Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Волгоградский государственный технический университет»

Факультет электроники и вычислительной техники

Направление 09.03.04 «Программная инженерия»   
Кафедра «Программное обеспечение автоматизированных систем»

Дисциплина «Объектно-ориентированный анализ и программирование»

Утверждаю

и. о. зав. кафедрой \_\_\_\_\_\_\_\_\_\_ Сычев О.А.

**Задание**

**на курсовую работу**

Студент: Рейн В.С.

Группа: ПрИн-366

1. Тема: «Проектирование и реализация программы с использованием объектно-ориентированного подхода» (индивидуальное задание – вариант №17\_02)

Утверждена приказом от «15» января   2024г. № 37-ст

2. Срок представления работы к защите « 04 »   июня  2024 г.

3. Содержание пояснительной записки:

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

4. Перечень графического материала:

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

5. Дата выдачи задания «14» февраля 2024 г.

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

Задание принял к исполнению: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Жадаев М.Ю.

«14» февраля 2024 г.

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Волгоградский государственный технический университет»

Факультет электроники и вычислительной техники

Кафедра «Программное обеспечение автоматизированных систем»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовой работе**

по дисциплине «Объектно-ориентированный анализ и программирование»

на тему: «Проектирование и реализация программы с использованием объектно-ориентированного подхода»

(индивидуальное задание – вариант №11\_02)

Студент: Жадаев М. Ю.

Группа: ПрИн-366

Работа зачтена с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ «    »     июня   2024 г.

Руководитель проекта, нормоконтроллер \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Литовкин Д.В.

Волгоград 2024 г.

Содержание

[1 Формулировка задания 4](#_Toc168266094)

[2 Нефункциональные требования 5](#_Toc168266095)

[3 Первая итерация разработки 6](#_Toc168266096)

[3.1 Формулировка упрощенного варианта задания 6](#_Toc168266097)

[3.2 Функциональные требования (сценарии) 9](#_Toc168266098)

[3.4 Структура программы на уровне классов 10](#_Toc168266100)

[3.5 Типовые процессы в программе 11](#_Toc168266101)

[3.6 Человеко-машинное взаимодействие 13](#_Toc168266102)

[3.7 Реализация ключевых классов 15](#_Toc168266103)

[3.8 Реализация ключевых тестовых случаев 37](#_Toc168266104)

[4 Вторая итерация разработки 43](#_Toc168266105)

[4.1 Функциональные требования (сценарии) 44](#_Toc168266106)

[3.3 Словарь предметной области 45](#_Toc168266107)

[4.3 Структура программы на уровне классов 46](#_Toc168266108)

[4.4 Типовые процессы в программе 47](#_Toc168266109)

[4.5 Человеко-машинное взаимодействие 49](#_Toc168266111)

[4.6 Реализация ключевых классов 51](#_Toc168266112)

[4.7 Реализация ключевых тестовых случаев 92](#_Toc168266113)

[5 Список использованной литературы и других источников 98](#_Toc168266114)

**1 Формулировка задания**

Игра "Танки".

* Ира ведется пошагово двумя  игроками; игрок может пропускать свой ход, т.е. ничего не делать
* На поле NxM клеток находятся танк игрока, танк противника, 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
* Препятствиями могут быть: кирпичная стена, непроходимое и непробиваемое препятствие

Дополнительные требования:

* полет снаряда должен визуализироваться
* разрушение объекта должно визуализироваться

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

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

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

# 2 Нефункциональные требования

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

**3 Первая итерация разработки**

**Игра "Танки". (Упрощенное задание)**

* Ира ведется пошагово двумя  игроками; игрок может пропускать свой ход, т.е. ничего не делать
* На поле NxM клеток находятся танк игрока, танк противника, 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
* Препятствиями могут быть:
  + кирпичная стена, непроходимое и непробиваемое препятствие

Дополнительные требования:

* полет снаряда должен визуализироваться
* разрушение объекта должно визуализироваться

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

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

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

**Функциональные требования (Сценарии)**

**Главный сценарий - победил один из игроков**

1. Игра создает поле с ячейками размера n\*m и отображает его

2. Игра создает танки и стены в ячейках на поле

3. Игра размещает стенки на заданные позиции

4. Игра размещает танки на стартовые позиции

5. Игра назначается начальное количество очков хода танкам

6. Игра делает первый танк текущим

Делать

6.1. Текущий игрок выполняет действие танком

6.1.1 Поворот танка любое количество раз

6.1.1.1 Поворот танка в любое из четырех направлений (север, юг, запад, восток)

6.1.2 Движение танка 1 раз за ход

6.1.2.1. Танк выполняет движение на одну клетку в направлении поворота танка, если в ней не находится стены или танка противника

6.1.3 Выстрел если счетчик ходов танка не меньше N

6.1.3.1 Снаряд совершает движение прямо по направлению поворота танка до встречи с препятствием (стена, край поля, танк противника)

6.1.3.2 Снаряд уничтожается при встрече с препятствием

6.1.3.3 Если снаряд уничтожился в ячейке с танком, то игра считает этот танк разрушенным

6.1.3.4 Счетчик ходов танка обнуляется

6.2. Количество очков хода текущего танка увеличивается на 1

6.3. Если один из танков разрушен и очки его жизней больше 0 , то игра восстанавливает его на стартовую позицию и количество жизней танка уменьшается на единицу

6.5. Игра назначает следующего игрока текущим

Пока у танка противника количество очков жизней больше 0

7. Игра определяет победителем того игрока, у которых останутся жизни.

**Сценарий “Выстрел”**

1. Пользователь даёт команду игрe “Выстрелить”
2. Игра создаёт патрон, передаёт его танку и даёт команду танку “Выстрелить”
3. Танк даёт команду снаряду, двигаться до препятствия
4. Снаряд двигается до тех пор, пока не встретит препятствие или не закончится поле
5. Если снаряд встретил препятствие он уничтожается и даёт команду ячейке, в которой находится препятствие уничтожить его

**Сценарий “Движение танка”**

1. Пользователь даёт команду игре “Двигаться”
2. Игра даёт команду танку “Двигаться”
3. Танк, узнаёт у своей ячейки, есть ли в следующей ячейке unit
4. Если следующая ячейка пуста, то танк перемещается

**Альтернативный сценарий: “**Пользователь инициирует досрочное завершение Игры**”:**

1. **По указанию** Пользователя Игра Завершается.

**Альтернативный сценарий «Пропуск хода»**

6.1.1.Счетчик ходов текущего игрока увеличивается на 1

6.1.2. Ход переходит оппоненту

**Словарь предметной области**

**Словарь:**

**Игра** –управляет игровым циклом и определяет исход игры

**Поле** – контейнер ячеек

**Ячейки** – позиции на поле, на которых расставляются танки и стены и знает о своих соседях со всех сторон, хранит только один объект

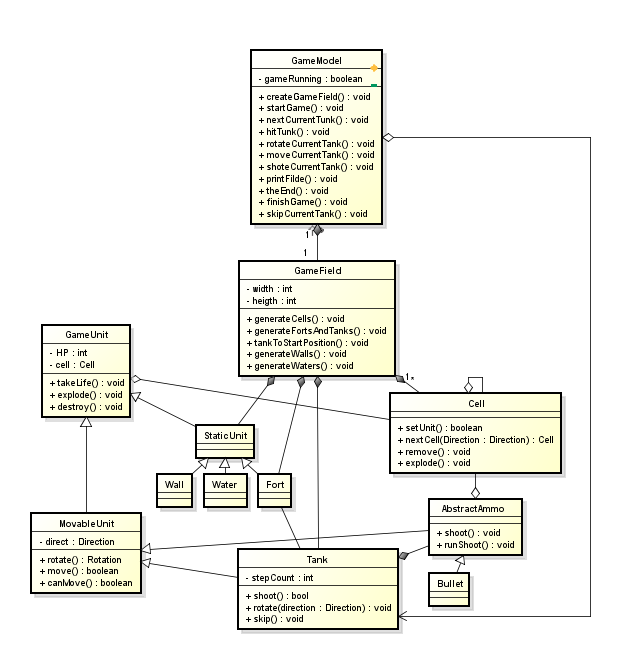
**Танк** –предназначен для уничтожения других танков, управляется игроком(выстрелить, повернуть, передвинуться), имеющая очки жизни, порождающая снаряды и находящийся в ячейке на поле

**Стена** – непроходимое и неуничтожаемое препятствие для танка, находящееся в ячейке на поле

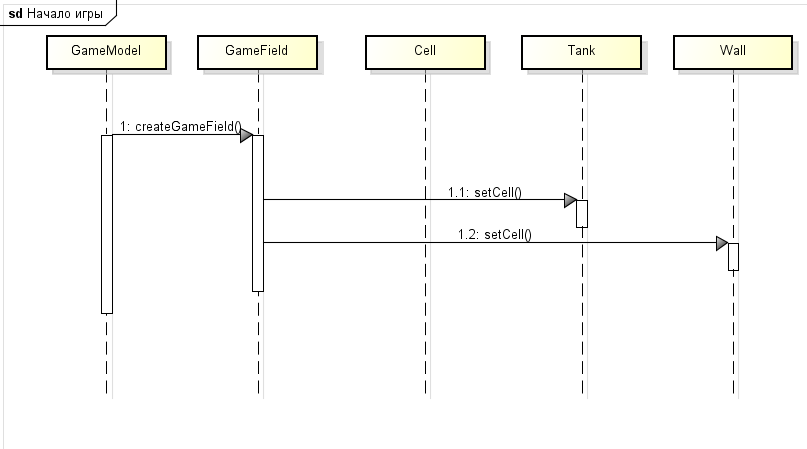
**Снаряд** – предназначен для уничтожения танков при контакте с ними, уничтожается при контакте со стенами или танками.

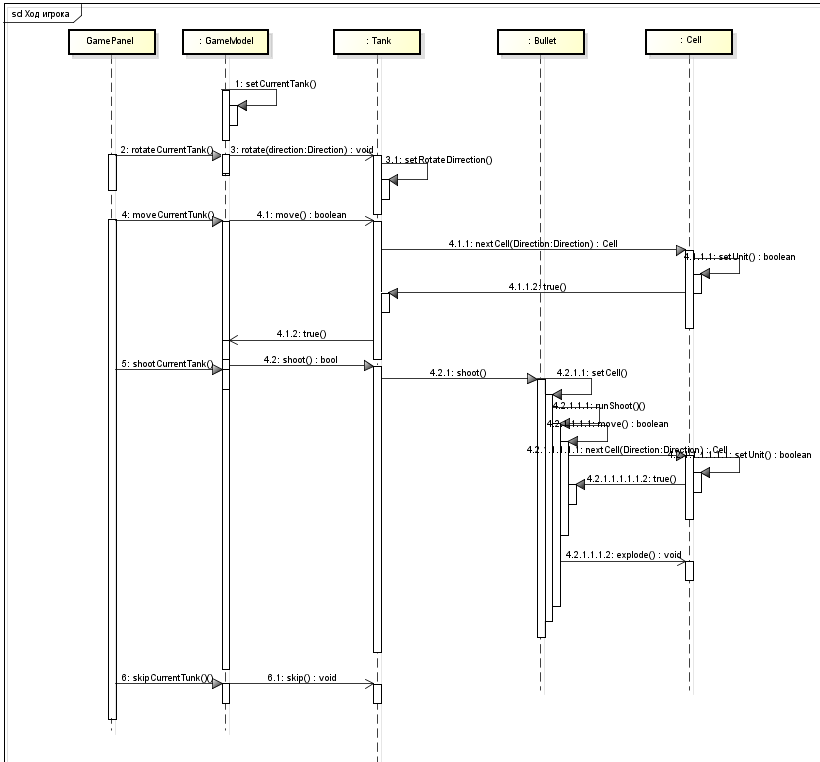
**Игрок** – Пользователь дающий команды исполнения действий танками

**Структура программы на уровне классов**

****

**Типовые процессы в программе**

****

****

**Человеко-машинное взаимодействие**

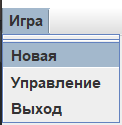
****

Рис. 1. Основное меню

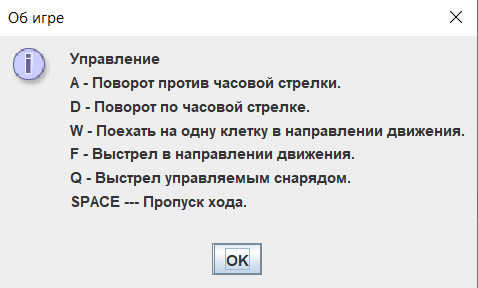
****

Рис. 2. Меню управления

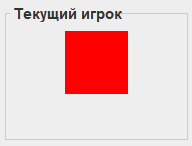


Рис. 3. Индикатор игрока (Красный или Синий)

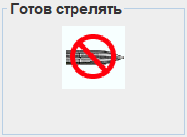


Рис. 4. Показатель готовности текущего танка к стрельбе (не готов)

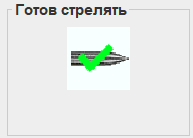


Рис. 5. Показатель готовности текущего танка к стрельбе (готов)

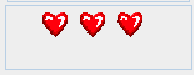


Рис. 6. Показатель кол-ва жизней у текущего танка

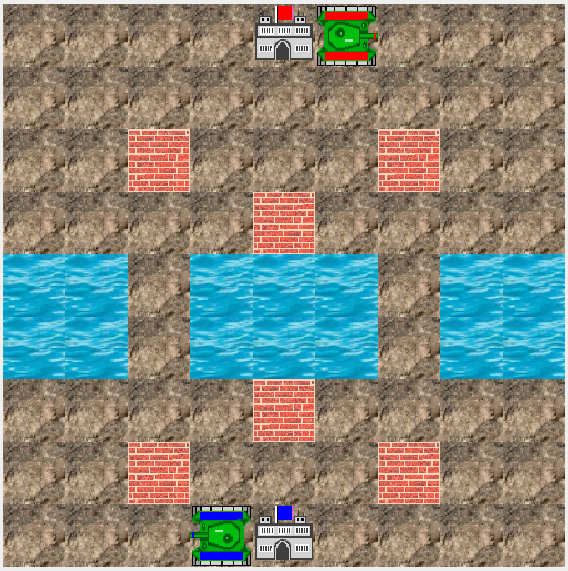


Рис. 7. Основное игровое поле

3.7 Реализация ключевых классов

package units;

import tanks.Cell;

public abstract class GameUnit {

protected int \_HP = 1;//Количество жизней

public static final Integer NOT\_DESTROYED\_AND\_MISS = -2;//Показатель уровня жизни при котором объект является неубиваемым и пропускает снаряды

protected Cell \_cell;

public GameUnit(Cell cell) {

\_cell = cell;

\_cell.setUnit(this);

}

public void setCell(Cell cell){

\_cell.remove(this);

\_cell = cell;

\_cell.setUnit(this);

}

public Cell getCell(){

return \_cell;

}

public int getHP(){

return \_HP;

}

public void takeLife(){

\_HP --;

}

public void explode(){

if(\_HP > 0){

takeLife();

if(\_HP == 0){

destroy();

}

}

}

protected void destroy(){

\_cell.remove(this);

\_cell = null;

}

}

package units;

import Coordination.Direction;

import Coordination.Rotation;

import tanks.Cell;

public abstract class MovableUnit extends GameUnit{

protected Direction \_direct;

public MovableUnit(Cell cell, Direction direction) {

super(cell);

\_direct = direction;

\_HP = 1;

}

public void setDirection(Direction \_direct) {

this.\_direct = \_direct;

}

/\*\*

\*Установить поворот

\*/

public Direction getDirection(){

return new Direction(\_direct.direct());

}

/\*\*

\*Повернуть

\*/

public void rotate(Rotation rotation){

\_direct.rotate(rotation);

}

/\*\*

\*Движение вперед в соответствии с поворотом

\*/

public boolean move(){

Cell newCell = \_cell.nextCell(\_direct);

//Если можно переместиться

if (newCell != null ) {

\_cell.remove(this);

\_cell = newCell;

\_cell.setUnit(this);

return true;

}

return false;

}

/\*\*

\*Возможность движения в двнном направлении

\*/

public boolean canMove(Direction direct){

return \_cell.nextCell(\_direct) != null;

}

}

public abstract class StaticUnit extends GameUnit{

public StaticUnit(Cell cell) {

super(cell);

//Стандартно у статических объектов есть одна жизнь

\_HP = 1;

}

}

public class Tank extends MovableUnit{

private final int COUNT\_STAP\_FOR\_SHOOT = 3; //Кол-во ходов после которых есть возможность сделать выстрел

private int \_stepCount; //Счетчик ходов от последнего выстрела

private Fort \_fort;

public Tank(Cell cell, Direction direction, Fort fort) {

super(cell, direction);

\_HP = 3;

\_stepCount = 0;

\_fort = fort;

\_fort.setTank(this);

}

@Override

public void rotate(Rotation rotation) {

super.rotate(rotation);

InformAboutRotate(this);

}

/\*\*

\* Выстрелить

\* @param ammo снаряд которым происходит выстрел

\* @return

\*/

public boolean shoot(AbstractAmmo ammo) {

if (isReadyShoot()) {

InformAboutFire(this);

ammo.shoot();

\_stepCount = 0;

return true;

}

return false;

}

/\*\*

\* Готовность стрелять

\* @return

\*/

public boolean isReadyShoot(){

return \_stepCount >= COUNT\_STAP\_FOR\_SHOOT;

}

/\*\*

\* Перемещение

\* @return

\*/

@Override

public boolean move() {

//Если можно переместиться

if (canMove(\_direct) && \_cell.nextCell(\_direct).getUnit() == null) {

super.move();

\_stepCount++;

InformAboutMove(this);

return true;

}

return false;

}

/\*\*

\* Пропустить ход

\*/

public void skipStep(){

InformAboutSkip(this);

\_stepCount++;

}

@Override

/\*\*

\* Взрыв танка

\*/

public void explode(){

super.explode();

InformAboutExplosion(this);

}

@Override

/\*\*

\* Уничтожиться

\*/

public void destroy(){

super.destroy();

\_listeners.clear();

}

//События танка

static private ArrayList<TankListener> \_listeners = new ArrayList<TankListener>();

public static void AddListener(TankListener list)

{

\_listeners.add(list);

}

private void InformAboutExplosion(Tank tank)

{

TankEvent event = new TankEvent(this,tank);

try{

for(TankListener i : \_listeners){

i.ExplosiveTank(event);

}

}

catch(ConcurrentModificationException e){}

}

private void InformAboutRotate(Tank tank)

{

TankEvent event = new TankEvent(this,tank);

for(TankListener i : \_listeners){

i.RotateTank(event);

}

}

private void InformAboutMove(Tank tank)

{

TankEvent event = new TankEvent(this,tank);

for(TankListener i : \_listeners){

i.MoveTank(event);

}

}

private void InformAboutSkip(Tank tank)

{

TankEvent event = new TankEvent(this,tank);

for(TankListener i : \_listeners){

i.SkipStep(event);

}

}

private void InformAboutFire(Tank tank)

{

TankEvent event = new TankEvent(this,tank);

for(TankListener i : \_listeners){

i.FireTank(event);

}

}

}

public class Wall extends StaticUnit{

public Wall(Cell cell) {

super(cell);

}

}

package units;

import Events.FortEvent;

import Events.FortListener;

import Events.TankEvent;

import Events.TankListener;

import java.util.ArrayList;

import java.util.ConcurrentModificationException;

import tanks.Cell;

public class Fort extends StaticUnit{

private Tank \_tank;

public Fort(Cell cell) {

super(cell);

}

public Tank getTank(){

return \_tank;

}

public void setTank(Tank tank){

\_tank = tank;

}

@Override

public void explode() {

InformAboutExplosion(this);

super.explode();

}

public void destroy(){

\_listeners.clear();

}

//------Listeners----

//События танка

static private ArrayList<FortListener> \_listeners = new ArrayList<FortListener>();

public static void AddListener(FortListener list)

{

\_listeners.add(list);

}

private void InformAboutExplosion(Fort fort)

{

FortEvent event = new FortEvent(this, fort);

try{

for(FortListener i : \_listeners){

i.FortExplose(event);

}

}

catch(ConcurrentModificationException e){}

}

}

public class Bullet extends AbstractAmmo{

public Bullet(Cell cell, Direction direction, int lengthFly) {

super(cell, direction, lengthFly);

}

@Override

public void runShoot(){

//Если есть следующая ячейка

if(\_cell.nextCell(\_direct) != null){

move();

int countLengthFly = 1;

while((\_cell.getUnit() == null || (\_cell.getUnit() != null && \_cell.getUnit().getHP() == GameUnit.NOT\_DESTROYED\_AND\_MISS)) && \_cell.nextCell(\_direct) != null && countLengthFly<getLengthFly()){

try {

bulletThread.sleep(100);

} catch (InterruptedException ex) {

Logger.getLogger(Bullet.class.getName()).log(Level.SEVERE, null, ex);

}

move();

countLengthFly++;

}

//Взорваться

explode();

}

}

}

public class GameField {

private int width;

private int height;

public GameField(int width, int height) {

this.width = width;

this.height = height;

//Создание ячеек

generateCells();

}

public int width(){

return width;

}

public int height(){

return height;

}

//------------Cells--------------

private Cell[][] field;

/\*\*

\*Сгенерировать матрицу ячеек

\*/

private void generateCells() {

field = new Cell[height][width];

//Заполнение матрицы ячеек

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

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

field[i][j] = new Cell(this);

}

}

//Установка соседей ячеек

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

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

int cnt = 0;

Direction curDir = Direction.Up();

do {

int nextDirX = j + Direction.x[cnt];

int nextDirY = i + Direction.y[cnt];

if (nextDirX >= 0 && nextDirX < width && nextDirY >= 0 && nextDirY < height) {

field[i][j].SetCell(field[nextDirY][nextDirX], curDir);

}

curDir = curDir.rotate(Rotation.Right());

++cnt;

} while (curDir.direct() != Direction.Up().direct());

}

}

}

/\*\*

\*Вернуть ячейку по ее координате на поле

\*/

public Cell getCell(Coordinate coord) {

if (coord.getX() > width || coord.getY() > height) {

return null;

}

return field[coord.getY()-1][coord.getX()-1];

}

/\*\*

\*Вернуть матрицу ячеек

\*/

public Cell[][] getCells(){

return field;

}

/\*\*

\*Вернуть координаты ячейки

\*/

public Coordinate getCoordinateCell(Cell cell){

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

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

if(field[i][j] == cell){

return new Coordinate(j+1, i+1);

}

}

}

return null;

}

//---------------Tanks & Forts------------

private Fort[] \_forts = new Fort[2];

private Tank[] \_tanks = new Tank[2];

private Cell[] \_startPositios = new Cell[2];

public Fort[] get\_forts() {

return \_forts;

}

/\*\*

\*Генерация баз и их танков

\*/

protected void generateFortsAndTanks() {

//Форты

\_forts[0] = new Fort(getCell(new Coordinate(5, 1)));

\_forts[1] = new Fort(getCell(new Coordinate(5, height)));

//Стандартные стартовые позиции танков

\_startPositios[0] = \_forts[0].getCell().nextCell(Direction.Right());

\_startPositios[1] = \_forts[1].getCell().nextCell(Direction.Left());

//Танки

\_tanks[0] = new Tank(\_startPositios[0], Direction.Right(), \_forts[0]);

\_tanks[1] = new Tank(\_startPositios[1], Direction.Left(), \_forts[1]);

}

public Tank[] getTanks() {

return \_tanks;

}

/\*\*

\*Премещение танка к его стартовой позиции

\*/

protected void tankToStartPosition(Tank tank) {

for (int i = 0; i < \_tanks.length; i++) {

if(\_tanks[i]==tank){

tank.setCell(\_startPositios[i]);

}

}

}

//---------------Walls------------

private ArrayList<StaticUnit> \_walls = new ArrayList<>();

protected void generateWalls() {

ArrayList<Coordinate> \_wallsPositions = new ArrayList<>();

\_wallsPositions.add(new Coordinate(3, 3));

\_wallsPositions.add(new Coordinate(7, 3));

\_wallsPositions.add(new Coordinate(3, 8));

\_wallsPositions.add(new Coordinate(7, 8));

\_wallsPositions.add(new Coordinate(5, 4));

\_wallsPositions.add(new Coordinate(5, 7));

for (int i = 0; i < \_wallsPositions.size(); i++) {

\_walls.add(new Wall(getCell(\_wallsPositions.get(i))));

}

}

//--------------Waters------------

private ArrayList<StaticUnit> \_waters = new ArrayList<>();

protected void generateWaters() {

ArrayList<Coordinate> \_watersPositions = new ArrayList<>();

\_watersPositions.add(new Coordinate(9, 5));

\_watersPositions.add(new Coordinate(9, 6));

\_watersPositions.add(new Coordinate(8, 5));

\_watersPositions.add(new Coordinate(8, 6));

\_watersPositions.add(new Coordinate(1, 5));

\_watersPositions.add(new Coordinate(1, 6));

\_watersPositions.add(new Coordinate(2, 5));

\_watersPositions.add(new Coordinate(2, 6));

\_watersPositions.add(new Coordinate(4, 5));

\_watersPositions.add(new Coordinate(4, 6));

\_watersPositions.add(new Coordinate(5, 5));

\_watersPositions.add(new Coordinate(5, 6));

\_watersPositions.add(new Coordinate(6, 5));

\_watersPositions.add(new Coordinate(6, 6));

for (int i = 0; i < \_watersPositions.size(); i++) {

\_waters.add(new Water(getCell(\_watersPositions.get(i))));

}

}

/\*\*

\*Уничтожить поле

\*/

public void destroy(){

//Уничтожить танки

for(Tank tank:\_tanks){

tank.destroy();

}

//Уничтожить форты

for(Fort f:\_forts){

f.destroy();

}

field = null;

\_walls = null;

\_tanks = null;

}

}

public class GameModel {

//------------Field-------------

private GameField \_field;

private Tank \_currentTank;

private boolean \_gameRunning = false;

//Создание игрового поля

private void createGameField() {

\_field = new GameField(9, 9);

\_field.generateFortsAndTanks();

listeningTanksAndForts();

\_currentTank = \_field.getTanks()[0];

\_field.generateWalls();

\_field.generateWaters();

}

public GameField field(){

return \_field;

}

//-----------Game---------------

//Начало игры

public void startGame() {

//Установить индикатор идущей игры

\_gameRunning = true;

InformAboutStartGame("");

createGameField();

}

public Tank getCurrentTank(){

return \_currentTank;

}

public Tank[] getTanks(){

return \_field.getTanks();

}

//Сделать текущим следующий танк

void nextCurrentTank() {

for(int i = 0; i < \_field.getTanks().length; i++){

if(\_currentTank == \_field.getTanks()[i]){

if(i < \_field.getTanks().length-1)

\_currentTank = \_field.getTanks()[i+1];

else

\_currentTank = \_field.getTanks()[0];

InformChangeCurrentTank("");

return;

}

}

}

//Отнять жизнь игрока

private void hitTank(Tank tank) {

if (tank.getHP() > 0) {

\_field.tankToStartPosition(tank);

nextCurrentTank();

printField();

}

else{

theEnd(tank);

}

}

public void rotateCurrentTank(Rotation rotation){

\_currentTank.rotate(rotation);

}

public void moveCurrentTank(){

\_currentTank.move();

}

public void shootCurrentTank(){

Bullet ammo = new Bullet(\_currentTank.getCell(), \_currentTank.getDirection(), 9);

\_currentTank.shoot(ammo);

}

public void skipCurrentTank(){

\_currentTank.skipStep();

}

//Испустить событие изменения ситуации игры

private void printField(){

InformAboutRebuildField();

}

//Конец игры

private void theEnd(Tank luseTank){

String mess = "Выиграл красный танк";

if(luseTank == \_field.getTanks()[0])

mess = "Выиграл синий танк";

InformAboutEndGame(mess);

finishGame();

}

//Прекращение игры

public void finishGame(){

\_gameRunning = false;

if(\_field.getTanks() != null){

\_field.destroy();

\_currentTank = null;

}

}

public boolean gameIsRunning(){

return \_gameRunning;

}

//---------События танка------------

TankListener listenTanks = new TankEventsForModel();

FortListener listenFort = new FortEvents();

private void listeningTanksAndForts(){

Tank.AddListener(listenTanks);

Fort.AddListener(listenFort);

}

public class TankEventsForModel implements TankListener{

@Override

public void ExplosiveTank(TankEvent e){

hitTank(e.\_tank);

System.out.println("hit tank "+e.\_tank);

}

@Override

public void RotateTank(TankEvent e) {

System.out.println("rotate tank "+e.\_tank);

printField();

}

@Override

public void MoveTank(TankEvent e) {

nextCurrentTank();

System.out.println("move tank "+e.\_tank);

printField();

}

@Override

public void SkipStep(TankEvent e) {

System.out.println("skip tank "+e.\_tank);

nextCurrentTank();

}

@Override

public void FireTank(TankEvent e) {

System.out.println("fire tank "+e.\_tank);

nextCurrentTank();

}

}

//-------Fort event-------

public class FortEvents implements FortListener{

@Override

public void FortExplose(FortEvent e) {

theEnd(e.\_fort.getTank());

}

}

//-------Event model------

static private ArrayList<ModelListener> \_listenersRebuild = new ArrayList<ModelListener>();

public static void AddListener(ModelListener list)

{

\_listenersRebuild.add(list);

}

private void InformAboutRebuildField()

{

ModelEvent event = new ModelEvent(this, "");

for(ModelListener i : \_listenersRebuild){

i.RebuildFieldEvent(event);

}

}

private void InformChangeCurrentTank(String mess)

{

ModelEvent event = new ModelEvent(this, mess);

for(ModelListener i : \_listenersRebuild){

i.ChangeCurrentTank(event);

}

}

private void InformAboutStartGame(String mess)

{

ModelEvent event = new ModelEvent(this, mess);

for(ModelListener i : \_listenersRebuild){

i.StartGame(event);

}

}

private void InformAboutEndGame(String mess)

{

ModelEvent event = new ModelEvent(this, mess);

for(ModelListener i : \_listenersRebuild){

i.EndGame(event);

}

}

}

public class GameModel {

//------------Field-------------

private GameField \_field;

private Tank \_currentTank;

private boolean \_gameRunning = false;

//Создание игрового поля

private void createGameField() {

\_field = new GameField(9, 9);

\_field.generateFortsAndTanks();

listeningTanksAndForts();

\_currentTank = \_field.getTanks()[0];

\_field.generateWalls();

\_field.generateWaters();

}

public GameField field(){

return \_field;

}

//-----------Game---------------

//Начало игры

public void startGame() {

//Установить индикатор идущей игры

\_gameRunning = true;

InformAboutStartGame("");

createGameField();

}

public Tank getCurrentTank(){

return \_currentTank;

}

public Tank[] getTanks(){

return \_field.getTanks();

}

//Сделать текущим следующий танк

void nextCurrentTank() {

for(int i = 0; i < \_field.getTanks().length; i++){

if(\_currentTank == \_field.getTanks()[i]){

if(i < \_field.getTanks().length-1)

\_currentTank = \_field.getTanks()[i+1];

else

\_currentTank = \_field.getTanks()[0];

InformChangeCurrentTank("");

return;

}

}

}

//Отнять жизнь игрока

private void hitTank(Tank tank) {

if (tank.getHP() > 0) {

\_field.tankToStartPosition(tank);

nextCurrentTank();

printField();

}

else{

theEnd(tank);

}

}

public void rotateCurrentTank(Rotation rotation){

\_currentTank.rotate(rotation);

}

public void moveCurrentTank(){

\_currentTank.move();

}

public void shootCurrentTank(){

Bullet ammo = new Bullet(\_currentTank.getCell(), \_currentTank.getDirection(), 9);

\_currentTank.shoot(ammo);

}

public void skipCurrentTank(){

\_currentTank.skipStep();

}

//Испустить событие изменения ситуации игры

private void printField(){

InformAboutRebuildField();

}

//Конец игры

private void theEnd(Tank luseTank){

String mess = "Выиграл красный танк";

if(luseTank == \_field.getTanks()[0])

mess = "Выиграл синий танк";

InformAboutEndGame(mess);

finishGame();

}

//Прекращение игры

public void finishGame(){

\_gameRunning = false;

if(\_field.getTanks() != null){

\_field.destroy();

\_currentTank = null;

}

}

public boolean gameIsRunning(){

return \_gameRunning;

}

//---------События танка------------

TankListener listenTanks = new TankEventsForModel();

FortListener listenFort = new FortEvents();

private void listeningTanksAndForts(){

Tank.AddListener(listenTanks);

Fort.AddListener(listenFort);

}

public class TankEventsForModel implements TankListener{

@Override

public void ExplosiveTank(TankEvent e){

hitTank(e.\_tank);

System.out.println("hit tank "+e.\_tank);

}

@Override

public void RotateTank(TankEvent e) {

System.out.println("rotate tank "+e.\_tank);

printField();

}

@Override

public void MoveTank(TankEvent e) {

nextCurrentTank();

System.out.println("move tank "+e.\_tank);

printField();

}

@Override

public void SkipStep(TankEvent e) {

System.out.println("skip tank "+e.\_tank);

nextCurrentTank();

}

@Override

public void FireTank(TankEvent e) {

System.out.println("fire tank "+e.\_tank);

nextCurrentTank();

}

}

//-------Fort event-------

public class FortEvents implements FortListener{

@Override

public void FortExplose(FortEvent e) {

theEnd(e.\_fort.getTank());

}

}

//-------Event model------

static private ArrayList<ModelListener> \_listenersRebuild = new ArrayList<ModelListener>();

public static void AddListener(ModelListener list)

{

\_listenersRebuild.add(list);

}

private void InformAboutRebuildField()

{

ModelEvent event = new ModelEvent(this, "");

for(ModelListener i : \_listenersRebuild){

i.RebuildFieldEvent(event);

}

}

private void InformChangeCurrentTank(String mess)

{

ModelEvent event = new ModelEvent(this, mess);

for(ModelListener i : \_listenersRebuild){

i.ChangeCurrentTank(event);

}

}

private void InformAboutStartGame(String mess)

{

ModelEvent event = new ModelEvent(this, mess);

for(ModelListener i : \_listenersRebuild){

i.StartGame(event);

}

}

private void InformAboutEndGame(String mess)

{

ModelEvent event = new ModelEvent(this, mess);

for(ModelListener i : \_listenersRebuild){

i.EndGame(event);

}

}

}

3 Реализация ключевых тестовых случаев

public class Tests {

@Test

public void TestMoveTank(){

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

Cell cell = tank.getCell();

tank.move();

assertEquals(cell.nextCell(tank.getDirection()), tank.getCell());

}

@Test

public void TestMoveTank1(){

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

Cell cell = tank.getCell();

tank.rotate(Rotation.Right());

tank.move();

assertEquals(cell.nextCell(tank.getDirection()), tank.getCell());

}

@Test

public void TestMoveTank2(){

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

tank.setCell(gameModel.field().getCell(new Coordinate(2, 2)));

Cell cell = tank.getCell();

tank.rotate(Rotation.Left());

tank.move();

assertEquals(cell.nextCell(tank.getDirection()), tank.getCell());

}

@Test

public void TestMoveTank3(){

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

tank.setCell(gameModel.field().getCell(new Coordinate(2, 2)));

Cell cell = tank.getCell();

tank.rotate(Rotation.Right());

tank.rotate(Rotation.Right());

tank.move();

assertEquals(cell.nextCell(tank.getDirection()), tank.getCell());

}

@Test

public void TestMoveTank4(){

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

tank.setCell(gameModel.field().getCell(new Coordinate(2, 2)));

Cell cell = tank.getCell();

tank.rotate(Rotation.Right());

tank.rotate(Rotation.Right());

tank.move();

assertEquals(cell.nextCell(tank.getDirection()), tank.getCell());

}

@Test

public void TestMoveTankToWater(){

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

tank.setCell(gameModel.field().getCell(new Coordinate(1, 4)));

tank.rotate(Rotation.Right());

Cell cell = tank.getCell();

tank.move();

assertEquals(cell, tank.getCell());

}

@Test

public void TestMoveTankToWaterAfterShoot(){

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

tank.setCell(gameModel.field().getCell(new Coordinate(1, 4)));

tank.rotate(Rotation.Right());

Cell cell = tank.getCell().nextCell(tank.getDirection());

tank.shoot(new Bullet(tank.getCell(),tank.getDirection(),9));

assertEquals(cell.getUnit(), tank.getCell().nextCell(tank.getDirection()).getUnit());

}

@Test

public void TestMoveTankToWall(){

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

tank.setCell(gameModel.field().getCell(new Coordinate(3, 2)));

tank.rotate(Rotation.Right());

Cell cell = tank.getCell();

tank.move();

assertEquals(cell, tank.getCell());

}

@Test

public void TestMoveTankToWallAfterShoot() throws InterruptedException {

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

tank.setCell(gameModel.field().getCell(new Coordinate(3, 2)));

tank.rotate(Rotation.Right());

Cell cell = tank.getCell();

tank.skipStep();

tank.skipStep();

tank.skipStep();

tank.shoot(new Bullet(tank.getCell(),tank.getDirection(),9));

Thread.sleep(100);

tank.move();

assertEquals(cell.nextCell(tank.getDirection()), tank.getCell());

}

@Test

public void TestMoveTankToWallAfterShoot2() throws InterruptedException {

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

tank.setCell(gameModel.field().getCell(new Coordinate(3, 2)));

tank.rotate(Rotation.Right());

Cell cell = tank.getCell();

tank.skipStep();

tank.skipStep();

tank.skipStep();

tank.shoot(new Bullet(tank.getCell(),tank.getDirection(),9));

Thread.sleep(100);

tank.move();

assertEquals(cell.nextCell(tank.getDirection()), tank.getCell());

}

@Test

public void TestMoveTankToBound(){

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

tank.setCell(gameModel.field().getCell(new Coordinate(1, 9)));

tank.rotate(Rotation.Right());

Cell cell = tank.getCell();

tank.move();

assertEquals(cell, tank.getCell());

}

@Test

public void TestExplosionWall() throws InterruptedException {

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

tank.setCell(gameModel.field().getCell(new Coordinate(3, 2)));

tank.rotate(Rotation.Right());

tank.skipStep();

tank.skipStep();

tank.skipStep();

tank.shoot(new Bullet(tank.getCell(),tank.getDirection(),9));

Thread.sleep(100);

assertNull(tank.getCell().nextCell(tank.getDirection()).getUnit());

}

@Test

public void TestExplosionWall2() throws InterruptedException {

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

tank.setCell(gameModel.field().getCell(new Coordinate(3, 2)));

tank.rotate(Rotation.Right());

tank.skipStep();

tank.skipStep();

tank.skipStep();

tank.shoot(new Bullet(tank.getCell(),tank.getDirection(),9));

Thread.sleep(100);

assertNull(tank.getCell().nextCell(tank.getDirection()).getUnit());

}

@Test

public void TestExplosionWater() throws InterruptedException {

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

tank.setCell(gameModel.field().getCell(new Coordinate(1, 3)));

tank.rotate(Rotation.Right());

Cell cell = tank.getCell().nextCell(tank.getDirection());

tank.skipStep();

tank.skipStep();

tank.skipStep();

tank.shoot(new Bullet(tank.getCell(),tank.getDirection(),9));

Thread.sleep(100);

assertEquals(tank.getCell().nextCell(tank.getDirection()).getUnit(), cell.getUnit());

}

@Test

public void TestExplosionTank() throws InterruptedException {

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

tank.setCell(gameModel.field().getCell(new Coordinate(4, 8)));

tank.rotate(Rotation.Right());

int hp = tank.getCell().nextCell(tank.getDirection()).getUnit().getHP();

gameModel.skipCurrentTank();

gameModel.skipCurrentTank();

gameModel.skipCurrentTank();

gameModel.skipCurrentTank();

gameModel.skipCurrentTank();

tank.shoot(new Bullet(tank.getCell(),tank.getDirection(),9));

Thread.sleep(100);

assertEquals(hp-1,tank.getCell().nextCell(tank.getDirection()).getUnit().getHP());

}

@Test

public void TestExplosionFort() throws InterruptedException {

gameModel.startGame();

assertTrue(gameModel.gameIsRunning());

boolean a = gameModel.gameIsRunning();

gameModel.field().get\_forts()[0].explode();

assertFalse(gameModel.gameIsRunning());

}

}

**4 Вторая итерация разработки**

**Главный сценарий - победил один из игроков**

1. Игра создает поле с ячейками размера n\*m и отображает его

2. Игра создает танки и стены в ячейках на поле

3. Игра размещает стенки на заданные позиции

4. Игра размещает танки на стартовые позиции

5. Игра назначается начальное количество очков хода танкам

6. Игра делает первый танк текущим

Делать

6.1. Текущий игрок выполняет действие танком

6.1.1 Поворот танка любое количество раз

6.1.1.1 Поворот танка в любое из четырех направлений (север, юг, запад, восток)

6.1.2 Движение танка 1 раз за ход

6.1.2.1. Танк выполняет движение на одну клетку в направлении поворота танка, если в ней не находится стены или танка противника

6.1.3 Выстрел если счетчик ходов танка не меньше N

6.1.3.1 Снаряд совершает движение прямо по направлению поворота танка до встречи с препятствием (стена, край поля, танк противника)

6.1.3.2 Снаряд уничтожается при встрече с препятствием

6.1.3.3 Если снаряд уничтожился в ячейке с танком, то игра считает этот танк разрушенным

6.1.3.4 Счетчик ходов танка обнуляется

6.2. Количество очков хода текущего танка увеличивается на 1

6.3. Если один из танков разрушен и очки его жизней больше 0 , то игра восстанавливает его на стартовую позицию и количество жизней танка уменьшается на единицу

6.5. Игра назначает следующего игрока текущим

Пока у танка противника количество очков жизней больше 0

7. Игра определяет победителем того игрока, у которых останутся жизни.

**Сценарий “Выстрел”**

1. Пользователь даёт команду игрe “Выстрелить”
2. Игра создаёт патрон, передаёт его танку и даёт команду танку “Выстрелить”
3. Танк даёт команду снаряду, двигаться до препятствия
4. Снаряд двигается до тех пор, пока не встретит препятствие или не закончится поле
5. Если снаряд встретил препятствие он уничтожается и даёт команду ячейке, в которой находится препятствие уничтожить его

**Сценарий “Выстрел умным снарядом”**

1. Пользователь даёт команду игрe “Выстрелить”
2. Пользователь выбирает конечную точку полёта снаряда
3. Игра создаёт умный патрон, передаёт его танку и даёт команду танку “Выстрелить”
4. Танк даёт команду снаряду, двигаться до цели
5. Снаряд просчитывает кротчайший путь до цели
6. Снаряд двигается до тех пор, пока не достигнет цели
7. Когда снаряд достигает цели, то ячейка и соседние 4 ячейки взрываются

**Сценарий “Движение танка”**

1. Пользователь даёт команду игре “Двигаться”
2. Игра даёт команду танку “Двигаться”
3. Танк, узнаёт у своей ячейки, есть ли в следующей ячейке unit
4. Если следующая ячейка пуста, то танк перемещается

**Альтернативный сценарий: “**Пользователь инициирует досрочное завершение Игры**”:**

1. **По указанию** Пользователя Игра Завершается.

**Альтернативный сценарий «Пропуск хода»**

6.1.1.Счетчик ходов текущего игрока увеличивается на 1

6.1.2. Ход переходит оппоненту

**Словарь предметной области**

**Словарь:**

**Игра** –управляет игровым циклом и определяет исход игры

**Поле** – контейнер ячеек

**Ячейки** – позиции на поле, на которых расставляются танки и стены и знает о своих соседях со всех сторон, хранит только один объект

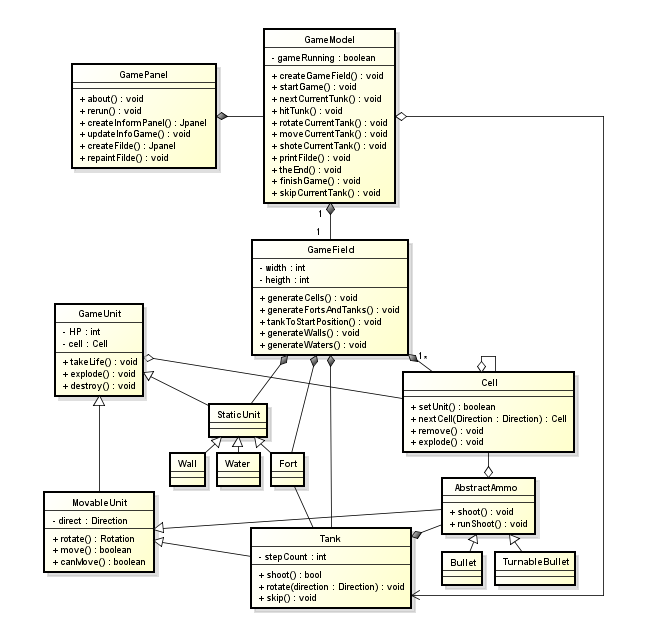
**Танк** –предназначен для уничтожения других танков, управляется игроком(выстрелить, повернуть, передвинуться), имеющая очки жизни, порождающая снаряды и находящийся в ячейке на поле

**Стена** – непроходимое и неуничтожаемое препятствие для танка, находящееся в ячейке на поле

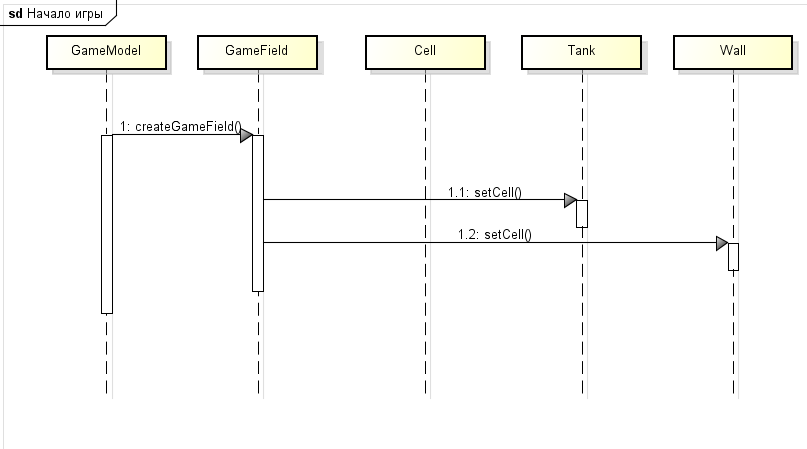
**Снаряд** – предназначен для уничтожения танков при контакте с ними, уничтожается при контакте со стенами или танками.

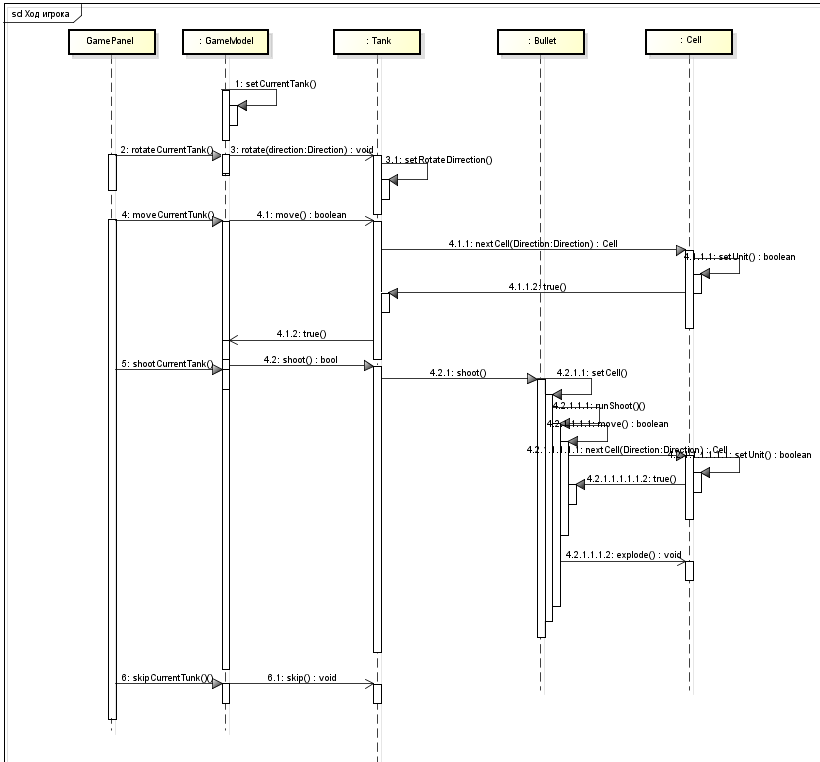
**Игрок** – Пользователь дающий команды исполнения действий танками

**Структура программы на уровне классов**

****

Типовые процессы в программе

****

****

**Человеко-машинное взаимодействие**

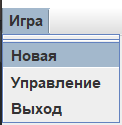
****

Рис. 1. Основное меню

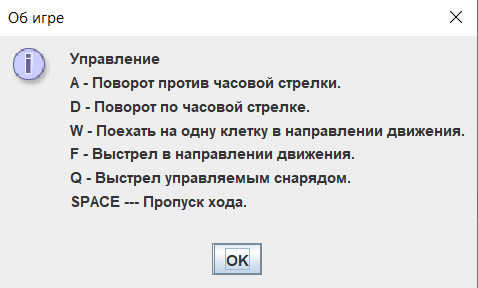
****

Рис. 2. Меню управления

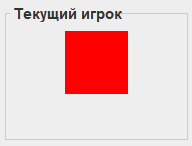


Рис. 3. Индикатор игрока (Красный или Синий)

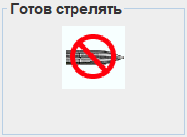


Рис. 4. Показатель готовности текущего танка к стрельбе (не готов)

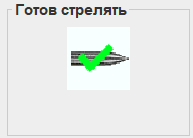


Рис. 5. Показатель готовности текущего танка к стрельбе (готов)

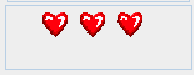


Рис. 6. Показатель кол-ва жизней у текущего танка

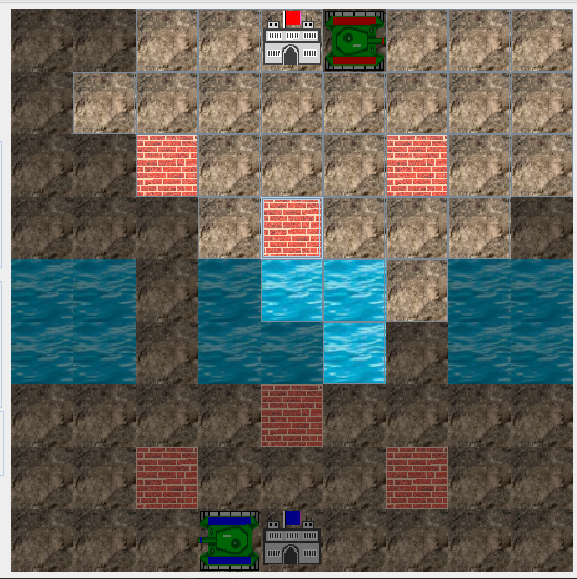


Рис. 7. Игрок выбирает цель для умного снаряда

Реализация ключевых классов

package units;

import tanks.Cell;

public abstract class GameUnit {

protected int \_HP = 1;//Количество жизней

public static final Integer NOT\_DESTROYED\_AND\_MISS = -2;//Показатель уровня жизни при котором объект является неубиваемым и пропускает снаряды

protected Cell \_cell;

public GameUnit(Cell cell) {

\_cell = cell;

\_cell.setUnit(this);

}

public void setCell(Cell cell){

\_cell.remove(this);

\_cell = cell;

\_cell.setUnit(this);

}

public Cell getCell(){

return \_cell;

}

public int getHP(){

return \_HP;

}

public void takeLife(){

\_HP --;

}

public void explode(){

if(\_HP > 0){

takeLife();

if(\_HP == 0){

destroy();

}

}

}

protected void destroy(){

\_cell.remove(this);

\_cell = null;

}

}

package units;

import Coordination.Direction;

import Coordination.Rotation;

import tanks.Cell;

public abstract class MovableUnit extends GameUnit{

protected Direction \_direct;

public MovableUnit(Cell cell, Direction direction) {

super(cell);

\_direct = direction;

\_HP = 1;

}

public void setDirection(Direction \_direct) {

this.\_direct = \_direct;

}

/\*\*

\*Установить поворот

\*/

public Direction getDirection(){

return new Direction(\_direct.direct());

}

/\*\*

\*Повернуть

\*/

public void rotate(Rotation rotation){

\_direct.rotate(rotation);

}

/\*\*

\*Движение вперед в соответствии с поворотом

\*/

public boolean move(){

Cell newCell = \_cell.nextCell(\_direct);

//Если можно переместиться

if (newCell != null ) {

\_cell.remove(this);

\_cell = newCell;

\_cell.setUnit(this);

return true;

}

return false;

}

/\*\*

\*Возможность движения в двнном направлении

\*/

public boolean canMove(Direction direct){

return \_cell.nextCell(\_direct) != null;

}

}

public abstract class StaticUnit extends GameUnit{

public StaticUnit(Cell cell) {

super(cell);

//Стандартно у статических объектов есть одна жизнь

\_HP = 1;

}

}

public class Tank extends MovableUnit{

private final int COUNT\_STAP\_FOR\_SHOOT = 3; //Кол-во ходов после которых есть возможность сделать выстрел

private int \_stepCount; //Счетчик ходов от последнего выстрела

private Fort \_fort;

public Tank(Cell cell, Direction direction, Fort fort) {

super(cell, direction);

\_HP = 3;

\_stepCount = 0;

\_fort = fort;

\_fort.setTank(this);

}

@Override

public void rotate(Rotation rotation) {

super.rotate(rotation);

InformAboutRotate(this);

}

/\*\*

\* Выстрелить

\* @param ammo снаряд которым происходит выстрел

\* @return

\*/

public boolean shoot(AbstractAmmo ammo) {

if (isReadyShoot()) {

InformAboutFire(this);

ammo.shoot();

\_stepCount = 0;

return true;

}

return false;

}

/\*\*

\* Готовность стрелять

\* @return

\*/

public boolean isReadyShoot(){

return \_stepCount >= COUNT\_STAP\_FOR\_SHOOT;

}

/\*\*

\* Перемещение

\* @return

\*/

@Override

public boolean move() {

//Если можно переместиться

if (canMove(\_direct) && \_cell.nextCell(\_direct).getUnit() == null) {

super.move();

\_stepCount++;

InformAboutMove(this);

return true;

}

return false;

}

/\*\*

\* Пропустить ход

\*/

public void skipStep(){

InformAboutSkip(this);

\_stepCount++;

}

@Override

/\*\*

\* Взрыв танка

\*/

public void explode(){

super.explode();

InformAboutExplosion(this);

}

@Override

/\*\*

\* Уничтожиться

\*/

public void destroy(){

super.destroy();

\_listeners.clear();

}

//События танка

static private ArrayList<TankListener> \_listeners = new ArrayList<TankListener>();

public static void AddListener(TankListener list)

{

\_listeners.add(list);

}

private void InformAboutExplosion(Tank tank)

{

TankEvent event = new TankEvent(this,tank);

try{

for(TankListener i : \_listeners){

i.ExplosiveTank(event);

}

}

catch(ConcurrentModificationException e){}

}

private void InformAboutRotate(Tank tank)

{

TankEvent event = new TankEvent(this,tank);

for(TankListener i : \_listeners){

i.RotateTank(event);

}

}

private void InformAboutMove(Tank tank)

{

TankEvent event = new TankEvent(this,tank);

for(TankListener i : \_listeners){

i.MoveTank(event);

}

}

private void InformAboutSkip(Tank tank)

{

TankEvent event = new TankEvent(this,tank);

for(TankListener i : \_listeners){

i.SkipStep(event);

}

}

private void InformAboutFire(Tank tank)

{

TankEvent event = new TankEvent(this,tank);

for(TankListener i : \_listeners){

i.FireTank(event);

}

}

}

public class Wall extends StaticUnit{

public Wall(Cell cell) {

super(cell);

}

}

package units;

import Events.FortEvent;

import Events.FortListener;

import Events.TankEvent;

import Events.TankListener;

import java.util.ArrayList;

import java.util.ConcurrentModificationException;

import tanks.Cell;

public class TurnableBullet extends AbstractAmmo{

public TurnableBullet(Cell cell, Direction direction, int lengthFly) {

super(cell, direction, lengthFly);

RADIUS\_EXPLOSION = 2;

\_maxLegth = 4;

fromCell = cell;

}

@Override

public void runShoot() {

if(traectory!= null){

//Перемещение в соответствии с траекорией

for(Direction d: traectory){

try {

bulletThread.sleep(100);

} catch (InterruptedException ex) {}

setDirection(d);

move();

}

explode();

}

else{

System.err.println("no traectory in "+this );

destroy();

}

}

//---------------------------

//Логика построения траектории

//---------------------------

private Cell fromCell;

private int \_maxLegth;

private ArrayList<ArrayList<Direction> > allPath;

/\*\*

\* Поиск доступных ячеек для снаряда

\* @return

\*/

public ArrayList<Cell> getAvailableCells(){

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

buildAvailableCells(cells, fromCell, \_maxLegth, 0);

return cells;

}

/\*\*

\* Сбор всех доступных для снавряда клеток

\* @param cells общий список доступных ячеек

\* @param curCell текущая яейка

\* @param legth максимальная длина пути

\* @param index текущий индекс

\*/

private void buildAvailableCells(ArrayList<Cell> cells, Cell curCell, int legth, int index){

index++;

for(int i = 0; i<4; i++){

Cell \_currCell = curCell.nextCell(new Direction(i));

//Если есть ячейка

if(\_currCell != null && \_currCell!= fromCell){

//Добавить в список, если ранее не посещалась

if(!cells.contains(\_currCell))

cells.add(\_currCell);

if((\_currCell.getUnit() == null ||(\_currCell.getUnit() != null && \_currCell.getUnit().getHP() == NOT\_DESTROYED\_AND\_MISS ))

&& index<=legth ){

//Продолжаем рекурсию

buildAvailableCells(cells, \_currCell, legth, index);

}

}

}

}

/\*\*

\* Посмтоение наикротчайшего пути

\* @param to ячейка в которую нужно построить путь

\*/

public void buildMinPath(Cell to){

Cell[][] \_cells = \_cell.field().getCells();

int[][] matrix = new int[\_cells.length][\_cells[0].length];//МАтрица кротчайших расстояний от текущей ячейки

Coordinate coordTo = null;

//Создание пустой матрицы

for(int i = 0; i < matrix.length; i++){

for(int j = 0; j < matrix[i].length; j++){

matrix[i][j] = -1;

if(\_cells[i][j]==fromCell)

matrix[i][j] = 0;

else

if(\_cells[i][j]==to)

coordTo = new Coordinate(j, i);

}

}

boolean matrixFilled = false;//Индикатор окончания заполнения матрицы

//Заполнение матрицы кротчайших расстояний

while(!matrixFilled){

matrixFilled = true;

for(int i = 0; i < matrix.length; i++){

for(int j = 0; j < matrix[i].length; j++){

int thisNum = matrix[i][j];

Cell thisCell = \_cells[i][j];

if(thisNum != -1 && thisNum != -2){

for(int k = 0; k < 4; k++){

int x = j;

int y = i;

switch(k){

case 0:{

y--;

} break;

case 1:{

x++;

} break;

case 2:{

y++;

} break;

case 3:{

x--;

} break;

}

if(x>=0 && x < matrix[i].length && y >=0 && y < matrix.length){

int nextNum = thisNum + 1;

//Пометить позицию следующим значением

if(matrix[y][x] > nextNum || matrix[y][x] == -1){

matrix[y][x] = nextNum;

}

//Если стена, пометить позицию -2

if(\_cells[y][x]!=to &&( \_cells[y][x].getUnit() instanceof Wall))

matrix[y][x] = -2;

}

}

}

else if(thisNum == -1)

matrixFilled = false;

}

}

}

int legthPathTo = matrix[coordTo.getY()][coordTo.getX()];

//Координаты целевой ячейки

int x = coordTo.getX();

int y = coordTo.getY();

//Запись кротчайшего пути

for(int currNum = legthPathTo; currNum > 0; currNum--){

for(int k = 0; k < 4; k++){

int xd = x;

int yd = y;

switch(k){

case 0:{

yd--;

break;

}

case 1:{

xd++;

break;

}

case 2:{

yd++;

break;

}

case 3:{

xd--;

break;

}

}

if(xd>=0 && xd < matrix[y].length && yd >=0 && yd < matrix.length){

if(matrix[yd][xd] == currNum-1){

//Так как проход пути идет от обратного, разворачиваем направления и записываем их в начало траектории

traectory.add(0, new Direction(k).Reverse());

x = xd;

y = yd;

//Завершаем пробег по направлениям

k = 4;

}

}

}

}

}

}

public class Fort extends StaticUnit{

private Tank \_tank;

public Fort(Cell cell) {

super(cell);

}

public Tank getTank(){

return \_tank;

}

public void setTank(Tank tank){

\_tank = tank;

}

@Override

public void explode() {

InformAboutExplosion(this);

super.explode();

}

public void destroy(){

\_listeners.clear();

}

//------Listeners----

//События танка

static private ArrayList<FortListener> \_listeners = new ArrayList<FortListener>();

public static void AddListener(FortListener list)

{

\_listeners.add(list);

}

private void InformAboutExplosion(Fort fort)

{

FortEvent event = new FortEvent(this, fort);

try{

for(FortListener i : \_listeners){

i.FortExplose(event);

}

}

catch(ConcurrentModificationException e){}

}

}

public class Bullet extends AbstractAmmo{

public Bullet(Cell cell, Direction direction, int lengthFly) {

super(cell, direction, lengthFly);

}

@Override

public void runShoot(){

//Если есть следующая ячейка

if(\_cell.nextCell(\_direct) != null){

move();

int countLengthFly = 1;

while((\_cell.getUnit() == null || (\_cell.getUnit() != null && \_cell.getUnit().getHP() == GameUnit.NOT\_DESTROYED\_AND\_MISS)) && \_cell.nextCell(\_direct) != null && countLengthFly<getLengthFly()){

try {

bulletThread.sleep(100);

} catch (InterruptedException ex) {

Logger.getLogger(Bullet.class.getName()).log(Level.SEVERE, null, ex);

}

move();

countLengthFly++;

}

//Взорваться

explode();

}

}

}

public class GameField {

private int width;

private int height;

public GameField(int width, int height) {

this.width = width;

this.height = height;

//Создание ячеек

generateCells();

}

public int width(){

return width;

}

public int height(){

return height;

}

//------------Cells--------------

private Cell[][] field;

/\*\*

\*Сгенерировать матрицу ячеек

\*/

private void generateCells() {

field = new Cell[height][width];

//Заполнение матрицы ячеек

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

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

field[i][j] = new Cell(this);

}

}

//Установка соседей ячеек

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

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

int cnt = 0;

Direction curDir = Direction.Up();

do {

int nextDirX = j + Direction.x[cnt];

int nextDirY = i + Direction.y[cnt];

if (nextDirX >= 0 && nextDirX < width && nextDirY >= 0 && nextDirY < height) {

field[i][j].SetCell(field[nextDirY][nextDirX], curDir);

}

curDir = curDir.rotate(Rotation.Right());

++cnt;

} while (curDir.direct() != Direction.Up().direct());

}

}

}

/\*\*

\*Вернуть ячейку по ее координате на поле

\*/

public Cell getCell(Coordinate coord) {

if (coord.getX() > width || coord.getY() > height) {

return null;

}

return field[coord.getY()-1][coord.getX()-1];

}

/\*\*

\*Вернуть матрицу ячеек

\*/

public Cell[][] getCells(){

return field;

}

/\*\*

\*Вернуть координаты ячейки

\*/

public Coordinate getCoordinateCell(Cell cell){

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

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

if(field[i][j] == cell){

return new Coordinate(j+1, i+1);

}

}

}

return null;

}

//---------------Tanks & Forts------------

private Fort[] \_forts = new Fort[2];

private Tank[] \_tanks = new Tank[2];

private Cell[] \_startPositios = new Cell[2];

public Fort[] get\_forts() {

return \_forts;

}

/\*\*

\*Генерация баз и их танков

\*/

protected void generateFortsAndTanks() {

//Форты

\_forts[0] = new Fort(getCell(new Coordinate(5, 1)));

\_forts[1] = new Fort(getCell(new Coordinate(5, height)));

//Стандартные стартовые позиции танков

\_startPositios[0] = \_forts[0].getCell().nextCell(Direction.Right());

\_startPositios[1] = \_forts[1].getCell().nextCell(Direction.Left());

//Танки

\_tanks[0] = new Tank(\_startPositios[0], Direction.Right(), \_forts[0]);

\_tanks[1] = new Tank(\_startPositios[1], Direction.Left(), \_forts[1]);

}

public Tank[] getTanks() {

return \_tanks;

}

/\*\*

\*Премещение танка к его стартовой позиции

\*/

protected void tankToStartPosition(Tank tank) {

for (int i = 0; i < \_tanks.length; i++) {

if(\_tanks[i]==tank){

tank.setCell(\_startPositios[i]);

}

}

}

//---------------Walls------------

private ArrayList<StaticUnit> \_walls = new ArrayList<>();

protected void generateWalls() {

ArrayList<Coordinate> \_wallsPositions = new ArrayList<>();

\_wallsPositions.add(new Coordinate(3, 3));

\_wallsPositions.add(new Coordinate(7, 3));

\_wallsPositions.add(new Coordinate(3, 8));

\_wallsPositions.add(new Coordinate(7, 8));

\_wallsPositions.add(new Coordinate(5, 4));

\_wallsPositions.add(new Coordinate(5, 7));

for (int i = 0; i < \_wallsPositions.size(); i++) {

\_walls.add(new Wall(getCell(\_wallsPositions.get(i))));

}

}

//--------------Waters------------

private ArrayList<StaticUnit> \_waters = new ArrayList<>();

protected void generateWaters() {

ArrayList<Coordinate> \_watersPositions = new ArrayList<>();

\_watersPositions.add(new Coordinate(9, 5));

\_watersPositions.add(new Coordinate(9, 6));

\_watersPositions.add(new Coordinate(8, 5));

\_watersPositions.add(new Coordinate(8, 6));

\_watersPositions.add(new Coordinate(1, 5));

\_watersPositions.add(new Coordinate(1, 6));

\_watersPositions.add(new Coordinate(2, 5));

\_watersPositions.add(new Coordinate(2, 6));

\_watersPositions.add(new Coordinate(4, 5));

\_watersPositions.add(new Coordinate(4, 6));

\_watersPositions.add(new Coordinate(5, 5));

\_watersPositions.add(new Coordinate(5, 6));

\_watersPositions.add(new Coordinate(6, 5));

\_watersPositions.add(new Coordinate(6, 6));

for (int i = 0; i < \_watersPositions.size(); i++) {

\_waters.add(new Water(getCell(\_watersPositions.get(i))));

}

}

/\*\*

\*Уничтожить поле

\*/

public void destroy(){

//Уничтожить танки

for(Tank tank:\_tanks){

tank.destroy();

}

//Уничтожить форты

for(Fort f:\_forts){

f.destroy();

}

field = null;

\_walls = null;

\_tanks = null;

}

}

public class GameModel {

//------------Field-------------

private GameField \_field;

private Tank \_currentTank;

private boolean \_gameRunning = false;

//Создание игрового поля

private void createGameField() {

\_field = new GameField(9, 9);

\_field.generateFortsAndTanks();

listeningTanksAndForts();

\_currentTank = \_field.getTanks()[0];

\_field.generateWalls();

\_field.generateWaters();

}

public GameField field(){

return \_field;

}

//-----------Game---------------

//Начало игры

public void startGame() {

//Установить индикатор идущей игры

\_gameRunning = true;

InformAboutStartGame("");

createGameField();

}

public Tank getCurrentTank(){

return \_currentTank;

}

public Tank[] getTanks(){

return \_field.getTanks();

}

//Сделать текущим следующий танк

void nextCurrentTank() {

for(int i = 0; i < \_field.getTanks().length; i++){

if(\_currentTank == \_field.getTanks()[i]){

if(i < \_field.getTanks().length-1)

\_currentTank = \_field.getTanks()[i+1];

else

\_currentTank = \_field.getTanks()[0];

InformChangeCurrentTank("");

return;

}

}

}

//Отнять жизнь игрока

private void hitTank(Tank tank) {

if (tank.getHP() > 0) {

\_field.tankToStartPosition(tank);

nextCurrentTank();

printField();

}

else{

theEnd(tank);

}

}

public void rotateCurrentTank(Rotation rotation){

\_currentTank.rotate(rotation);

}

public void moveCurrentTank(){

\_currentTank.move();

}

public void shootCurrentTank(){

Bullet ammo = new Bullet(\_currentTank.getCell(), \_currentTank.getDirection(), 9);

\_currentTank.shoot(ammo);

}

public void skipCurrentTank(){

\_currentTank.skipStep();

}

//Испустить событие изменения ситуации игры

private void printField(){

InformAboutRebuildField();

}

//Конец игры

private void theEnd(Tank luseTank){

String mess = "Выиграл красный танк";

if(luseTank == \_field.getTanks()[0])

mess = "Выиграл синий танк";

InformAboutEndGame(mess);

finishGame();

}

//Прекращение игры

public void finishGame(){

\_gameRunning = false;

if(\_field.getTanks() != null){

\_field.destroy();

\_currentTank = null;

}

}

public boolean gameIsRunning(){

return \_gameRunning;

}

//---------События танка------------

TankListener listenTanks = new TankEventsForModel();

FortListener listenFort = new FortEvents();

private void listeningTanksAndForts(){

Tank.AddListener(listenTanks);

Fort.AddListener(listenFort);

}

public class TankEventsForModel implements TankListener{

@Override

public void ExplosiveTank(TankEvent e){

hitTank(e.\_tank);

System.out.println("hit tank "+e.\_tank);

}

@Override

public void RotateTank(TankEvent e) {

System.out.println("rotate tank "+e.\_tank);

printField();

}

@Override

public void MoveTank(TankEvent e) {

nextCurrentTank();

System.out.println("move tank "+e.\_tank);

printField();

}

@Override

public void SkipStep(TankEvent e) {

System.out.println("skip tank "+e.\_tank);

nextCurrentTank();

}

@Override

public void FireTank(TankEvent e) {

System.out.println("fire tank "+e.\_tank);

nextCurrentTank();

}

}

//-------Fort event-------

public class FortEvents implements FortListener{

@Override

public void FortExplose(FortEvent e) {

theEnd(e.\_fort.getTank());

}

}

//-------Event model------

static private ArrayList<ModelListener> \_listenersRebuild = new ArrayList<ModelListener>();

public static void AddListener(ModelListener list)

{

\_listenersRebuild.add(list);

}

private void InformAboutRebuildField()

{

ModelEvent event = new ModelEvent(this, "");

for(ModelListener i : \_listenersRebuild){

i.RebuildFieldEvent(event);

}

}

private void InformChangeCurrentTank(String mess)

{

ModelEvent event = new ModelEvent(this, mess);

for(ModelListener i : \_listenersRebuild){

i.ChangeCurrentTank(event);

}

}

private void InformAboutStartGame(String mess)

{

ModelEvent event = new ModelEvent(this, mess);

for(ModelListener i : \_listenersRebuild){

i.StartGame(event);

}

}

private void InformAboutEndGame(String mess)

{

ModelEvent event = new ModelEvent(this, mess);

for(ModelListener i : \_listenersRebuild){

i.EndGame(event);

}

}

}

public class GameModel {

//------------Field-------------

private GameField \_field;

private Tank \_currentTank;

private boolean \_gameRunning = false;

//Создание игрового поля

private void createGameField() {

\_field = new GameField(9, 9);

\_field.generateFortsAndTanks();

listeningTanksAndForts();

\_currentTank = \_field.getTanks()[0];

\_field.generateWalls();

\_field.generateWaters();

}

public GameField field(){

return \_field;

}

//-----------Game---------------

//Начало игры

public void startGame() {

//Установить индикатор идущей игры

\_gameRunning = true;

InformAboutStartGame("");

createGameField();

}

public Tank getCurrentTank(){

return \_currentTank;

}

public Tank[] getTanks(){

return \_field.getTanks();

}

//Сделать текущим следующий танк

void nextCurrentTank() {

for(int i = 0; i < \_field.getTanks().length; i++){

if(\_currentTank == \_field.getTanks()[i]){

if(i < \_field.getTanks().length-1)

\_currentTank = \_field.getTanks()[i+1];

else

\_currentTank = \_field.getTanks()[0];

InformChangeCurrentTank("");

return;

}

}

}

//Отнять жизнь игрока

private void hitTank(Tank tank) {

if (tank.getHP() > 0) {

\_field.tankToStartPosition(tank);

nextCurrentTank();

printField();

}

else{

theEnd(tank);

}

}

public void rotateCurrentTank(Rotation rotation){

\_currentTank.rotate(rotation);

}

public void moveCurrentTank(){

\_currentTank.move();

}

public void shootCurrentTank(){

Bullet ammo = new Bullet(\_currentTank.getCell(), \_currentTank.getDirection(), 9);

\_currentTank.shoot(ammo);

}

public void skipCurrentTank(){

\_currentTank.skipStep();

}

//Испустить событие изменения ситуации игры

private void printField(){

InformAboutRebuildField();

}

//Конец игры

private void theEnd(Tank luseTank){

String mess = "Выиграл красный танк";

if(luseTank == \_field.getTanks()[0])

mess = "Выиграл синий танк";

InformAboutEndGame(mess);

finishGame();

}

//Прекращение игры

public void finishGame(){

\_gameRunning = false;

if(\_field.getTanks() != null){

\_field.destroy();

\_currentTank = null;

}

}

public boolean gameIsRunning(){

return \_gameRunning;

}

//---------События танка------------

TankListener listenTanks = new TankEventsForModel();

FortListener listenFort = new FortEvents();

private void listeningTanksAndForts(){

Tank.AddListener(listenTanks);

Fort.AddListener(listenFort);

}

public class TankEventsForModel implements TankListener{

@Override

public void ExplosiveTank(TankEvent e){

hitTank(e.\_tank);

System.out.println("hit tank "+e.\_tank);

}

@Override

public void RotateTank(TankEvent e) {

System.out.println("rotate tank "+e.\_tank);

printField();

}

@Override

public void MoveTank(TankEvent e) {

nextCurrentTank();

System.out.println("move tank "+e.\_tank);

printField();

}

@Override

public void SkipStep(TankEvent e) {

System.out.println("skip tank "+e.\_tank);

nextCurrentTank();

}

@Override

public void FireTank(TankEvent e) {

System.out.println("fire tank "+e.\_tank);

nextCurrentTank();

}

}

//-------Fort event-------

public class FortEvents implements FortListener{

@Override

public void FortExplose(FortEvent e) {

theEnd(e.\_fort.getTank());

}

}

//-------Event model------

static private ArrayList<ModelListener> \_listenersRebuild = new ArrayList<ModelListener>();

public static void AddListener(ModelListener list)

{

\_listenersRebuild.add(list);

}

private void InformAboutRebuildField()

{

ModelEvent event = new ModelEvent(this, "");

for(ModelListener i : \_listenersRebuild){

i.RebuildFieldEvent(event);

}

}

private void InformChangeCurrentTank(String mess)

{

ModelEvent event = new ModelEvent(this, mess);

for(ModelListener i : \_listenersRebuild){

i.ChangeCurrentTank(event);

}

}

private void InformAboutStartGame(String mess)

{

ModelEvent event = new ModelEvent(this, mess);

for(ModelListener i : \_listenersRebuild){

i.StartGame(event);

}

}

private void InformAboutEndGame(String mess)

{

ModelEvent event = new ModelEvent(this, mess);

for(ModelListener i : \_listenersRebuild){

i.EndGame(event);

}

}

}

public class GamePanel extends JFrame implements KeyListener{

private JPanel windowPanel = new JPanel();

private final int CELL\_SIZE = 50;

private GameModel \_model;

private JButton[][] \_field;

//Изображения

private BufferedImage \_explosionImg;

private BufferedImage \_cellImg;

private BufferedImage \_tankImg;

private BufferedImage \_wallImg;

private BufferedImage \_bulletImg;

private BufferedImage \_readyShootImg;

private BufferedImage \_notReadyShootImg;

private BufferedImage \_HPImg;

private BufferedImage \_waterImg;

private BufferedImage \_fortImg;

private static final String folder = "./res/"; //Корневая папка ресурсов

public GamePanel(GameModel model) {

super();

\_model = model;

// Листнеры

GameModel.AddListener(new ListenerModel());

addKeyListener(this);

Bullet.AddListener(new ListeningBullet());

this.setTitle("Танки");

//Загрузка изображений

try

{

\_cellImg = ImageIO.read(new File(folder + "Field/Earth.png"));

\_tankImg = ImageIO.read(new File(folder + "Tank1.png"));

\_wallImg = ImageIO.read(new File(folder + "Field/Wall.png"));

\_bulletImg = ImageIO.read(new File(folder + "Bullet.png"));

\_explosionImg = ImageIO.read(new File(folder + "explosion.png"));

\_readyShootImg = ImageIO.read(new File(folder + "ReadyShoot/ReadyShoot.png"));

\_notReadyShootImg = ImageIO.read(new File(folder + "ReadyShoot/NotReadyShoot.png"));

\_HPImg = ImageIO.read(new File(folder + "HP.png"));

\_fortImg = ImageIO.read(new File(folder + "Fort.png"));

\_waterImg = ImageIO.read(new File(folder + "Field/Water.png"));

}

catch(IOException e)

{

e.printStackTrace();

}

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

// Создание панели информации

windowPanel.add(createInformPanel());

// Создание игрового поля

windowPanel.add(createField());

this.add(windowPanel);

//Формирования меню игры

JMenuBar menu = new JMenuBar();

JMenu fileMenu = new JMenu("Игра");

fileMenu.insertSeparator(1);

JMenuItem item1 = new JMenuItem("Новая");

JMenuItem item2 = new JMenuItem("Управление");

JMenuItem item3 = new JMenuItem("Выход");

item3.addActionListener(new GameMenuListener());

item2.addActionListener(new GameMenuListener());

item1.addActionListener(new GameMenuListener());

fileMenu.add(item1);

fileMenu.add(item2);

fileMenu.add(item3);

menu.add(fileMenu);

setJMenuBar(menu);

pack();

setResizable(false);

//Обновления игровой информаци

updateInfoGame();

}

//Окно с информацией об управлении

private void about(){

JOptionPane.showMessageDialog(null, "Управление\n"

+ "A - Поворот против часовой стрелки.\n"

+ "D - Поворот по часовой стрелке.\n"

+ "W - Поехать на одну клетку в направлении движения.\n"

+ "F - Выстрел в направлении движения.\n"

+ "Q - Выстрел управляемым снарядом.\n"

+ "SPACE --- Пропуск хода.\n\n",

"Об игре", JOptionPane.INFORMATION\_MESSAGE);

}

/\*\*

\*Перезапуск игры (новая игра)

\*/

void rerun(){

//удаляем старые связи

removeKeyListener(this);

//Добавляем новые

addKeyListener(this);

//Завершение старой игры

if(\_model.gameIsRunning())

\_model.finishGame();

//Начало новой игры

\_model.startGame();

//Удаление с поля старых элементов

windowPanel.removeAll();

// Создание панели информации

windowPanel.add(createInformPanel());

// Создание игрового поля

windowPanel.add(createField());

pack();

updateInfoGame();

windowPanel.repaint();

}

private class GameMenuListener implements ActionListener {

@Override

public void actionPerformed(ActionEvent e) {

String command = e.getActionCommand();

if ("Управление".equals(command)) {

about();

}

else if ("Новая".equals(command)) {

try{

rerun();

}

catch(Throwable t)

{};

}

else if ("Выход".equals(command)) {

System.exit(0);

}

}

}

/\*\*

\* Сгенерировать панель информации хода игры

\*/

private JPanel createInformPanel()

{

JPanel UpperMenu = new JPanel();

//Панель текущего игрока (цвета)

JPanel ColorPanel = new JPanel();

UpperMenu.setLayout(new BoxLayout(UpperMenu, BoxLayout.Y\_AXIS));

UpperMenu.add(ColorPanel);

UpperMenu.setPreferredSize(new Dimension(150,300));

Font font = new Font("Courier",0,23);

JPanel PointPanel = new JPanel();

UpperMenu.add(PointPanel);

PointPanel.setPreferredSize(new Dimension(130,100));

PointPanel.setBorder(BorderFactory.createTitledBorder("Текущий игрок"));

//Панель готовности к выстрелу

JButton But = new JButton();

PointPanel.add(But);

But.setPreferredSize(new Dimension(50,50));

But.setFocusable(false);

But.setBorderPainted(false);

But.setFont(font);

JPanel CooldownPanel = new JPanel();

UpperMenu.add(CooldownPanel);

CooldownPanel.setPreferredSize(new Dimension(130,100));

CooldownPanel.setBorder(BorderFactory.createTitledBorder("Готов стрелять"));

But = new JButton();

CooldownPanel.add(But);

//Панель жизней

JPanel HPBar = new JPanel();

HPBar.setPreferredSize(new Dimension(100,40));

HPBar.setBorder(BorderFactory.createTitledBorder(""));

But.setPreferredSize(new Dimension(CELL\_SIZE,CELL\_SIZE));

But.setFocusable(false);

But.setBorderPainted(false);

UpperMenu.add(HPBar);

JLabel canv = new JLabel();

canv.setPreferredSize(new Dimension(90,20));

canv.setLocation(new Point(0,0));

HPBar.add(canv);

return UpperMenu;

}

/\*\*

\*Обновление поля иформации

\*/

private void updateInfoGame()

{

JPanel UpperPanel = (JPanel)windowPanel.getComponent(0);

JPanel PointPanel = (JPanel)UpperPanel.getComponent(1);

JPanel CooldownPanel = (JPanel)UpperPanel.getComponent(2);

JPanel HPPanel = (JPanel)UpperPanel.getComponent(3);

JLabel HPConvas = (JLabel)HPPanel.getComponent(0);

JButton But2 = (JButton)PointPanel.getComponent(0);

JButton But3 = (JButton)CooldownPanel.getComponent(0);

Color color = Color.RED;

if(\_model.getCurrentTank()==\_model.getTanks()[1]){

color = Color.BLUE;

}

But2.setEnabled(false);

But2.setBackground(color);

if(\_model.getCurrentTank().isReadyShoot()){

But3.setIcon(new ImageIcon(\_readyShootImg));

}

else

But3.setIcon(new ImageIcon(\_notReadyShootImg));

HPConvas.setIcon(new ImageIcon(getHPImgTank(\_model.getCurrentTank())));

}

/\*\*

\*Создание игрового поля

\*/

private JPanel createField(){

JPanel fieldPanel = new JPanel();

fieldPanel.setDoubleBuffered(true);

fieldPanel.setLayout(new GridLayout(\_model.field().height(), \_model.field().width()));

Dimension fieldDimension = new Dimension(CELL\_SIZE\*\_model.field().height(), CELL\_SIZE\*\_model.field().width());

fieldPanel.setPreferredSize(fieldDimension);

fieldPanel.setMinimumSize(fieldDimension);

fieldPanel.setMaximumSize(fieldDimension);

//Заполнение поля ячейками

fieldPanel.removeAll();

\_field = new JButton[\_model.field().height()][\_model.field().width()];

for (int row = 0; row < \_model.field().height(); row++)

{

for (int col = 0; col < \_model.field().width(); col++)

{

JButton button = new JButton("");

\_field[row][col] = button;

fieldPanel.add(button);

}

}

fieldPanel.validate();

repaintField();

return fieldPanel;

}

/\*\*

\*Перерисовка игрового поля

\*/

private void repaintField() {

for (int row = 0; row < \_model.field().height(); row++){

for (int col = 0; col < \_model.field().width(); col++){

Cell curCell = \_model.field().getCell(new Coordinate(col+1, row+1));

if(timerExplode == null || (timerExplode != null && timerExplode.\_cell != curCell)){

\_field[row][col].setIcon(new ImageIcon(GetCellImage(\_model.field().getCells()[row][col])));

\_field[row][col].removeActionListener(\_field[row][col].getActionListeners().length>0 ? \_field[row][col].getActionListeners()[0] : null);

\_field[row][col].setFocusable(false);

\_field[row][col].setBorderPainted(false);

}

}

}

validate();

}

/\*\*

\*Генерация изображения для определенной ячейки

\*/

private BufferedImage GetCellImage(Cell curCell){

//Создание пустого изображения

BufferedImage cellImg = new BufferedImage(CELL\_SIZE, CELL\_SIZE, BufferedImage.TYPE\_INT\_ARGB);

cellImg.getGraphics().drawImage(\_cellImg, 0, 0, null);

//Отрисовка ячейки в соответствии с находямимися в ней юнитами

if(curCell.getUnit() instanceof Tank){

cellImg.getGraphics().drawImage(getImgTank((Tank)curCell.getUnit()), 0, 0, null);

}

if(curCell.getUnit() instanceof Wall){

cellImg.getGraphics().drawImage(\_wallImg, 0, 0, null);

}

if(curCell.getUnit() instanceof Water){

cellImg.getGraphics().drawImage(\_waterImg, 0, 0, null);

}

if(curCell.getUnit() instanceof Fort){

cellImg.getGraphics().drawImage(getFortImg((Fort) curCell.getUnit()), 0, 0, null);

}

if(curCell.getAmmo()!= null){

cellImg.getGraphics().drawImage(getAmmoImage(curCell.getAmmo()), 0, 0, null);

}

return cellImg;

}

/\*\*

\*Генерация изображения снаряда в соответствии с его состоянием

\*/

private BufferedImage getAmmoImage(AbstractAmmo ammo){

BufferedImage img = new BufferedImage(CELL\_SIZE, CELL\_SIZE, BufferedImage.TYPE\_INT\_ARGB);

Graphics2D g = img.createGraphics();

g.rotate(ammo.getDirection().direct()\*Math.PI/2, 25, 25);

g.drawImage(\_bulletImg, 0, 0, null);

return img;

}

/\*\*

\*Генерация изображения жизней танка

\*/

private BufferedImage getHPImgTank(Tank tank){

BufferedImage tankImg = new BufferedImage(100, 40, BufferedImage.TYPE\_INT\_ARGB);

Graphics2D g = tankImg.createGraphics();

for(int i = 0; i< tank.getHP(); i++){

g.drawImage(\_HPImg, i\*30, 10, null);

}

return tankImg;

}

/\*\*

\*Генерация изображения танка

\*/

private BufferedImage getImgTank(Tank tank){

BufferedImage tankImg = new BufferedImage(CELL\_SIZE, CELL\_SIZE, BufferedImage.TYPE\_INT\_ARGB);

Color color = Color.RED;

if(tank == \_model.getTanks()[1])

color = Color.BLUE;

Graphics2D g = tankImg.createGraphics();

g.rotate(tank.getDirection().direct()\*Math.PI/2, 25, 25);

g.drawImage(\_tankImg, 0, 0, null);

g.setColor(color);

g.fillRect(6, 8, 6, 34);

g.fillRect(38, 8, 6, 34);

g.fillRect(23, 1, 5, 2);

return tankImg;

}

/\*\*

\*Генерация изображения форта в соответствии с цветом его танка

\*/

private BufferedImage getFortImg(Fort fort){

BufferedImage tankImg = new BufferedImage(CELL\_SIZE, CELL\_SIZE, BufferedImage.TYPE\_INT\_ARGB);

Color color = Color.RED;

if(fort.getTank() == \_model.getTanks()[1])

color = Color.BLUE;

Graphics2D g = tankImg.createGraphics();

g.drawImage(\_fortImg, 0, 0, null);

g.setColor(color);

g.fillRect(20, 1, 11, 11);

return tankImg;

}

/\*\*

\*Открыть панель окончания игр

\* @param message сообщение

\*/

private void openEndGame(String message){

JPanel panel = (JPanel)windowPanel;

panel.removeAll();

JLabel \_endGameLabel = new JLabel(message);

\_endGameLabel.setVerticalAlignment(JLabel.CENTER);

\_endGameLabel.setHorizontalAlignment(JLabel.CENTER);

Dimension dimen = new Dimension(panel.getWidth(),panel.getHeight());

\_endGameLabel.setPreferredSize(dimen);

\_endGameLabel.setOpaque(true);

panel.add(\_endGameLabel);

panel.validate();

}

private PaintExploseTimer timerExplode; //Таймер отрисовки взрыва

/\*\*

\*Отрисока взрыва

\*/

private void paintExplose(Cell cell) {

Coordinate coord = \_model.field().getCoordinateCell(cell);

BufferedImage img = new BufferedImage(CELL\_SIZE, CELL\_SIZE, BufferedImage.TYPE\_INT\_ARGB);

Graphics2D g = img.createGraphics();

\_field[coord.getY()-1][coord.getX()-1].getIcon().paintIcon(this, g, 0, 0);

g.drawImage(\_explosionImg, 0, 0, null);

\_field[coord.getY()-1][coord.getX()-1].setIcon(new ImageIcon(img));

validate();

timerExplode = new PaintExploseTimer(300, cell, new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

if(\_model.gameIsRunning()){

//Удалить картинку взрыва

\_field[coord.getY()-1][coord.getX()-1].setIcon(new ImageIcon(GetCellImage(cell)));

timerExplode = null;

}

}

});

timerExplode.setRepeats(false);

timerExplode.start();

}

//Таймер отрисовки взрыва

private class PaintExploseTimer extends Timer{

public Cell \_cell;

public PaintExploseTimer(int delay, Cell cell, ActionListener listener) {

super(delay, listener);

\_cell = cell;

}

}

/\*\*

\*Конец игры

\*/

public void endGame(String message){

removeKeyListener(this);

openEndGame(message);

}

//----------------Observering--------------------

//События модели

public class ListenerModel implements ModelListener{

@Override

public void RebuildFieldEvent(Events.ModelEvent e) {

repaintField();

}

@Override

public void ChangeCurrentTank(ModelEvent e) {

updateInfoGame();

}

@Override

public void StartGame(ModelEvent e) {

}

@Override

public void EndGame(ModelEvent e) {

endGame(e.\_message);

}

}

//----Взрыв-----------

public class ListeningBullet implements BulletListener{

@Override

public void ExplosiveBullet(BulletEvent e) {

System.out.println("explose bullet");

paintExplose(e.\_cell);

}

@Override

public void MoveBullet(BulletEvent e) {

System.out.println("move bullet");

repaintField();

}

@Override

public void StartShootBullet(BulletEvent e) {

System.out.println("start bullet");

}

}

//Обработка нажатий

@Override

public void keyPressed(KeyEvent e)

{

if(\_model.field()!= null && \_model.getCurrentTank()!= null)

{

switch (e.getKeyCode()) {

case KeyEvent.VK\_A:

\_model.rotateCurrentTank(Rotation.Left());

break;

case KeyEvent.VK\_D:

\_model.rotateCurrentTank(Rotation.Right());

break;

case KeyEvent.VK\_W:

\_model.moveCurrentTank();

break;

case KeyEvent.VK\_F:

{

//Если танк готов стрелять

if(\_model.getCurrentTank().isReadyShoot() && \_model.getCurrentTank().getCell().nextCell(\_model.getCurrentTank().getDirection()) != null){

\_model.shootCurrentTank();

}

}

break;

case KeyEvent.VK\_SPACE:

\_model.skipCurrentTank();

break;

case KeyEvent.VK\_Q:

{

//Если танк готов стрелять

if(\_model.getCurrentTank().isReadyShoot()){

//Создание управляемого снаряда

TurnableBullet ammo = new TurnableBullet(\_model.getCurrentTank().getCell(), \_model.getCurrentTank().getDirection(), 9);

//Переход в режим выбора ячейки (цели снаряда)

startChooseCellMode(ammo);

}

}

default:

break;

}

}

}

private boolean runChooseCellMode = false; //Индикатор режима выбора ячейки

private void startChooseCellMode(TurnableBullet bullet){

runChooseCellMode = true;

ArrayList<Cell> cells = bullet.getAvailableCells();//Ячейки которые должны быть активны для выбора

for (int row = 0; row < \_model.field().height(); row++){

for (int col = 0; col < \_model.field().width(); col++){

Cell curCell = \_model.field().getCell(new Coordinate(col+1, row+1));

//Затемнить неакитивные ячейки

if(!cells.contains(curCell)){

BufferedImage img = new BufferedImage(CELL\_SIZE, CELL\_SIZE, BufferedImage.TYPE\_INT\_ARGB);

Graphics2D g = img.createGraphics();

\_field[row][col].getIcon().paintIcon(this, g, 0, 0);

g.setColor(new Color(0,0,0,120));

g.fillRect(0, 0, CELL\_SIZE, CELL\_SIZE);

\_field[row][col].setIcon(new ImageIcon(img));

\_field[row][col].repaint();

}

else{ //Сделать активные ячейки кликабельными

\_field[row][col].setBorderPainted(true);

\_field[row][col].setEnabled(true);

//Добавляем слушателя нажатия ячейки

\_field[row][col].addActionListener(new CellActionListener(curCell, bullet));

}

}

}

validate();

}

public class CellActionListener implements ActionListener {

TurnableBullet \_bullet; //Активный управляемый снаряд

Cell \_cell; //Ячейка нажатия

public CellActionListener(Cell cell, TurnableBullet bullet) {

\_bullet = bullet;

\_cell = cell;

}

public void actionPerformed(ActionEvent e) {

//Построение кротчайшего пути к цели

\_bullet.buildMinPath(\_cell);

//Отключение индикаора режима выбора ячейки

runChooseCellMode = false;

//Выстрел в ячейку, по которой осуществили нажатие

\_model.getCurrentTank().shoot(\_bullet);

repaintField();

}

}

@Override

public void keyTyped(KeyEvent e) {

}

@Override

public void keyReleased(KeyEvent e) {

}

}

3.8 Реализация ключевых тестовых случаев

public class Tests {

@Test

public void TestMoveTank(){

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

Cell cell = tank.getCell();

tank.move();

assertEquals(cell.nextCell(tank.getDirection()), tank.getCell());

}

@Test

public void TestMoveTank1(){

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

Cell cell = tank.getCell();

tank.rotate(Rotation.Right());

tank.move();

assertEquals(cell.nextCell(tank.getDirection()), tank.getCell());

}

@Test

public void TestMoveTank2(){

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

tank.setCell(gameModel.field().getCell(new Coordinate(2, 2)));

Cell cell = tank.getCell();

tank.rotate(Rotation.Left());

tank.move();

assertEquals(cell.nextCell(tank.getDirection()), tank.getCell());

}

@Test

public void TestMoveTank3(){

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

tank.setCell(gameModel.field().getCell(new Coordinate(2, 2)));

Cell cell = tank.getCell();

tank.rotate(Rotation.Right());

tank.rotate(Rotation.Right());

tank.move();

assertEquals(cell.nextCell(tank.getDirection()), tank.getCell());

}

@Test

public void TestMoveTank4(){

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

tank.setCell(gameModel.field().getCell(new Coordinate(2, 2)));

Cell cell = tank.getCell();

tank.rotate(Rotation.Right());

tank.rotate(Rotation.Right());

tank.move();

assertEquals(cell.nextCell(tank.getDirection()), tank.getCell());

}

@Test

public void TestMoveTankToWater(){

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

tank.setCell(gameModel.field().getCell(new Coordinate(1, 4)));

tank.rotate(Rotation.Right());

Cell cell = tank.getCell();

tank.move();

assertEquals(cell, tank.getCell());

}

@Test

public void TestMoveTankToWaterAfterShoot(){

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

tank.setCell(gameModel.field().getCell(new Coordinate(1, 4)));

tank.rotate(Rotation.Right());

Cell cell = tank.getCell().nextCell(tank.getDirection());

tank.shoot(new Bullet(tank.getCell(),tank.getDirection(),9));

assertEquals(cell.getUnit(), tank.getCell().nextCell(tank.getDirection()).getUnit());

}

@Test

public void TestMoveTankToWall(){

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

tank.setCell(gameModel.field().getCell(new Coordinate(3, 2)));

tank.rotate(Rotation.Right());

Cell cell = tank.getCell();

tank.move();

assertEquals(cell, tank.getCell());

}

@Test

public void TestMoveTankToWallAfterShoot() throws InterruptedException {

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

tank.setCell(gameModel.field().getCell(new Coordinate(3, 2)));

tank.rotate(Rotation.Right());

Cell cell = tank.getCell();

tank.skipStep();

tank.skipStep();

tank.skipStep();

tank.shoot(new Bullet(tank.getCell(),tank.getDirection(),9));

Thread.sleep(100);

tank.move();

assertEquals(cell.nextCell(tank.getDirection()), tank.getCell());

}

@Test

public void TestMoveTankToWallAfterShoot2() throws InterruptedException {

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

tank.setCell(gameModel.field().getCell(new Coordinate(3, 2)));

tank.rotate(Rotation.Right());

Cell cell = tank.getCell();

tank.skipStep();

tank.skipStep();

tank.skipStep();

tank.shoot(new Bullet(tank.getCell(),tank.getDirection(),9));

Thread.sleep(100);

tank.move();

assertEquals(cell.nextCell(tank.getDirection()), tank.getCell());

}

@Test

public void TestMoveTankToBound(){

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

tank.setCell(gameModel.field().getCell(new Coordinate(1, 9)));

tank.rotate(Rotation.Right());

Cell cell = tank.getCell();

tank.move();

assertEquals(cell, tank.getCell());

}

@Test

public void TestExplosionWall() throws InterruptedException {

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

tank.setCell(gameModel.field().getCell(new Coordinate(3, 2)));

tank.rotate(Rotation.Right());

tank.skipStep();

tank.skipStep();

tank.skipStep();

tank.shoot(new Bullet(tank.getCell(),tank.getDirection(),9));

Thread.sleep(100);

assertNull(tank.getCell().nextCell(tank.getDirection()).getUnit());

}

@Test

public void TestExplosionWall2() throws InterruptedException {

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

tank.setCell(gameModel.field().getCell(new Coordinate(3, 2)));

tank.rotate(Rotation.Right());

tank.skipStep();

tank.skipStep();

tank.skipStep();

tank.shoot(new Bullet(tank.getCell(),tank.getDirection(),9));

Thread.sleep(100);

assertNull(tank.getCell().nextCell(tank.getDirection()).getUnit());

}

@Test

public void TestExplosionWater() throws InterruptedException {

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

tank.setCell(gameModel.field().getCell(new Coordinate(1, 3)));

tank.rotate(Rotation.Right());

Cell cell = tank.getCell().nextCell(tank.getDirection());

tank.skipStep();

tank.skipStep();

tank.skipStep();

tank.shoot(new Bullet(tank.getCell(),tank.getDirection(),9));

Thread.sleep(100);

assertEquals(tank.getCell().nextCell(tank.getDirection()).getUnit(), cell.getUnit());

}

@Test

public void TestExplosionTank() throws InterruptedException {

gameModel.startGame();

Tank tank = gameModel.getCurrentTank();

tank.setCell(gameModel.field().getCell(new Coordinate(4, 8)));

tank.rotate(Rotation.Right());

int hp = tank.getCell().nextCell(tank.getDirection()).getUnit().getHP();

gameModel.skipCurrentTank();

gameModel.skipCurrentTank();

gameModel.skipCurrentTank();

gameModel.skipCurrentTank();

gameModel.skipCurrentTank();

tank.shoot(new Bullet(tank.getCell(),tank.getDirection(),9));

Thread.sleep(100);

assertEquals(hp-1,tank.getCell().nextCell(tank.getDirection()).getUnit().getHP());

}

@Test

public void TestExplosionFort() throws InterruptedException {

gameModel.startGame();

assertTrue(gameModel.gameIsRunning());

boolean a = gameModel.gameIsRunning();

gameModel.field().get\_forts()[0].explode();

assertFalse(gameModel.gameIsRunning());

}

# 5 Список использованной литературы и других источников

1. Логинова, Ф.С. Объектно-ориентированные методы программирования. [Электронный ресурс] : учеб. пособие — Электрон. дан. — СПб. : ИЭО СПбУТУиЭ, 2012. — 208 с. — Режим доступа: http://e.lanbook.com/book/64040
2. Васильев, А.Н. Самоучитель Java с примерами и программами. [Электронный ресурс] : самоучитель — Электрон. дан. — СПб. : Наука и Техника, 2016. — 368 с. — Режим доступа: http://e.lanbook.com/book/90231
3. Программирование на языке Java. Конспект лекций. [Электронный ресурс] : учеб. пособие / А.В. Гаврилов [и др.]. — Электрон. дан. — СПб.: НИУ ИТМО, 2015. — 126 с. — Режим доступа: http://e.lanbook.com/book/91488