **\_thicket**.update();  
 cell.addObject(bullet);  
  
 Assertions.*assertFalse*(bullet.isDestroying());  
 }  
}

**public class** WallTest {  
  
 **private** Wall **wall**;  
  
 **private class** WallObserver **implements** IObjectInCellEventListener{  
  
 @Override  
 **public void** onObjectInCellAction(ObjectInCellEvent event) {  
 **events**.add(event);  
 }  
 }  
  
 **private final** ArrayList<ObjectInCellEvent> **events** = **new** ArrayList<>();  
  
 @BeforeEach  
 **public void** testConfiguration(){  
 **wall** = **new** Wall();  
 **events**.clear();  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithTank(){  
 Tank tank = **new** TankForTest();  
  
 **boolean** result = **wall**.canFaceWith(tank);  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithObstacle(){  
 Obstacle obstacle = **new** Wall();  
  
 **boolean** result = **wall**.canFaceWith(obstacle);  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithSameObject(){  
 **boolean** result = **wall**.canFaceWith(**wall**);  
  
 Assertions.*assertFalse*(result);  
 }  
  
 @Test  
 **public void** canFaceWith\_collisionWithBullet(){  
 Bullet bullet = **new** BulletForTest();  
  
 **boolean** result = **wall**.canFaceWith(bullet);  
  
 Assertions.*assertTrue*(result);  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithTank(){  
 Tank tank = **new** TankForTest();  
  
 Assertions.*assertThrows*(IllegalArgumentException.**class**, () ->  
 **wall**.faceWith(tank));  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithObstacle(){  
 Obstacle wall = **new** Wall();  
  
 Assertions.*assertThrows*(IllegalArgumentException.**class**, () ->  
 wall.faceWith(wall));  
 }  
  
 @Test  
 **public void** faceWith\_collisionWithBullet() {  
 Bullet bullet = **new** BulletForTest();  
  
 **wall**.addListener(**new** WallObserver());  
  
 **wall**.faceWith(bullet);  
  
 ArrayList<ObjectInCellEvent> expectedEvents = **new** ArrayList<>();  
 expectedEvents.add(**new** ObjectInCellEvent(**wall**, ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
  
 Assertions.*assertEquals*(expectedEvents.size(), **events**.size());  
 **for**(**int** i = 0; i < **events**.size(); i++){  
 Assertions.*assertEquals*(expectedEvents.get(i).getType(), **events**.get(i).getType());  
 Assertions.*assertEquals*(expectedEvents.get(i).getObject(), **events**.get(i).getObject());  
 }  
 }  
  
 @Test  
 **public void** update\_destroying() {  
 Cell cell = **new** Cell(**new** Position(0, 0));  
 Bullet bullet = **new** BulletForTest();  
  
 **wall**.addListener(**new** WallObserver());  
  
 cell.addObject(**wall**);  
 cell.addObject(bullet);  
 **wall**.update();  
  
 ArrayList<ObjectInCellEvent> expectedEvents = **new** ArrayList<>();  
 expectedEvents.add(**new** ObjectInCellEvent(**wall**, ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
 expectedEvents.add(**new** ObjectInCellEvent(**wall**, ObjectInCellEvent.EventType.***DESTROYED***));  
  
 Assertions.*assertEquals*(expectedEvents.size(), **events**.size());  
 **for**(**int** i = 0; i < **events**.size(); i++){  
 Assertions.*assertEquals*(expectedEvents.get(i).getType(), **events**.get(i).getType());  
 Assertions.*assertEquals*(expectedEvents.get(i).getObject(), **events**.get(i).getObject());  
 }  
 Assertions.*assertNull*(**wall**.getCell());  
 }  
  
 @Test  
 **public void** faceWith\_destroyingWall(){  
 Wall wall = **new** Wall();  
 wall.addListener(**new** WallObserver());  
  
 wall.faceWith(**new** Explosion());  
  
 ArrayList<ObjectInCellEvent> expectedEvents = **new** ArrayList<>();  
 expectedEvents.add(**new** ObjectInCellEvent(wall, ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
  
 Assertions.*assertEquals*(expectedEvents.size(), **events**.size());  
 **for**(**int** i = 0; i < **events**.size(); i++){  
 Assertions.*assertEquals*(expectedEvents.get(i).getType(), **events**.get(i).getType());  
 Assertions.*assertEquals*(expectedEvents.get(i).getObject(), **events**.get(i).getObject());  
 }  
 }  
  
 @Test  
 **public void** update\_destroyingWall(){  
 Cell cell = **new** Cell(**new** Position(0, 0));  
 Wall wall = **new** Wall();  
 wall.addListener(**new** WallObserver());  
  
 cell.addObject(wall);  
 cell.addObject(**new** Explosion());  
 wall.update();  
  
 ArrayList<ObjectInCellEvent> expectedEvents = **new** ArrayList<>();  
 expectedEvents.add(**new** ObjectInCellEvent(wall, ObjectInCellEvent.EventType.***NEED\_UPDATE***));  
 expectedEvents.add(**new** ObjectInCellEvent(wall, ObjectInCellEvent.EventType.***DESTROYED***));  
  
 Assertions.*assertEquals*(expectedEvents.size(), **events**.size());  
 **for**(**int** i = 0; i < **events**.size(); i++){  
 Assertions.*assertEquals*(expectedEvents.get(i).getType(), **events**.get(i).getType());  
 Assertions.*assertEquals*(expectedEvents.get(i).getObject(), **events**.get(i).getObject());  
 }  
 Assertions.*assertNull*(wall.getCell());  
 }  
}

# 5. Список используемой литературы

1. Бадд, Т. Объектно-ориентированное программирование в действии / Т. Бадд. - СПб. : Питер, 1997. - 460 с.
2. Шилдг, Герберт. Java 8: руководство для начинающих, 6-е и зд.: Пер. с англ. — М.: ООО “И.Д. Вильямс”, 2015. — 720 с.
3. Портянкин И. А. Swing: Эффектные пользовательские интерфейсы. Библиотека программиста. - СПб.: Питер, 2005. - 336 с.
4. Фаулер М., Скотт К. UML. Основы, 3-е издание. - Пер. с англ. - СПб.: Символ-Плюс, 2004. – 192 с.
5. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Паттерны объектно-ориентированного проектирования. — СПб.: Питер, 2020. — 448 с.: ил. — (Серия «Библиотека программиста»).