

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 программы с использованием объектно-  
ориентированного подхода»

(индивидуальное задание – вариант №04, подвариант №01)

Студент: Артемов Д.С.

Группа: ПрИн-366

Работа зачтена с оценкой \_\_\_\_\_ « 04 » июня 2020 г.

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

Волгоград 2020 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: Артемов Д.С.  
Группа: ПрИн-366

1. Тема: «Проектирование-программы с использованием объектно-ориентированного подхода» (индивидуальное задание – вариант №04, подвариант №01)

Утверждена приказом от «24» января 2020г. № 101-ст

2. Срок представления работы к защите «04» июня 2020 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 «13» февраля 2020 г.

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

Задание принял к исполнению: —  Артемов Д.С.

«13» февраля 2020 г.

## Содержание

1	Формулировка задания.....	4
2	Нефункциональные требования .....	5
3	Первая итерация разработки.....	6
3.1	Формулировка упрощенного варианта задания.....	6
3.2	Функциональные требования (сценарии) .....	7
3.3	Словарь предметной области.....	10
3.4	Структура программы на уровне классов .....	12
3.5	Типовые процессы в программе .....	13
3.6	Человеко-машинное взаимодействие .....	15
3.7	Реализация ключевых классов.....	18
4.1	Функциональные требования (сценарии) .....	48
4.2	Словарь предметной области.....	51
4.3	Структура программы на уровне классов .....	53
4.4	Типовые процессы в программе .....	54
4.5	Человеко-машинное взаимодействие .....	55
4.6	Реализация ключевых классов.....	59
5	Список использованной литературы и других источников .....	61

## 1 Формулировка задания

Игра "Морской бой".

Правила игры стандартные за исключением следующих:

- размер игрового поля может быть произвольным;
- минимальные требования к интеллекту компьютерного игрока: выстрел осуществляется вероятностным способом в тех клетках, где возможно нахождение кораблей противника. Если корабль обнаружен, то должна использоваться стратегия «добивания» корабля;
- расстановка кораблей выполняется программой.

Дополнительные требования:

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

Вариативность:

- возможны другие типы кораблей, отличающиеся от стандартных формой и способом потопления. Типы кораблей должны быть визуально различимы.

НЕ изменяя ранее созданные классы, а используя точки расширения, реализовать: подводную лодку, у которой над водой видна только рубка (одна клетка). Сразу подбить можно только рубку, остальную часть подводной лодки можно подбить только при повторном попадании.

## 2 Нефункциональные требования

1. Программа должна быть реализована на языке Java SE 8 с использованием стандартных библиотек, в том числе, библиотеки Swing.
2. Форматирование исходного кода программы должно соответствовать Java Code Conventions, September 12, 1997.

### 3 Первая итерация разработки

#### 3.1 Формулировка упрощенного варианта задания

Стандартные правила игры «Морской бой»:

- игра происходит на двух квадратных игровых полях 10x10, разделенных на ячейки;
- на каждом поле расположены корабли в следующем составе: 4 однопалубных, 3 двухпалубных, 2 трехпалубных и 1 четырехпалубный;
- ячейки, содержащие палубы одного корабля, должны образовывать непрерывную линию по горизонтали или по вертикали;
- между соседними кораблями должен быть зазор минимум в одну ячейку;
- ячейки, не содержащие палубу какого-либо корабля, - море;
- в одно из полей стреляет пользователь, в другое – ИИ;
- если при выстреле игрок попал по палубе корабля, то он стреляет еще раз. В противном случае, ход переходит другому игроку;
- ИИ осуществляет выстрел по клеткам, где возможно нахождение кораблей противника. Если корабль обнаружен, то должна использоваться стратегия «добивания» корабля;
- если при выстреле была потоплена последняя из палуб корабля (корабль «убит»), то область вокруг корабля в радиусе 1 клетки автоматически отмечается открытой;
- игра завершается, когда на одном из полей не осталось непотопленных кораблей. Победителем считается игрок, на поле которого корабли еще есть;
- расстановка кораблей осуществляется программой.

## 3.2 Функциональные требования (сценарии)

### 1) Главный успешный сценарий — Победил один из игроков

1 Игрок выбирает режим новой игры

2 Игрок задает размер полей (от 10 до 25)

3 Система расставляет корабли на поле игрока и поле соперника случайным образом, не допуская соприкосновений и пересечений

4 Система назначает игрока-человека текущим

4 Делать

4.1 Делать

Текущий игрок выбирает 1 клетку на поле соперника для выстрела

Пока игрок не промахнулся или игра не завершена

4.2 Система передает ход другому игроку, если игра не завершена, и текущий игрок промахнулся

Пока игра не завершена

5 Система определяет победителем того игрока, у кого остались недобитые корабли

### 1.2) Альтернативный сценарий главного успешного сценария — Досрочное завершение игры (всей программы)

1 Сценарий начинается в любой точке сценария 1), когда игрок инициирует завершение игры

2 Система завершает свою работу без каких-либо запросов и предупреждений

### 1.3) Альтернативный сценарий главного успешного сценария — Переигровка

1 Сценарий начинается в любой точке сценария 1), когда игрок инициирует начало новой игры

2 Если ранее стартовавший сеанс игры еще не выявил победителя

2.1 Система сообщает, что игра еще не завершена и запрашивает у игрока, действительно ли следует начать новый сеанс игры

2.2 Если игрок подтверждает начало новой игры

2.2.1 Система завершает текущий сеанс игры и стартует главный успешный сценарий

Иначе если игрок не желает начинать игру заново

2.2.2 Система продолжает выполнение главного успешного сценария

## **2.1) Система определяет результат выстрела - «Попал»**

1 Система проверяет, есть ли на выбранной клетке палуба

2 Т.к. на клетке есть палуба и не выполнены все условия потопления корабля, палуба изменяет свое состояние на «поврежденная» или «сломанная» в зависимости от вида самой палубы.

3 Система сигнализирует игроку: «Попал»

4 Игрок продолжает свой ход

## **2.2) Система определяет результат выстрела - «Убил»**

1 Система проверяет, есть ли на клетке палуба

2 Т.к. на клетке есть палуба и выполнены все условия потопления каждой палубы и корабля в целом, система сигнализирует игроку: «Убил»

3 Игрок продолжает свой ход

## **2.3) Система определяет результат выстрела - «Промых»**

1 Система проверяет, есть ли на клетке палуба

2 Т.к. клетка не содержит палубы, клетка меняет свое состояние

3 Система сигнализирует игроку о том, что он промахнулся

4 Система передает ход другому игроку



#### **2.4) Система определяет результат выстрела - «Повторный выстрел»**

1 Система проверяет состояние клетки или палубы, находящейся на ней.

2 Т.к. обнаружена сломанная клетка или сломанная палуба, система не засчитывает ход игроку

3 Система предоставляет игроку еще одну попытку выстрела

### 3.3 Словарь предметной области

Палуба — составляющая корабля, занимающая одну клетку

Промех — результат выстрела в клетку поля, не содержащую палубу

Попал — результат выстрела в некоторую палубу, означающий полное ее разрушение

Убил — результат полного разрушения всех палуб, входящих в данный корабль

Состояния клетки:

- Целая клетка — клетка поля, в которую еще ни разу не стреляли
- Сломанная клетка — клетка поля, в которую либо уже стреляли, либо уже открыли после потопления находящегося по соседству корабля

Состояния части корабля:

- Целая — палуба, в которую еще ни разу не стреляли
- Сломанная — палуба, в которую стреляли необходимое количество раз для ее потопления

Состояния корабля:

- Активный — еще не потопленный корабль: хотя бы одна его часть не сломанная
- Потопленный — все его части сломанные

Стандартный режим игры — используются только стандартные корабли

Нестандартный режим игры — все стандартные корабли заменяются на выбранные случайным образом нестандартные корабли из списка кораблей.

По соседству — нахождение объекта (палубы, корабля, клетки) в 1 из 8 клеток вокруг данной клетки.

Игрок — игрок-человек или игрок-бот

Поле игрока – поле игрока-человека, содержащее стандартные или нестандартные корабли (в зависимости от режима игры). Обстреливается игроком-ботом

Поле соперника – поле игрока-бота, стандартные или нестандартные корабли (в зависимости от режима игры). Обстреливается игроком-человеком.

### 3.4 Структура программы на уровне классов

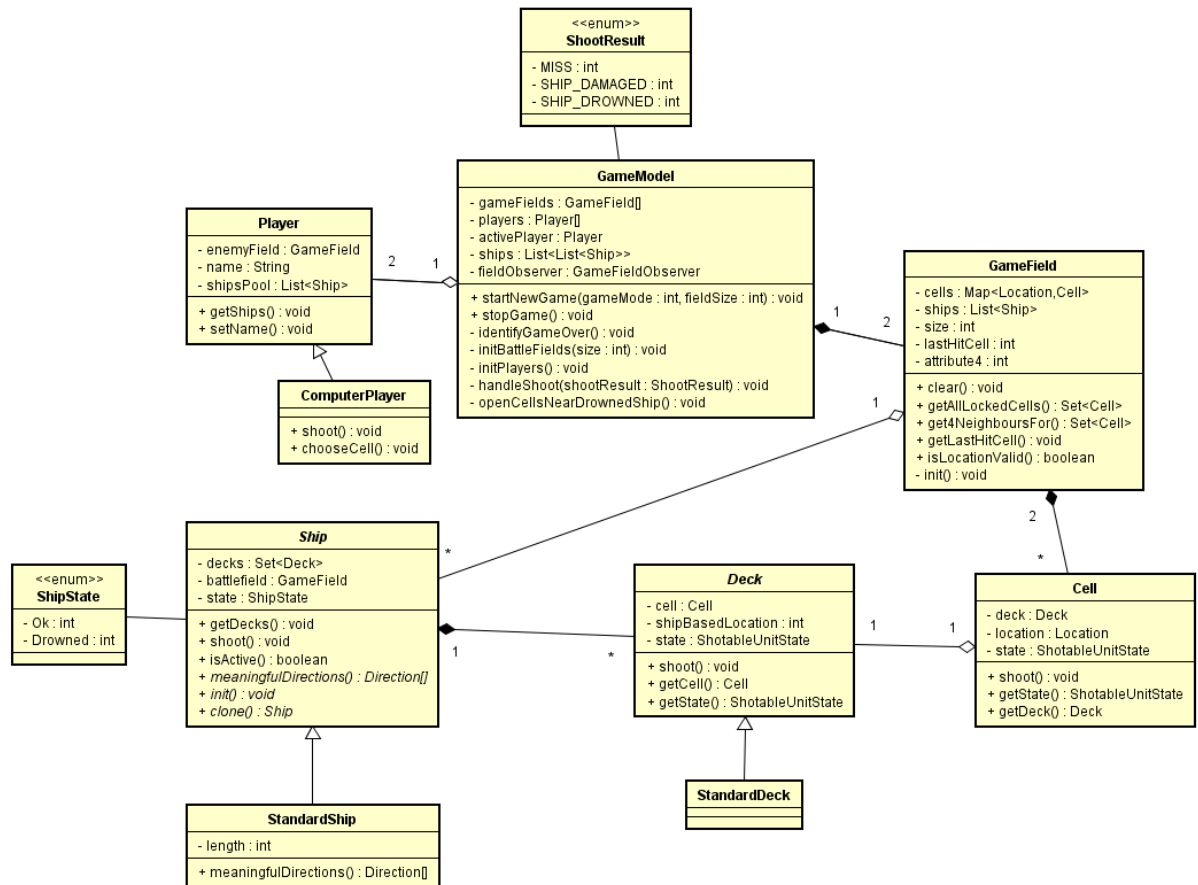


Диаграмма классов вычислительной модели

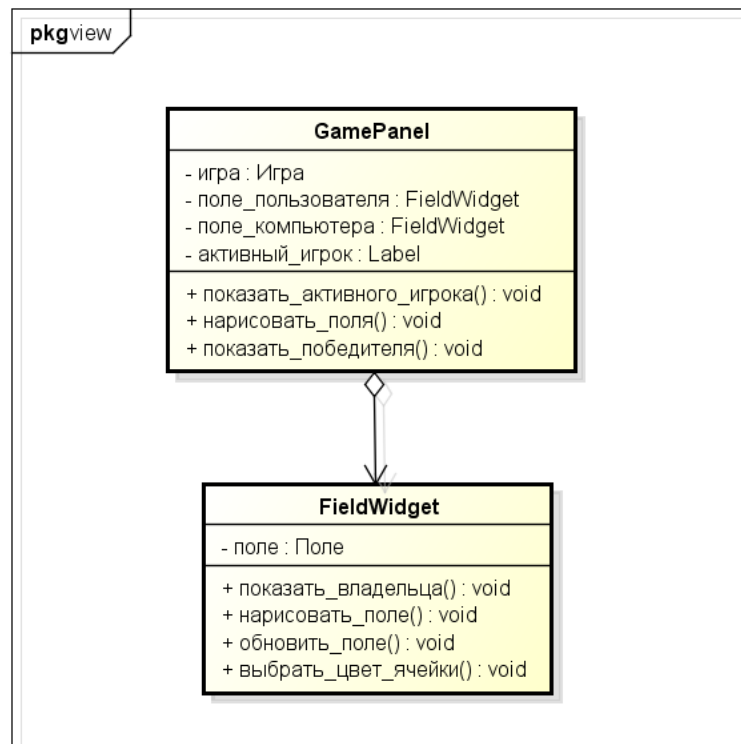
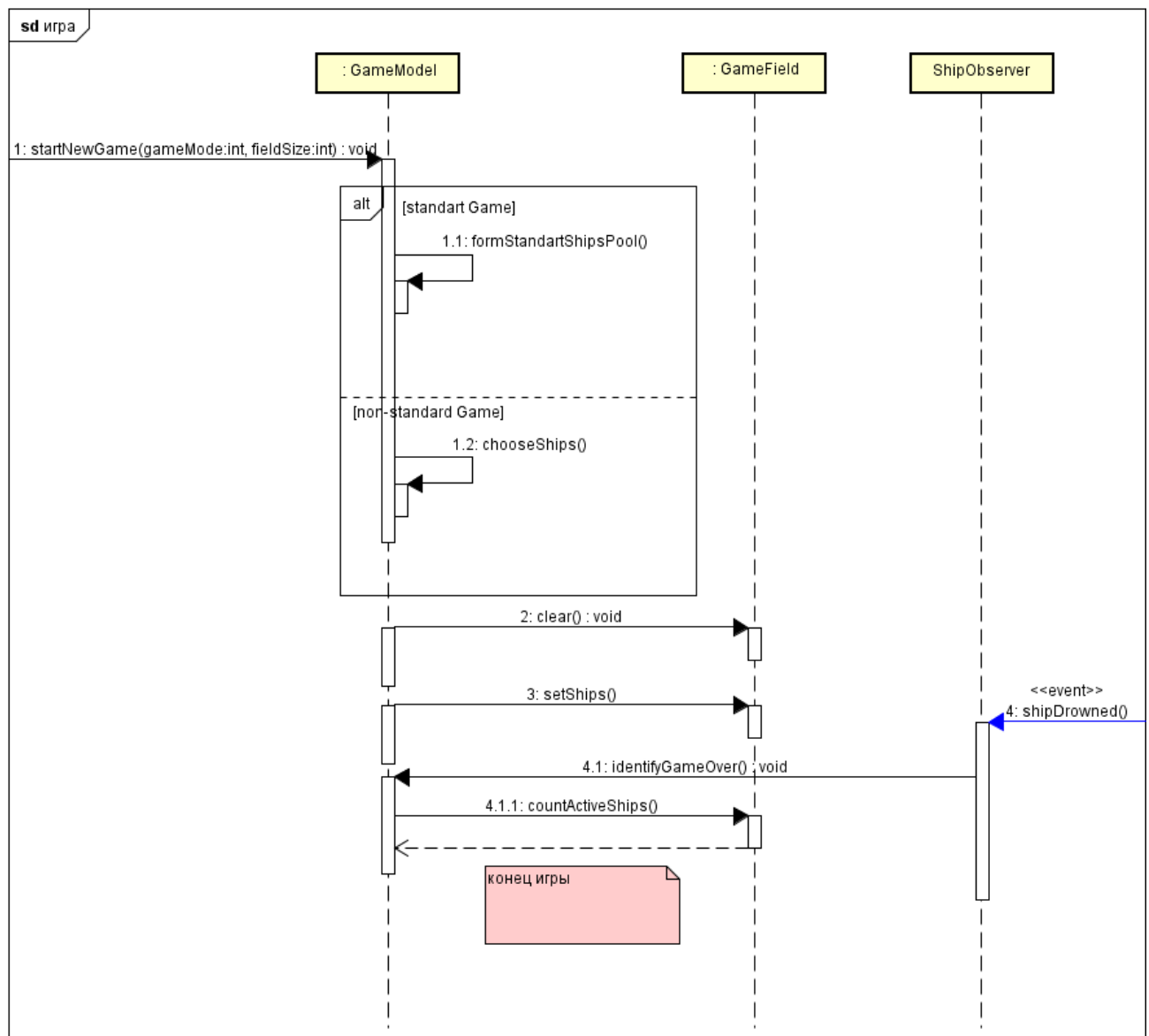
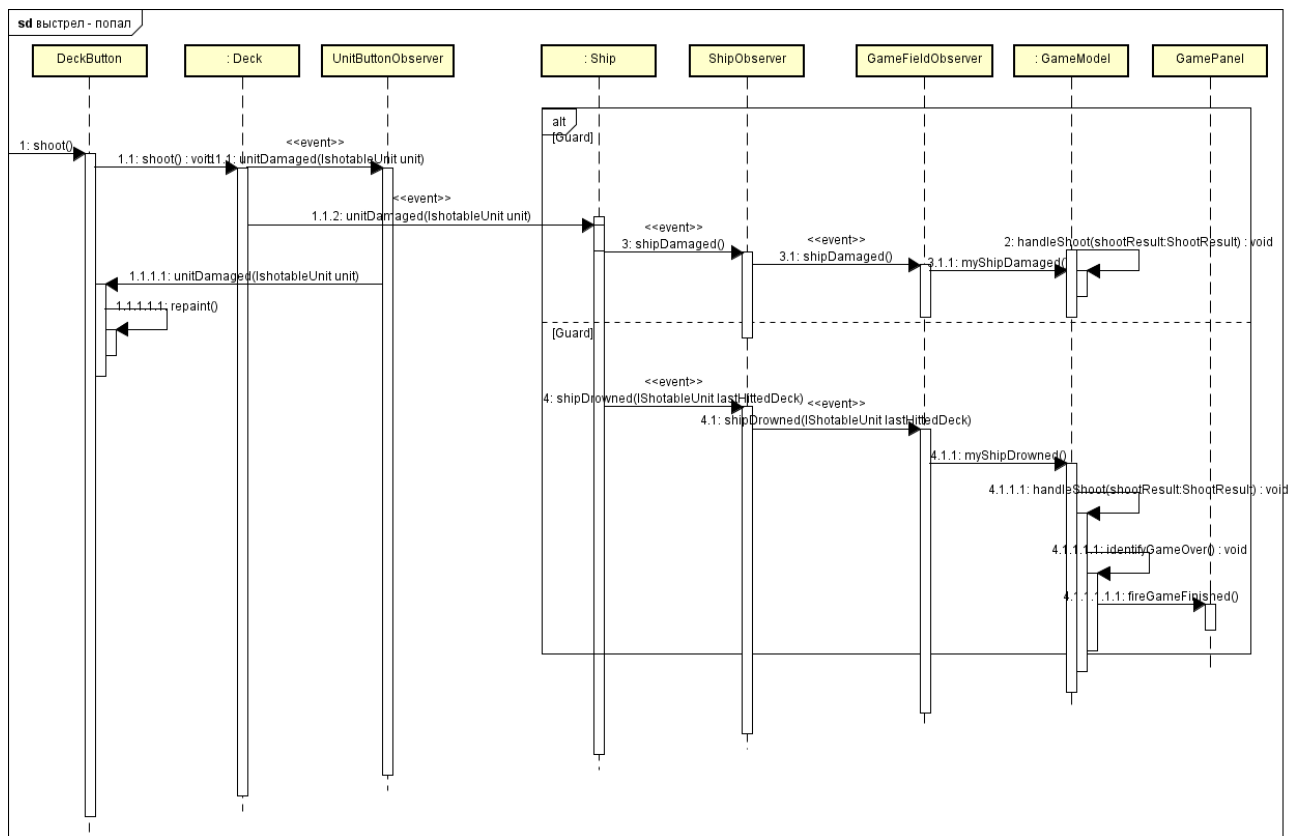


Диаграмма классов представления

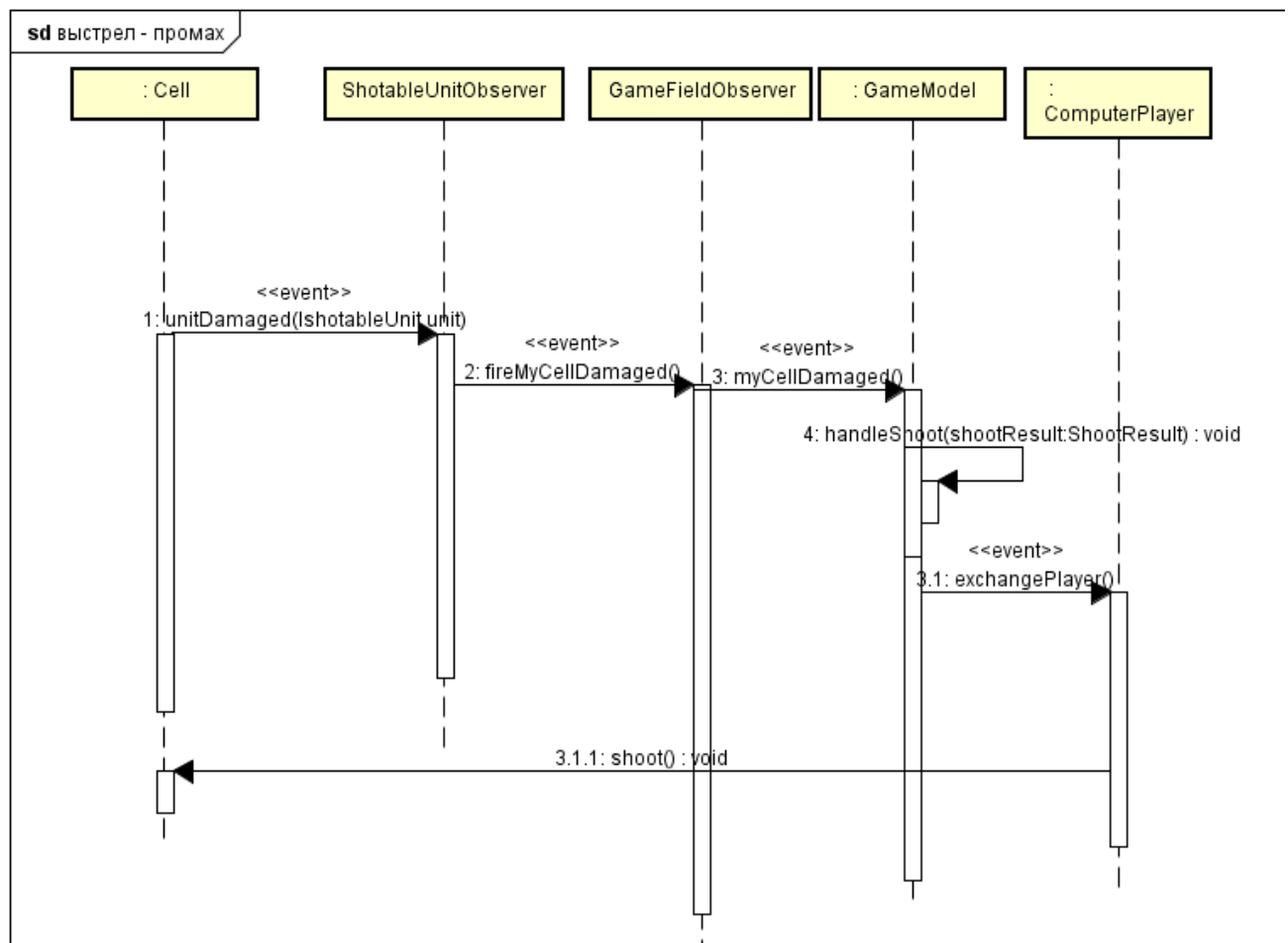
### 3.5 Типовые процессы в программе



Общий игровой цикл



Выстрел - попадание



Выстрел - промах

### 3.6 Человеко-машинное взаимодействие

При запуске игры пользователь видит экран настроек (рис. 1), на котором предлагается выбрать размер игрового поля и тип игры.

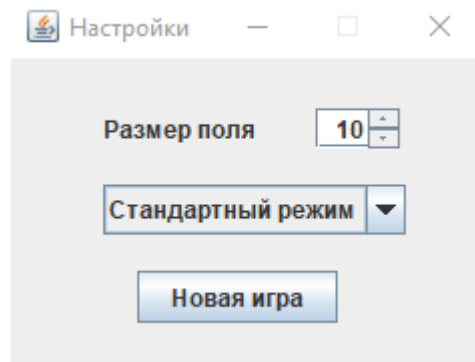


Рисунок 1 – Начальный экран программы

Общий вид экрана игры представлен на рисунке 2. На нем располагаются поля с пользователем и компьютера. Поле с кораблями пользователя открыто, на нем цветом отмечены корабли (черный) и море (голубой). Синяя обводка вокруг поля показывает, что сейчас нужно стрелять по нему.

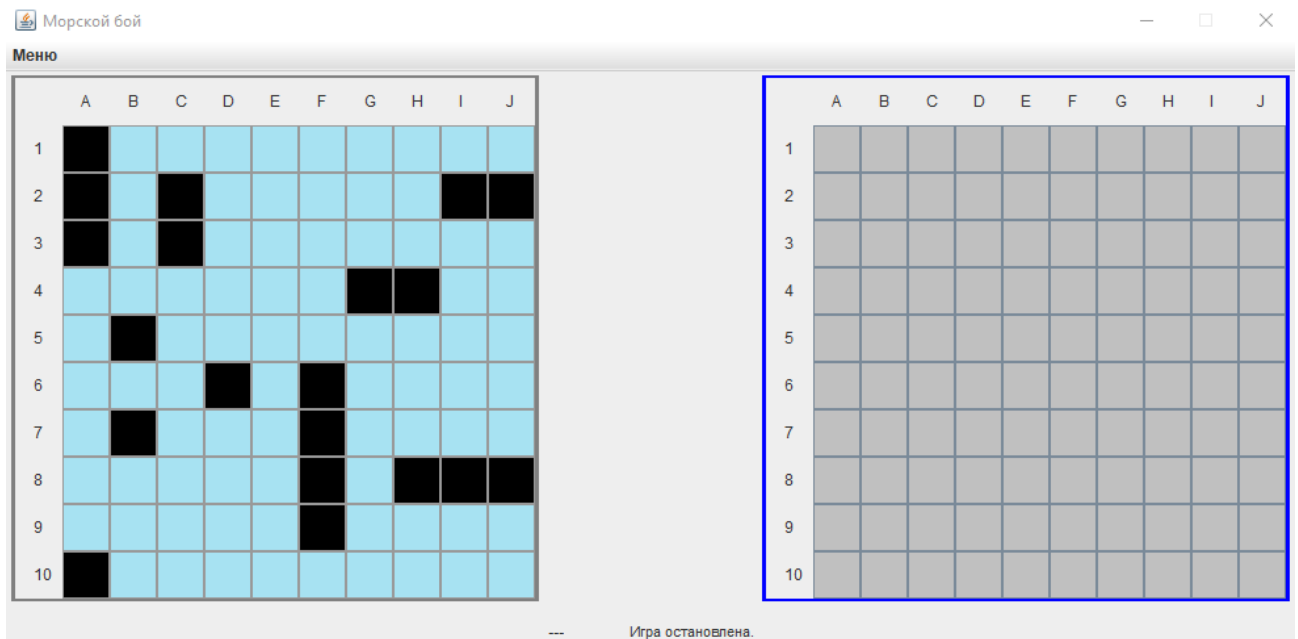


Рисунок 2 – Экран игры

Для выстрела пользователь кликает по ячейке, в которую он хочет выстрелить. Ячейка отмечается пораженной: если попали в море, то внутри клетки рисуется синий круг, если попали по палубе, то внутри клетки рисуется красный крест (рис. 3-4).



Рисунок 3 – Клетка корабля

а) не поражена

б) поражена

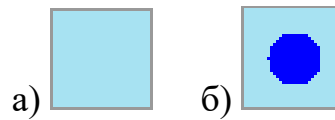


Рисунок 4 – Клетка моря

а) не поражена

б) поражена

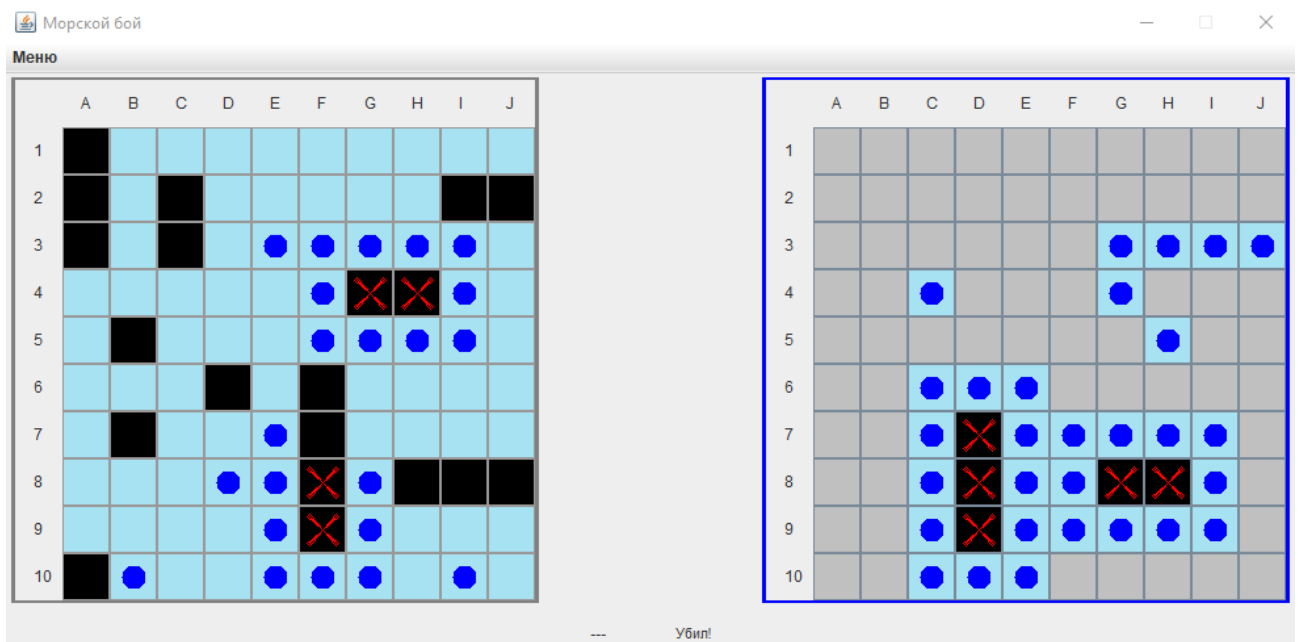


Рис. 5 – Общий вид игры с отметками выстрелов

Пользователь не может совершить выстрел по полю со своими кораблями.

Выстрел компьютера происходит автоматически, результат выстрела отображается на соответствующем поле.

При завершении игры выводится окно с сообщением о результате игры, представленное на рисунке 6. При этом оба поля открываются.



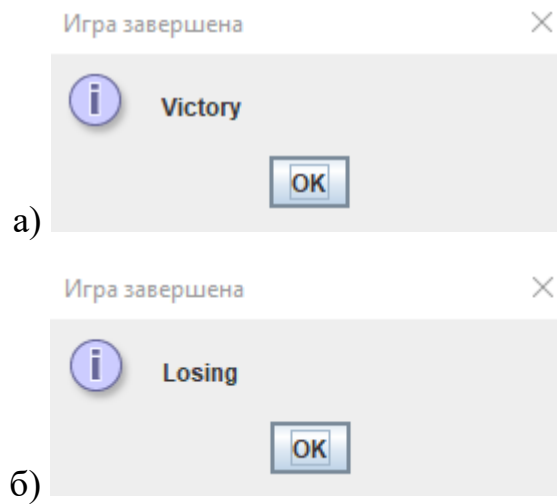


Рисунок 6 – Окно с сообщением о результате игры

а) победа человека

б) победа компьютера

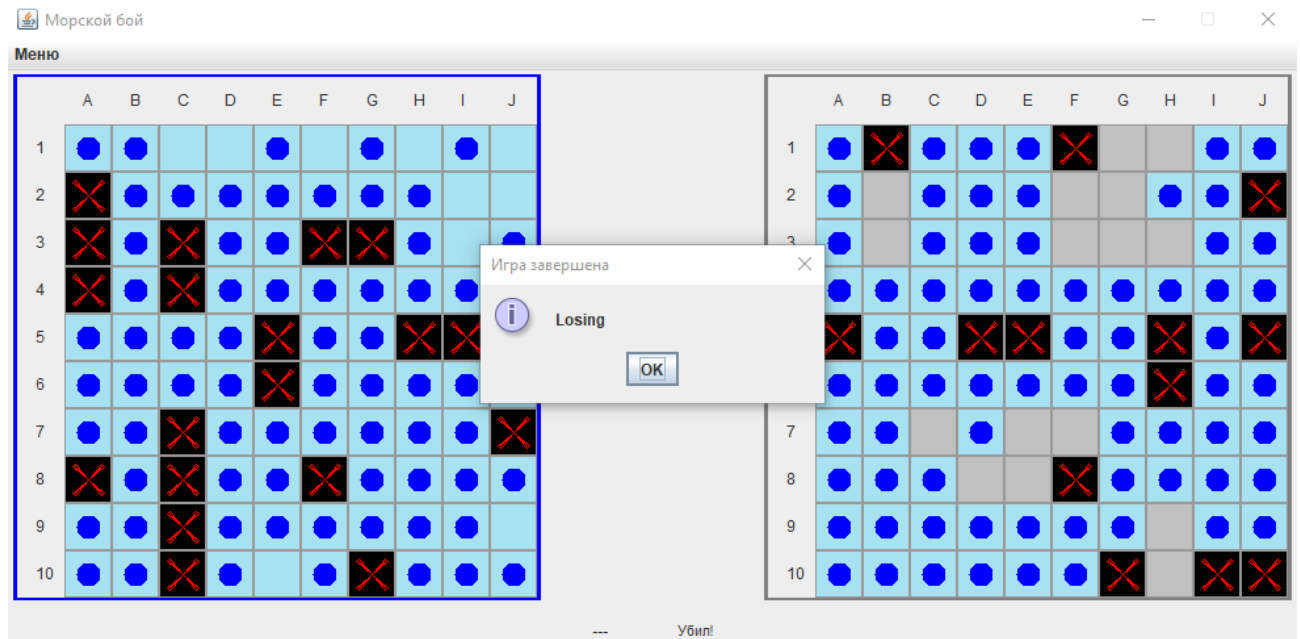


Рисунок 7 – Общий вид игры при завершении

После закрытия окна с сообщением о победе можно начать новую игру через меню «Игра» или выйти из программы.

### 3.7 Реализация ключевых классов

```
public abstract class Ship implements Cloneable {

    public enum ShipState {
        Ok,
        Drowned
    }

    protected Set<Deck> decks;

    protected GameField field;

    protected ShipState state;

    protected ShipObserver shipObserver;

    public Ship() {
        this.field = null;
        this.decks = new HashSet<>();
        this.state = ShipState.Ok;
        this.shipObserver = new ShipObserver();
    }

    /**
     * Существенные для расстановки направления (у 1-палубного всего 1
     * направление, у других могут быть 2 или 4 направления)
     */
    abstract public Direction[] meaningfulDirections();

    public int getDecksCount() {
        return decks.size();
    }

    public Set<Deck> getDecks() {
        return decks;
    }

    public boolean isActive() {
        for (Deck deck : decks) {
            if (deck.getState() == ShutableUnitState.Ok) {
                return true;
            }
        }
        return false;
    }

    public boolean isMyDeck(Deck deck) {
        return decks.contains(deck);
    }

    public void setup (GameField field) {
        if (this.field != null) {
            throw new RuntimeException("Cannot change field for ship !");
        } else {
            this.field = field;
        }
    }

    @Override
```

```

public abstract Ship clone();

/**
 * Инициализировать палубы
 */
abstract protected void init();

// ----- Порождает события -----
ArrayList<IShipListener> shipListeners = new ArrayList();

public void addShipListener(IShipListener l) {
    shipListeners.add(l);
}

public void fireShipDamaged() {
    for (IShipListener shipListener : shipListeners) {
        shipListener.shipDamaged();
    }
}

public void fireShipDrowned(IShutableUnit unit) {
    for (IShipListener shipListener : shipListeners) {
        shipListener.shipDrowned(unit);
    }
}

private class ShipObserver implements IShutableUnitListener {

    @Override
    public void unitDamaged(IShutableUnit unit) {
        //проверить, не затонул ли корабль
        int brokenDecks = 0;
        for (Deck obj : decks) {
            if (obj.getState() == ShutableUnitState.Broken) {
                brokenDecks++;
            }
        }
        if (brokenDecks == getDecksCount()) {
            state = ShipState.Drowned;
            //убил
            fireShipDrowned(unit);
        } else {
            //попал
            fireShipDamaged();
        }
    }
}

}

public class StandardShip extends Ship {

    public StandardShip(){
        super();
    }
    private StandardShip(StandardShip other){
        this();
    }
    protected int length;

    public StandardShip(int length) {
        super();
    }

```

```

        if(length < 1) {
            throw new RuntimeException("Invalid length for linear ship is specified: "+length);
        }
        this.length = length;

        init();
    }

    @Override
    protected void init() {

        for( int i = 0 ; i < this.length ; i++ ) {
            Deck deck = new StandardDeck(this, new Location(i,0));
            //теперь корабль следит за этой палубой//
            deck.addModelUnitListener(shipObserver);
            this.decks.add( deck);

        }
    }

    // для стандартных кораблей только 2 направления
    @Override
    public Direction[] meaningfulDirections() {
        return new Direction [] {Direction.NORTH, Direction.EAST} ;
    }

    @Override
    public Ship clone() {
        return new StandardShip(this);
    }
}

public abstract class Deck implements IShorableUnit{

    protected Cell cell;

    protected Location shipBasedLocation;

    protected ShorableUnitState state;

    public Deck(Ship ship, Location shipBasedLocation){
        cell = null;
        this.shipBasedLocation = shipBasedLocation;
        this.state = ShorableUnitState.Ok;
    }

    /**
     * @return the cell
     */
    public Cell getCell() {
        return cell;
    }

    @Override
    public ShorableUnitState getState() {
        return this.state;
    }

    public Location getShipBasedLocation(){
        return shipBasedLocation;
    }
}

```

```

/**
 * @param cell the cell to set
 */
public void setCell(Cell cell) {
    if(cell != null) {
        this.cell = cell;
        cell.setDeck(this);
    }
}

public void unsetCell() {
    if(this.cell != null) {
        Cell _cell = this.cell;
        this.cell = null;
        _cell.unsetDeck();
    }
}

/** Ударить в палубу
 */
@Override
public void shootAction() {
    if (this.state != ShorableUnitState.Broken) {
        this.state = ShorableUnitState.Broken;
        // в меня попали
        fireShorableUnitDamaged();
    }
}

ArrayList<IShorableUnitListener> modelListeners = new ArrayList<>();
ArrayList<IShorableUnitListener> buttonListeners = new ArrayList<>();

@Override
public void addModelUnitListener(IShorableUnitListener l) {
    modelListeners.add(l);
}

@Override
public void addButtonUnitListener(IShorableUnitListener l) {
    buttonListeners.add(l);
}

@Override
public void fireShorableUnitDamaged() {
    for (IShorableUnitListener listener : buttonListeners) {
        listener.unitDamaged(this);
    }
    for (IShorableUnitListener listener : modelListeners){
        listener.unitDamaged(this);
    }
}
}

public class SeveralShotResistantDeck extends Deck {

    protected int shotsToCrash;

    public SeveralShotResistantDeck(Ship ship, Location shipBasedLocation) {
        super(ship, shipBasedLocation);
    }

    public SeveralShotResistantDeck(Ship ship, Location shipBasedLocation, int shotsToCrash) {

```

```

        super(ship, shipBasedLocation);
        this.shotsToCrash = shotsToCrash;
    }

    /**
     * Выстрелить в палубу.
     */
    @Override
    public void shootAction() {
        if (this.state != ShorableUnitState.Broken) {
            this.shotsToCrash--;
            if (this.shotsToCrash == 0) {
                this.state = ShorableUnitState.Broken;
                // в меня попали
                fireShorableUnitDamaged();
            } else {
                this.state = ShorableUnitState.Damaged;
                // в меня попали
                fireShorableUnitDamaged();
            }
        }
    }
}

public class StandardDeck extends SeveralShotResistantDeck {

    public StandardDeck(Ship ship, Location shipBasedLocation) {
        super(ship, shipBasedLocation);
        this.shotsToCrash = 1;
    }
}

public abstract class Player {

    protected GameField enemyField;

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

    public GameField getEnemyField(){
        return this.enemyField;
    }
}

public class HumanPlayer extends Player{

    public HumanPlayer() {
        super();
    }
}

public class ComputerPlayer extends Player {

    private Set<Cell> allCells;
    private Set<Cell> cellsToShot;
    private Cell lastHurtedDeck;

    private GameModel game;
    private GameModelObserver observer;
    private UnitsObserver unitObserver;

```

```

public ComputerPlayer() {
    cellsToShot = new HashSet<>();
    allCells = new HashSet<>();
    game = null;
    observer = new GameModelObserver();
    unitObserver = new UnitsObserver();
    lastHurtedDeck = null;
}

public ComputerPlayer(GameModel game) {
    this();
    this.game = game;
    this.game.addGameListener(observer);
    this.observer.setComputerPlayer(this);
}

@Override
public void setField(GameField enemyField) {
    super.setField(enemyField);
    allCells.addAll(this.enemyField.getCells().values());

    for (Cell c : allCells) {
        c.addButtonUnitListener(unitObserver);
    }
}

public void shoot() {
    Cell nextCell;
    do {
        allCells.removeIf((c) -> c.getState() == IShorableUnit.SorableUnitState.Broken);
        cellsToShot.retainAll(allCells);
        //выбрать клетку для выстрела
        nextCell = chooseCell(enemyField);
        System.out.println("nextcell= " + nextCell.getLocation().x() + " " +
nextCell.getLocation().y() + "cellsToShot.size= " + cellsToShot.size());
        if (nextCell.getDeck() != null) {
            //запомнить клетку с палубой, в которую будем стрелять как последнюю подбитую
            lastHurtedDeck = nextCell;
            //сформировать множество клеток соседей, возможно содержащих еще палубу
            cellsToShot = enemyField.get8NeighboursFor(lastHurtedDeck.getLocation());
            //сама палуба тоже может быть не добита
            cellsToShot.add(lastHurtedDeck);
            allCells.removeIf(c -> (c.getDeck() !=null && c.getDeck().getState() ==
ISorableUnit.SorableUnitState.Broken));
            //перед выстрелом исключить уже убитые клетки
            cellsToShot.retainAll(allCells);
            if (cellsToShot.contains(nextCell)){
                nextCell.getDeck().shootAction();
            }
            allCells.removeIf(c -> (c.getDeck() !=null && c.getDeck().getState() ==
ISorableUnit.SorableUnitState.Broken));
        } else {
            if (allCells.contains(nextCell)){
                nextCell.shootAction();
            }
        }
    } while (nextCell.getDeck() != null);
}

public Cell chooseCell(GameField field) {
    Cell nextCell;

```

```

        Cell arr[] = {};
        if (lastHurledDeck == null || cellsToShot.isEmpty()) {
            //рандом
            nextCell = Utils.chooseOne(allCells.toArray(arr));
        } else {
            nextCell = Utils.chooseOne(cellsToShot.toArray(arr));
        }
        return nextCell;
    }

    private class GameModelObserver implements IGameListener {

        ComputerPlayer player;

        public void setComputerPlayer(ComputerPlayer player) {
            this.player = player;
        }

        @Override
        public void shootPerformed(ShootResult shootResult) {

        }

        @Override
        public void gameFinished(GameModel.GameResult result) {

        }

        @Override
        public void exchangePlayer(Player activePlayer) {
            if (activePlayer instanceof ComputerPlayer && activePlayer == player) {
                shoot();
            }
        }
    }

    private class UnitsObserver implements IShorableUnitListener {

        @Override
        public void unitDamaged(IShorableUnit unit) {
            if (unit instanceof Deck && unit.getState() ==
IShorableUnit.ShorableUnitState.Broken) {
                allCells.remove(((Deck) unit).getCell());
            }
        }
    }

    private class GameFieldObserver implements IGameFieldListener {

        @Override
        public void myCellDamaged() {

            throw new UnsupportedOperationException("Not supported yet."); //To change body of
generated methods, choose Tools | Templates.
        }

        @Override
        public void myShipDamaged() {
            throw new UnsupportedOperationException("Not supported yet."); //To change body of

```



```

generated methods, choose Tools | Templates.
    }

    @Override
    public void myShipDrowned() {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of
generated methods, choose Tools | Templates.
    }

}

}

public class GameModel {

    private boolean running;

    private GameField fields[] = {null, null};

    private Player players[] = {null, null}; //0 - bot, 1 - игрок

    private Player activePlayer;

    private List<List<Ship>> ships;

    private GameFieldObserver fieldObserver;

    private ShipPoolFactory shipPoolFactory;

    public enum GameResult {
        Victory, Losing, Ok
    }

    public GameModel() {
        this.running = false;
        this.fieldObserver = new GameFieldObserver();
        this.shipPoolFactory = new ShipPoolFactory();
        this.ships = new ArrayList<List<Ship>>();
    }

    public GameField[] getFields() {
        return fields;
    }

    public Player getActivePlayer() {
        return activePlayer;
    }

    public boolean startNewGame(boolean force, int gameMode, int fieldsSize) {

        if (running && !force) {
            return false;
        }
        if (running && force) {
            stopGame(true);
        }
        initBattleFields(fieldsSize);

        initPlayers();

        if (force) {
            ships.clear();
        }
    }
}

```

```

        if (gameMode == 0) {
            for (int i = 0; i < 2; i++) {
                ships.add(shipPoolFactory.createStandardShipPool());
            }
        } else {
            for (int i = 0; i < 2; i++) {
                ships.add(shipPoolFactory.createNonStandardShipPool());
            }
        }

        //очистить поля от кораблей
        for (int i = 0; i < 2; i++) {
            fields[i].clear();
        }
        //расставить корабли на поле
        for (int i = 0; i < 2; i++) {
            if (!fields[i].setShips(ships.get(i))) {
                //не получилось разместить корабли, надо увеличить размер поля!
                int a = 0;
            }
        }

        players[0].setField(fields[1]);
        players[1].setField(fields[0]);

        running = true;
        return true;
    }

    private void initBattleFields(int size) {
        for (int i = 0; i < 2; i++) {
            GameField newField = new GameField(size);
            newField.addFieldListener(fieldObserver);
            fields[i] = newField;
        }
    }

    private void initPlayers() {
        Player newPlayer = new ComputerPlayer(this);
        players[0] = newPlayer;

        newPlayer = new HumanPlayer();
        players[1] = newPlayer;
    }

    private void setActivePlayer(Player activePlayer) {
        this.activePlayer = activePlayer;
        exchangePlayer(activePlayer);
    }

    public void runGame() {
        // to highlight views
        setActivePlayer(players[1]);
        exchangePlayer(players[1]);
    }

    private void identifyGameOver() {

```

```

        if (fields[0].hasActiveShips()) {
            //если у робота не осталось активных кораблей
            fireGameFinished(GameResult.Victory);
        } else if (fields[1].hasActiveShips()) {
            ///если у человека не осталось активных кораблей
            fireGameFinished(GameResult.Losing);
        }
    }

private void handleShoot(ShootResult shootResult) {
    if (!running) {
        return;
    }
    // оповестить слушателей о событии
    fireShootPerfomed(shootResult);

    switch (shootResult) {
        case SHIP_DAMAGED:
            //ни на что не влияет
            break;
        case MISS:
            this.activePlayer = getInactivePlayer(activePlayer);
            // переход хода
            exchangePlayer(activePlayer);
            break;

        case SHIP_DROWNED:
            openCellsNearDrownedShip();
            identifyGameOver();
            break;
    }
}

private Player getInactivePlayer(Player activePlayer) {
    if (players[0] == activePlayer) {
        return players[1];
    } else {
        return players[0];
    }
}

private void openCellsNearDrownedShip() {

    GameField field = activePlayer.getEnemyField();

    Cell rememberedLastHitCell = field.getLastHitCell();

    Set<Cell> shipCells = new HashSet<>();
    shipCells.add(rememberedLastHitCell);
    Set<Cell> shipNeighbourCells = new HashSet<>();
    Set<Cell> neighbours;

    neighbours = field.get4NeighboursFor(rememberedLastHitCell.getLocation());

    while (!neighbours.isEmpty()) {
        neighbours.removeIf(c -> c.getDeck() == null);
        neighbours.removeAll(shipCells);

        shipCells.addAll(neighbours);

        shipNeighbourCells.removeAll(shipCells);
    }
}

```

```

        shipNeighbourCells.addAll(neighbours); // новые в корабле

        neighbours.clear();
        for (Cell c : shipNeighbourCells) {
            neighbours.addAll(field.get4NeighboursFor(c.getLocation()));
        }
    }

    shipNeighbourCells.clear();

    // все клетки корабля найдены
    for (Cell c : shipCells) {
        shipNeighbourCells.addAll(field.get8NeighboursFor(c.getLocation()));
    }

    shipNeighbourCells.retainAll(field.getAllLockedCells());

    for (Cell c : shipNeighbourCells) {
        c.openAction();
    }

    // возвращаем выделение
    rememberedLastHitCell.openAction();
}

public boolean stopGame(boolean force) {
    if (running && !force) {
        return false;
    }
    running = false;

    return true;
}

private class GameFieldObserver implements IGameFieldListener {

    @Override
    public void myCellDamaged() {
        handleShoot(ShootResult.MISS);
    }

    @Override
    public void myShipDamaged() {
        handleShoot(ShootResult.SHIP_DAMAGED);
    }

    @Override
    public void myShipDrowned() {
        handleShoot(ShootResult.SHIP_DROWNED);
    }

}

ArrayList<IGameListener> gameListeners = new ArrayList();

public void addGameListener(IGameListener l) {
    gameListeners.add(l);
}

public void deleteGameListener(IGameListener l) {
    gameListeners.remove(l);
}

```

```

    public void fireShootPerfomed(ShootResult shootResult) {
        for (IGameListener gameListener : gameListeners) {
            gameListener.shootPerformed(shootResult);
        }
    }

    public void fireGameFinished(GameResult result) {
        for (IGameListener gameListener : gameListeners) {
            gameListener.gameFinished(result);
        }
    }

    public void exchangePlayer(Player activePlayer) {
        for (IGameListener gameListener : gameListeners) {
            gameListener.exchangePlayer(activePlayer);
        }
    }
}

public class GameField {

    private List<Ship> ships = new ArrayList<>();
    private int size;
    private Map<Location, Cell> cells;
    private ShutableUnitObserver shutableUnitObserver;
    private ShipObserver shipObserver;
    private Cell lastHitCell;

    public GameField() {
        this.cells = new HashMap<>();
        this.shutableUnitObserver = new ShutableUnitObserver();
        this.size = 0;
        this.lastHitCell = null;
        this.shipObserver = new ShipObserver();
    }

    public GameField(int fieldSize) {
        this();
        init(fieldSize);
    }

    public boolean hasActiveShips() {
        for (Ship ship : ships) {
            if (ship.isActive()) {
                return false;
            }
        }

        return true;
    }

    private Cell[] emptyCells() {
        // найти незанятые ячейки...
        // все
        Set<Cell> emptyCells = new HashSet<>(cells.values());
        // вычесть занятые
        emptyCells.removeIf((c) -> c.getDeck() != null);

        Cell[] arr = new Cell[emptyCells.size()];
        return emptyCells.toArray(arr);
    }
}

```

```

}

private Set<Cell> filterCells(Predicate<Cell> p) {
    // найти незанятые ячейки

    return new HashSet<>(cells.values());
}

public Set<Cell> getAllLockedCells() {
    return filterCells(c -> c.getState() == IShorableUnit.ShorableUnitState.Broken);
}

public boolean setShips(List<Ship> shipsPool) {
    // рандомным образом расставить полученные корабли
    // очистить все клетки от палуб кораблей
    for (Cell cell : cells.values()) {
        cell.unsetDeck();
    }
    // составить очередь из кораблей
    Deque<Ship> shipsQueue = new ArrayDeque<>();

    for (Ship ship : shipsPool) {
        shipsQueue.addFirst(ship);
    }
    // расставленные корабли
    List<Ship> settedShips = new ArrayList<>();

    int totalLimit = shipsPool.size() * 3;
    int shipLimit = 4 * 10;

    int totalIterations = 0;
    // расставлять флот, пока не получится или не выйдет лимит
    while (totalIterations < totalLimit) {
        totalIterations++;

        Ship currentShip = shipsQueue.pollFirst();
        if (currentShip == null) {
            // all ships are already disposed
            break;
        }

        boolean shipOk = false;

        int shipIterations = 0;
        // расставлять корабль, пока не получится или не выйдет лимит
        while (shipIterations < shipLimit) {
            shipIterations++;

            Cell freeCell = Utils.chooseOne(emptyCells());

            if (freeCell == null) {
                shipOk = false;
                break;
            }

            for (Direction dir : currentShip.meaningfulDirections()) {
                // choose another
                dir = Utils.chooseOne(currentShip.meaningfulDirections());
                ShipLocation shipLoc = new ShipLocation(freeCell.getLocation(), dir);

                // разместить корабль
                shipOk = setShip(currentShip, shipLoc);
            }
        }
    }
}

```

```

        if (shipOk) {
            break;
        }
    }
    if (shipOk) {
        // add to disposed
        settedShips.add(currentShip);
        break;
    }
}

// if error
if (!shipOk) {
    totalIterations--;

    if (settedShips.isEmpty()) {
        System.out.println("Cannot dispose ship: nothing to unset");
        return false;
    }

    // unset a random ship
    // любого по индексу
    int index = (int) (Math.random() * settedShips.size());

    Ship shipToUnset = settedShips.get(index);
    settedShips.remove(index);

    unsetShip(shipToUnset);

    // return ships to queue
    shipsQueue.addLast(shipToUnset);
    shipsQueue.addLast(currentShip);
}

}

ships = settedShips;
//подключить наблюдателей
for (Ship ship : ships) {
    ship.addShipListener(shipObserver);
}

if (!shipsQueue.isEmpty()) {
    System.out.println("dispose ships failed, ships remaining: " + shipsQueue.size());
}
return totalIterations < totalLimit;
}

private boolean setShip(Ship ship, ShipLocation shipLocation) {
    boolean isOk = true;

    // по всем палубам корабля
    for (Deck deck : ship.getDecks()) {
        // абсолютная позиция палубы
        Location loc = deck.getShipBasedLocation()
            .movedBy(shipLocation.location().x(), shipLocation.location().y())
            .rotatedBy(shipLocation.location(), shipLocation.direction());

        if (!this.isLocationValid(loc)) {
            isOk = false;
            break;
        }
    }
}

```

```

        Cell cell = cells.get(loc);

        // пересекается по соседям с чужими палубами
        Set<Cell> influencedCells = this.get8NeighboursFor(loc);
        influencedCells.add(cell);

        for (Cell c : influencedCells) {
            if (c.getDeck() != null && !ship.isMyDeck(c.getDeck())) {
                isOk = false;
                break;
            }
        }

        if (!isOk) {
            break;
        }

        deck.setCell(cell);
    }

    if (!isOk) {
        unsetShip(ship);
    }

    return isOk;
}

public Set<Cell> get8NeighboursFor(Location location) {
    Set<Cell> set = new HashSet<>();

    for (int j = -1; j <= 1; j += 1) {
        for (int i = -1; i <= 1; i += 1) {
            if (i == 0 && j == 0) {
                continue;
            }

            Location loc = location.movedBy(i, j);

            if (this.isLocationValid(loc)) {
                set.add(cells.get(loc));
            }
        }
    }

    return set;
}

public Set<Cell> get4NeighboursFor(Location location) {
    Set<Cell> set = new HashSet<>();

    Location loc;
    loc = location.movedBy(-1, 0);
    if (this.isLocationValid(loc)) {
        set.add(cells.get(loc));
    }
    loc = location.movedBy(0, -1);
    if (this.isLocationValid(loc)) {
        set.add(cells.get(loc));
    }
    loc = location.movedBy(0, 1);
    if (this.isLocationValid(loc)) {

```



```

        set.add(cells.get(loc));
    }
    loc = location.movedBy(1, 0);
    if (this.isLocationValid(loc)) {
        set.add(cells.get(loc));
    }

    return set;
}

private void unsetShip(Ship ship) {
    // по всем палубам корабля
    for (Deck deck : ship.getDecks()) {
        deck.unsetCell();
    }
}

public List<Ship> getShips() {
    //получить список кораблей на поле
    return ships;
}

public Map<Location, Cell> getCells() {
    return cells;
}

public int size() {
    return size;
}

public Cell cellAt(Location location) {
    return cells.get(location);
}

private void init(int fieldSize) {

    this.size = fieldSize;

    cells.clear();

    // fill with cells
    for (int j = 0; j < size; ++j) {
        for (int i = 0; i < size; ++i) {
            Location loc = new Location(i, j);
            Cell newCell = new Cell(loc);
            //теперь поле слушает эту ячейку

            newCell.addModelUnitListener(shotableUnitObserver);
            cells.put(loc, newCell);
        }
    }

}

public Cell getLastHitCell() {
    return lastHitCell;
}

public boolean isLocationValid(Location location) {
    return location.x() >= 0
        && location.y() >= 0
        && location.x() < size

```

```

        && location.y() < size;
    }
    // ----- Порождает события -----

    ArrayList<IGameFieldListener> listeners = new ArrayList();

    public void addFieldListener(IGameFieldListener l) {
        listeners.add(l);
    }

    public void fireMyCellDamaged() {
        for (IGameFieldListener listener : listeners) {
            listener.myCellDamaged();
        }
    }

    public void fireMyShipDamaged() {
        for (IGameFieldListener listener : listeners) {
            listener.myShipDamaged();
        }
    }

    public void fireMyShipDrowned() {
        for (IGameFieldListener listener : listeners) {
            listener.myShipDrowned();
        }
    }

    void clear() {

    }

    private class ShorableUnitObserver implements ISorableUnitListener {

        @Override
        public void unitDamaged(ISorableUnit unit) {
            lastHitCell = (Cell) unit;
            fireMyCellDamaged();
        }
    }

    private class ShipObserver implements IShipListener {

        @Override
        public void shipDrowned(ISorableUnit lastHittedDeck) {
            lastHitCell = ((Deck) lastHittedDeck).getCell();
            fireMyShipDrowned();
        }

        @Override
        public void shipDamaged() {
            fireMyShipDamaged();
        }
    }
}

public class Location {

    private int x,y;

    /**

```

```

    * @param x col (column)
    * @param y row
    */
    public Location(int x, int y) {
        this.x = x;
        this.y = y;
    }

    // пары синонимов
    public int x() {
        return x;
    }
    public int col() {
        return x;
    }

    public int y() {
        return y;
    }
    public int row() {
        return y;
    }

    // ===== Преобразования =====

    /** Сдвинутая позиция
     * @return new Location
     */
    public Location movedBy(int shift_x, int shift_y) {
        return new Location(x() + shift_x, y() + shift_y);
    }

    /** Позиция, повернутая относительно базовой позиции в заданном направлении. Базовым
    направлением считается Direction.north().
     * @return new Location
     */
    public Location rotatedBy(Location basePoint, Direction direction) {
        int diff_x = x() - basePoint.x();
        int diff_y = y() - basePoint.y();

        switch(direction) {
            case NORTH:
                return this; //.clone();

            case EAST:
                return new Location(basePoint.x() - diff_y, basePoint.y() + diff_x);

            case SOUTH:
                return new Location(basePoint.x() - diff_x, basePoint.y() - diff_y);

            case WEST:
                return new Location(basePoint.x() + diff_y, basePoint.y() - diff_x);

            default:
                return null;
        }
    }

    // ===== Общие методы класса Object =====

    @Override
    public Location clone() throws CloneNotSupportedException{
        Location copy = (Location)super.clone();
    }

```

```

        return copy;
    }

    @Override
    public String toString() {
        return ((char)('A'+x())) + "" + (y()+1);
    }

    @Override
    public boolean equals(Object o) {
        if(o instanceof Location) {
            return ((Location)o).x() == x()
                && ((Location)o).y() == y();
        }
        return false;
    }

    @Override
    public int hashCode() {
        return 1024 * this.x + this.y;
    }
}

public enum Direction {

    NORTH(0),
    SOUTH(6),
    EAST (3),
    WEST (9);

    // определяем направление в часах (0 до 12)
    private int _hours;

    Direction(int hours) {
        // Приводим заданные часы к допустимому диапазону
        this._hours = suiteHours(hours);
    }

    /** Приводим заданные часы к допустимому диапазону */
    private static int suiteHours(int hours) {
        // Приводим заданные часы к допустимому диапазону
        hours = hours%12;
        if(hours < 0)
            hours += 12;
        return hours;
    }

    public int getClockHours() {
        return _hours;
    }

    public int getDegrees() {
        return _hours * 30;
    }

    // ----- Возможные направления -----

    private static HashMap<Integer, Direction> directions = new HashMap <>();
    private static void addDirectionFor(Direction d) {
        int hours = d.getClockHours();
        if( ! directions.containsKey(hours) ) {
            directions.put( hours, d );
        }
    }

```

```

    }
}
private static Direction getDirectionFor(int hours) {
    hours = suiteHours(hours);
    return directions.getOrDefault(hours, NORTH);
}
static {
    // Add all enum constants that exist
    for(Direction d : new Direction[] {NORTH,SOUTH,EAST,WEST} ) {
        addDirectionFor(d);
    }
}

// Для обратной совместимости
public static Direction north()
{ return NORTH; }

public static Direction south()
{ return SOUTH; }

public static Direction east()
{ return EAST; }

public static Direction west()
{ return WEST; }

// ----- Новые направления -----

public Direction clockwise() {
    return getDirectionFor(this._hours + 3);
}

public Direction anticlockwise() {
    return getDirectionFor(this._hours - 3);
}

public Direction opposite() {
    return getDirectionFor(this._hours + 6);
}

public Direction rightword() {
    return clockwise();
}

public Direction leftword() {
    return anticlockwise();
}

// ----- Сравнить направления -----

public boolean isOppositeTo(Direction other) {
    return this.opposite().equals(other);
}
}

public class Cell implements IShorableUnit {

    private Location location;
    private Deck deck;
    private ShorableUnitState state;

```

```

public Cell(Location location) {;

    this.location = location;
    this.state = ShorableUnitState.Ok;
}

@Override
public void shootAction() {
    if (this.deck == null && this.state != ShorableUnitState.Broken) {
        this.state = ShorableUnitState.Broken;
        // в меня попали
        fireShorableUnitDamaged();
    }
}

void openAction() {
    if (this.deck == null) {
        this.state = ShorableUnitState.Broken;
        // меня открыли
        fireCellOpened();
    }
}

public Location getLocation() {
    return location;
}

@Override
public ShorableUnitState getState() {
    return this.state;
}

/**
 * @return the deck
 */
public Deck getDeck() {
    return deck;
}

/**
 * @param deck the deck to set
 */
public void setDeck(Deck deck) {
    this.deck = deck;
    if (this.deck != null && this.deck.getCell() != this) {
        this.deck.setCell(this);
    }
}

public void unsetDeck() {
    if (this.deck != null && this.deck.getCell() != null) {
        this.deck.unsetCell();
    }
    this.deck = null;
}

ArrayList<IShorableUnitListener> modelListeners = new ArrayList<IShorableUnitListener>();

ArrayList<IShorableUnitListener> buttonListeners = new ArrayList<IShorableUnitListener>();

@Override

```

```

    public void addModelUnitListener(IShutableUnitListener l) {
        modellListeners.add(l);
    }

    @Override
    public void addButtonUnitListener(IShutableUnitListener l) {
        buttonListeners.add(l);
    }

    @Override
    public void fireShutableUnitDamaged() {
        for (IShutableUnitListener listener : buttonListeners) {
            listener.unitDamaged(this);
        }
        for (IShutableUnitListener listener : modellListeners) {
            listener.unitDamaged(this);
        }
    }

    public void fireCellOpened() {
        for (IShutableUnitListener listener : buttonListeners) {
            listener.unitDamaged(this);
        }
    }
}

public class GamePanel extends JFrame {

    private GameModel game;
    private JLabel welcomeLabel;
    private JLabel statusBarLabel;
    private JLabel statusBarLeftLabel;
    private Box mainBoxWithStatusBar;
    private final int gameMode;
    private final int fieldSize;
    boolean firstTimeMakeGameWindow;

    private final GameObserver observer;

    private JMenuBar menu = null;
    private static final String fileItems[] = new String[]{"Новая игра", "Выход"};

    private final GameFieldPanel fields[] = {null, null};

    public GameModel getGame() {
        return this.game;
    }

    public GamePanel(int gameMode, int fieldSize) {
        super();

        //create a Game
        this.game = new GameModel();
        this.observer = new GameObserver();
        this.game.addGameListener(observer);
        this.fieldSize = fieldSize;
        this.gameMode = gameMode;

        for (int i = 0; i < 2; i++) {
            fields[i] = new GameFieldPanel();
        }
    }
}

```

```

        // Первоначальная настройка окна
        setupWindow();

        this.setMinimumSize(new Dimension(1000, 500));

        startNewGame(true);

        setLocationRelativeTo(null);
    }

    /**
     * Первоначальная настройка окна
     */
    private void setupWindow() {

        this.firstTimeMakeGameWindow = true;

        this.setTitle("Морской бой");

        this.welcomeLabel = new JLabel();
        this.welcomeLabel.setFont(new Font("Verdana", Font.BOLD, 28));
        this.welcomeLabel.setToolTipText("(Меню -> Новая игра)");

        this.statusBarLabel = new JLabel();
        this.statusBarLabel.setFont(new Font("Courier", Font.PLAIN, 11));
        this.statusBarLabel.setText("Запустить игру : Меню -> Новая игра");

        this.statusBarLeftLabel = new JLabel("---");

        this.mainBoxWithStatusBar = Box.createVerticalBox();
        this.mainBoxWithStatusBar.add(Box.createVerticalStrut(80));
        this.mainBoxWithStatusBar.add(welcomeLabel);
        this.mainBoxWithStatusBar.add(Box.createVerticalStrut(110));
        this.mainBoxWithStatusBar.add(statusBarLabel);

        setContentPane(mainBoxWithStatusBar);

        // Меню
        createMenu();
        setJMenuBar(menu);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        pack();

        setResizable(false);
    }

    /**
     * Начать новую игру
     *
     * @param force прервать текущую игру, если true
     * @return false, если игра ещё идёт и force == false
     */
    private boolean startNewGame(boolean force) {

        // остановить игру, если запущена
        if (!stopCurrentGame(force)) {
            return false;
        }
    }

```



```

// тут создаем и инициализируем поля, клетки
game.startNewGame(true, gameMode, fieldSize);

// turn repaint off
boolean ignoreRepaint = true;
this.setIgnoreRepaint(ignoreRepaint);
for (int i = 0; i < 2; i++) {
    fields[i].setIgnoreRepaint(ignoreRepaint);
}

this.pack();

fields[0].init(game.getFields()[0], UnitButtonState.LOCKED);
fields[1].init(game.getFields()[1], UnitButtonState.OPENED);

// инициализация окна
initWindowForGame();

// turn repaint on
ignoreRepaint = false;
this.setIgnoreRepaint(ignoreRepaint);
for (int i = 0; i < 2; i++) {
    fields[i].setIgnoreRepaint(ignoreRepaint);
}

this.setTitle("Морской бой");

// старт игры (первый ход)
game.runGame();

return true;
}

private boolean stopCurrentGame(boolean force) {
    if (!game.stopGame(force)) {
        return false;
    }

    // заблокировать и открыть поля
    for (int i = 0; i < 2; i++) {
        fields[i].setEnabled(false);
        fields[i].highlight(false);
        fields[i].repaint();
    }

    if (force) {
        this.setStatusText("Игра остановлена.");
    }

    return true;
}

private void setStatusText(String text) {
    this.statusBarLabel.setText(text);
}

private void initWindowForGame() {
    if (this.firstTimeMakeGameWindow) {
        this.firstTimeMakeGameWindow = false;
    }
}

```

```

// Очистить всё
System.out.print("Clearing view...");

this.mainBoxWithStatusBar.removeAll();

System.out.println(" Finished.");

// Создать GUI-компоненты для новой игры...
Box mainBox = Box.createHorizontalBox();

// колонки для игроков
Box box[] = {Box.createVerticalBox(), Box.createVerticalBox()};

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

    System.out.println("playerPanel added");

    // Игровые поля: чужое на другой стороне, своё на своей
    box[i].add(fields[1 - i]);

    mainBox.add(box[i]);
    if (i < 1) {
        // вертикальный зазор между полями
        mainBox.add(Box.createVerticalStrut(10));
    }

    System.out.println("opponentsFieldPanel added");
}

mainBoxWithStatusBar.add(mainBox);

// Статусная панель
Box horBox = Box.createHorizontalBox();
horBox.add(statusBarLeftLabel);
horBox.add(Box.createHorizontalStrut(50));
horBox.add(statusBarLabel);

mainBoxWithStatusBar.add(horBox);
setContentPane(mainBoxWithStatusBar);
}

// turn repaint on
boolean ignoreRepaint = false;
this.setIgnoreRepaint(ignoreRepaint);
for (int i = 0; i < 2; i++) {
    fields[i].setIgnoreRepaint(ignoreRepaint);
}

pack();
setResizable(false);
}

// ----- Создаем меню -----
private void createMenu() {

    menu = new JMenuBar();
    JMenu fileMenu = new JMenu("Меню");

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

        JMenuItem item = new JMenuItem(fileItems[i]);
        item.setActionCommand(fileItems[i].toLowerCase());
    }
}

```

```

        item.addActionListener(new NewMenuListener(this));
        fileMenu.add(item);
    }
    fileMenu.insertSeparator(fileItems.length - 1);

    menu.add(fileMenu);
}

public class NewMenuListener implements ActionListener {

    private GamePanel gamePanel;

    public NewMenuListener(GamePanel gamePanel) {
        this.gamePanel = gamePanel;
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();
        if (GamePanel.fileItems[1].equalsIgnoreCase(command)) {
            // Выход
            boolean stopOk = gamePanel.stopCurrentGame(false);

            if (!stopOk) {
                int answer = JOptionPane.showOptionDialog(
                    null, "Текущая игра не закончена. Прервать её и выйти?", "Внимание",
                    JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE, null,
                    new Object[]{"Да!", "Нет"}, "Нет");

                if (answer == 0) {
                    gamePanel.stopCurrentGame(true);
                } else {
                    return;
                }
            }
            System.exit(0);
        }
        if (GamePanel.fileItems[0].equalsIgnoreCase(command)) {
            System.out.println("New GAME !");

            boolean startOk = gamePanel.startNewGame(false);

            if (!startOk) {
                int answer = JOptionPane.showOptionDialog(
                    null, "Текущая игра не закончена. Прервать её и начать новую игру?",
                    "Внимание", JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE, null,
                    new Object[]{"Да!", "Нет"}, "Нет");

                if (answer == 0) {
                    gamePanel.setVisible(false);
                    new Settings().setVisible(true);
                }
            }
            // debug
            System.out.println("New GAME started.");
        }
    }
}

private class GameObserver implements IGameListener {

    @Override

```

```

    public void shootPerformed(ShootResult shootResult) {
        //рисовка
        String humanReadableResult = " ";
        switch (shootResult) {
            case MISS:
                humanReadableResult += "Промех!";
                break;

            case SHIP_DAMAGED:
                humanReadableResult += "Ранил!";
                break;

            case SHIP_DROWNED:
                humanReadableResult += "Убил!";
                break;
        }
        setStatusText(humanReadableResult);

        repaint();
    }

    @Override
    public void gameFinished(GameModel.GameResult result) {
        stopCurrentGame(false);
        for (GameFieldPanel field: fields){
            field.setInputEnabled(false);
        }
        JOptionPane.showMessageDialog(null, result, "Игра завершена",
JOptionPane.INFORMATION_MESSAGE);
    }

    @Override
    public void exchangePlayer(Player activePlayer) {
        if (activePlayer instanceof HumanPlayer) {
            fields[0].highlight(true);
            fields[1].highlight(false);
            fields[0].setInputEnabled(true);
        } else {
            fields[1].highlight(true);
            fields[0].highlight(false);
            fields[0].setInputEnabled(false);
        }
        repaint();
    }
}
}

```

```

public class GameFieldPanel extends JPanel {

    private GameField _gamefield;
    private Set<UnitButton> _buttons;

    private boolean highlighted;

    // ----- Размеры -----
    private static final int CELL_SIZE = 36; // PREFERRED
    private static final int MAX_SIZE = 25;
    private static final int GAP = 8;
    private static final int MAX_WIDTH = 400;
    private static final int MAX_HEIGHT = 450;
    private static final Color ACTIVE_BORDER_COLOR = Color.BLUE;
    private static final Color INACTIVE_BORDER_COLOR = Color.GRAY;

```

```

public GameFieldPanel() {
    super();
    my_constructor();
}

public GameFieldPanel(GameField field) {
    my_constructor();
    this._gamefield = field;
}

private void my_constructor(){
    this._buttons = new HashSet<>();
    this._gamefield = null;
}

public void init(GameField field, UnitButtonState state){
    if(field != null) {
        this._gamefield = field;
    }

    // Проверить, что состояние допустимо
    assert this._gamefield != null;

    this.setVisible(false);

    this.removeAll();

    createField(state);

    this.setVisible(true);

    this.repaint();
}

public Set<UnitButton> getButtons() {
    return _buttons;
}

public void setInputEnabled(boolean isEnabled) {
    for(JButton cb : _buttons) {
        cb.setEnabled(isEnabled);
    }
}

public void highlight(boolean isOn) {
    this.highlighted = isOn;
    this.repaint();
}

private void createField(UnitButtonState state) {
    int size = this._gamefield.size();
    int actualSize = size + 1;

    JPanel buttonsPanel = new JPanel(true);
    buttonsPanel.setLayout(new GridLayout(actualSize, actualSize));

    Dimension fieldDimension = new Dimension(CELL_SIZE*actualSize, CELL_SIZE*actualSize);

    buttonsPanel.setPreferredSize(fieldDimension);
    buttonsPanel.setMinimumSize(fieldDimension);
    buttonsPanel.setMaximumSize(fieldDimension);
}

```

```

for (int row = 0; row < size; row++)
{
    if( row == 0) { // ADD labels
        Label l = new Label(""); // empty
        buttonsPanel.add(l);
        for (int col = 0; col < size; col++) { // letter
            String label_text = "    " + ((char)('A' + col));
            l = new Label(label_text);
            buttonsPanel.add(l);
        }
    }
    for (int col = 0; col < size; col++)
    {
        if (col == 0) { // ADD labels
            Label l = new Label("    " + (row + 1)); // digit
            buttonsPanel.add(l);
        }
        Cell cell = _gamefield.cellAt(new Location(col, row));
        UIButton button;
        if (cell.getDeck() != null){
            button = new DeckButton( cell.getDeck(), state);
        }
        else{
            button = new CellButton( cell, state);
        }

        button.setEnabled( false );

        buttonsPanel.add(button);
        _buttons.add(button);

        // ожидаем клика
        ActionListener actionlistener = new UnitButtonActionListener();
        button.addActionListener(actionlistener);
    }
}

this.add(buttonsPanel);
this.validate();

int width = 2*GAP + CELL_SIZE*actualSize;
int height = 2*GAP + CELL_SIZE*actualSize;
Dimension panelDimension = new Dimension(width, height);

this.setPreferredSize(panelDimension);
this.setMinimumSize(panelDimension);
this.setMaximumSize(panelDimension);
}

/** Рисуем поле */
@Override
public void paintComponent(Graphics g) {

    // Отрисовка фона
    int width  = getWidth();
    int height = getHeight();

    g.setColor(highlighted? ACTIVE_BORDER_COLOR : INACTIVE_BORDER_COLOR);
    g.fillRect(GAP-3, GAP-5, width-GAP-2, height-GAP-4);
    g.setColor(Color.BLACK);    // восстанавливаем цвет пера

```

```

    }
}

public abstract class UIButton extends JButton {

    protected UIButtonObserver unitButtonObserver;

    protected UIButtonState unitButtonState;

    public UIButton(UIButtonState state) {
        super();

        this.unitButtonObserver = new UIButtonObserver();

        this.unitButtonState = state;
    }

    public abstract void shoot();

    /**
     * Рисуем клетку на кнопке
     */
    @Override
    public void paintComponent(Graphics g) {
        if (this.unitButtonState == UIButtonState.LOCKED) {
            paintComponentLocked(g);
        } else {
            paintComponentOpened(g);
        }
    }

    protected void paintComponentLocked(Graphics g) {
        // Отрисовка фона
        int width = getWidth();
        int height = getHeight();

        // границы для клетки
        Rectangle rectForCell = new Rectangle();
        rectForCell.x = 0;
        rectForCell.y = 0;
        rectForCell.width = width;
        rectForCell.height = height;

        g.setColor(Color.LIGHT_GRAY);
        g.fillRect(0, 0, width, height);

        g.setColor(Color.BLACK);    // восстанавливаем цвет пера
    }

    protected abstract void paintComponentOpened(Graphics g);

    private class UIButtonObserver implements IShorableUnitListener {

        @Override
        public void unitDamaged(IShorableUnit unit) {
            if(unit.getState() == IShorableUnit.ShorableUnitState.Broken){
                unitButtonState = UIButtonState.OPENED;
            }
        }
    }
}

```

## 4 Вторая итерация разработки

### 4.1 Функциональные требования (сценарии)

#### 1) Главный успешный сценарий — Победил один из игроков

1 Игрок выбирает режим новой игры

2 Игрок задает размер полей (от 10 до 25)

3 Система расставляет корабли на поле игрока и поле соперника случайным образом, не допуская соприкосновений и пересечений

4 Система назначает игрока-человека текущим

4 Делать

4.1 Делать

Текущий игрок выбирает 1 клетку на поле соперника для выстрела

Пока игрок не промахнулся или игра не завершена

4.2 Система передает ход другому игроку, если игра не завершена, и текущий игрок промахнулся

Пока игра не завершена

5 Система определяет победителем того игрока, у кого остались недобитые корабли

#### 1.2) Альтернативный сценарий главного успешного сценария — Досрочное завершение игры (всей программы)

1 Сценарий начинается в любой точке сценария 1), когда игрок инициирует завершение игры

2 Система завершает свою работу без каких-либо запросов и предупреждений

#### 1.3) Альтернативный сценарий главного успешного сценария — Переигровка

1 Сценарий начинается в любой точке сценария 1), когда игрок инициирует начало новой игры



2 Если ранее стартовавший сеанс игры еще не выявил победителя

2.1 Система сообщает, что игра еще не завершена и запрашивает у игрока, действительно ли следует начать новый сеанс игры

2.2 Если игрок подтверждает начало новой игры

2.2.1 Система завершает текущий сеанс игры и запускает главный успешный сценарий

Иначе если игрок не желает начинать игру заново

2.2.2 Система продолжает выполнение главного успешного сценария

### **2.1) Система определяет результат выстрела - «Попал»**

1 Система проверяет, есть ли на выбранной клетке палуба

2 Т.к. на клетке есть палуба и не выполнены все условия потопления корабля, палуба изменяет свое состояние на «поврежденная» или «сломанная» в зависимости от вида самой палубы.

3 Система сигнализирует игроку: «Попал»

4 Игрок продолжает свой ход

### **2.2) Система определяет результат выстрела - «Убил»**

1 Система проверяет, есть ли на клетке палуба

2 Т.к. на клетке есть палуба и выполнены все условия потопления каждой палубы и корабля в целом, система сигнализирует игроку: «Убил»

3 Игрок продолжает свой ход

### **2.3) Система определяет результат выстрела - «Промых»**

1 Система проверяет, есть ли на клетке палуба

2 Т.к. клетка не содержит палубы, клетка меняет свое состояние

3 Система сигнализирует игроку о том, что он промахнулся

4 Система передает ход другому игроку

#### **2.4) Система определяет результат выстрела - «Повторный выстрел»**

- 1 Система проверяет состояние клетки или палубы, находящейся на ней.
- 2 Т.к. обнаружена сломанная клетка или сломанная палуба, система не засчитывает ход игроку
- 3 Система предоставляет игроку еще одну попытку выстрела

## 4.2 Словарь предметной области

Палуба — составляющая корабля, занимающая одну клетку

Промех — результат выстрела в клетку поля, не содержащую палубу

Попал — результат выстрела в некоторую палубу, означающий либо полное ее разрушение, либо частичное повреждение

Убил — результат полного разрушения всех палуб, входящих в данный корабль

Состояния клетки:

- Целая клетка – клетка поля, в которую еще ни разу не стреляли
- Сломанная клетка – клетка поля, в которую либо уже стреляли, либо уже открыли после потопления находящегося по соседству корабля

Состояния части корабля:

- Целая – палуба, в которую еще ни разу не стреляли
- Поврежденная – палуба, в которую стреляли хотя бы один раз, и которая еще не убита
- Сломанная – палуба, в которую стреляли необходимое количество раз для ее потопления

Состояния корабля:

- Активный – еще не потопленный корабль: хотя бы одна его часть не сломанная
- Потопленный – все его части сломанные

Стандартный режим игры – используются только стандартные корабли

Нестандартный режим игры – все стандартные корабли заменяются на выбранные случайным образом нестандартные корабли из списка кораблей.

По соседству – нахождение объекта (палубы, корабля, клетки) в 1 из 8 клеток вокруг данной клетки.

Игрок – игрок-человек или игрок-бот

Поле игрока – поле игрока-человека, содержащее стандартные или нестандартные корабли (в зависимости от режима игры). Обстреливается игроком-ботом

Поле соперника – поле игрока-бота, стандартные или нестандартные корабли (в зависимости от режима игры). Обстреливается игроком-человеком.

## 4.3 Структура программы на уровне классов

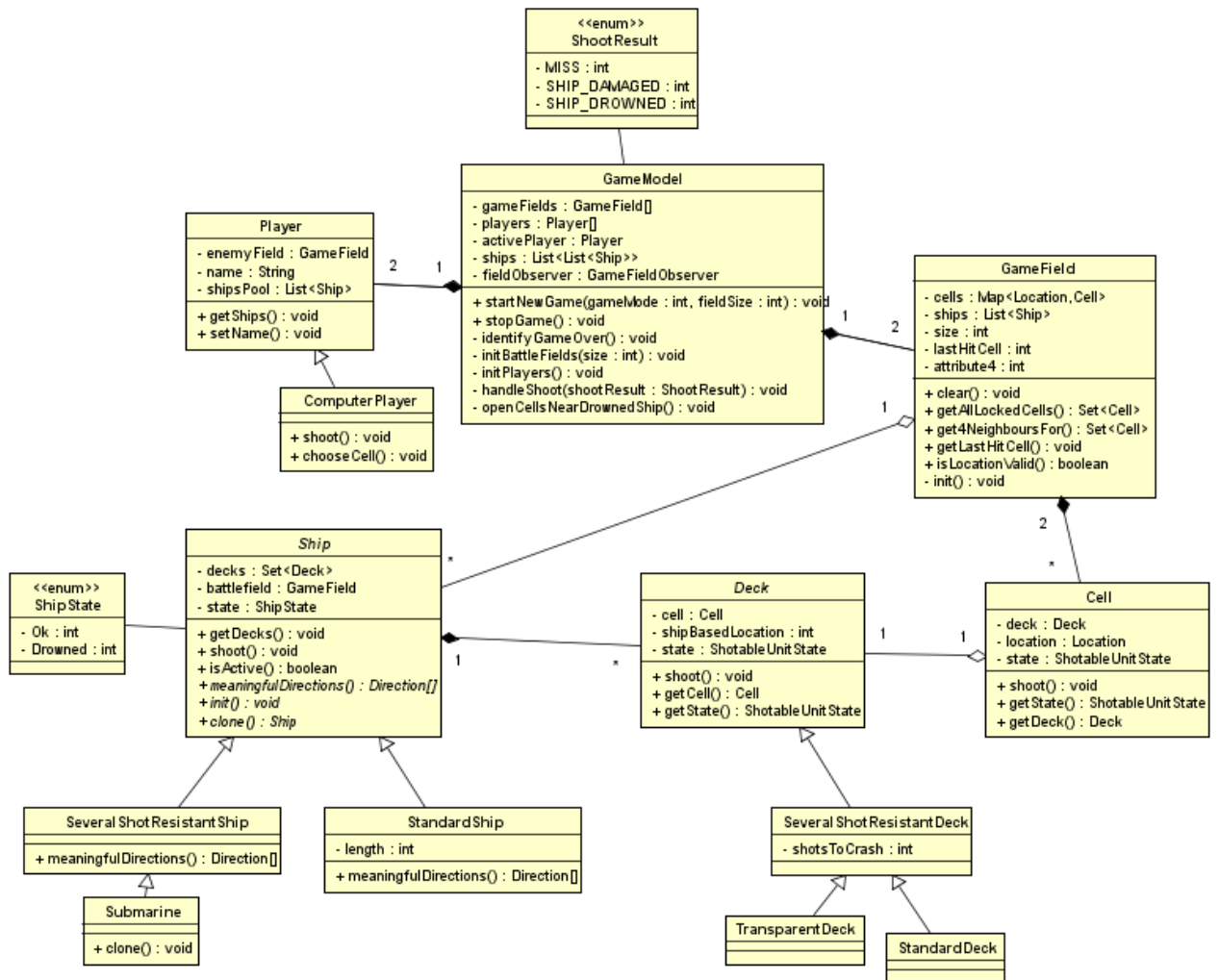


Диаграмма классов вычислительной модели

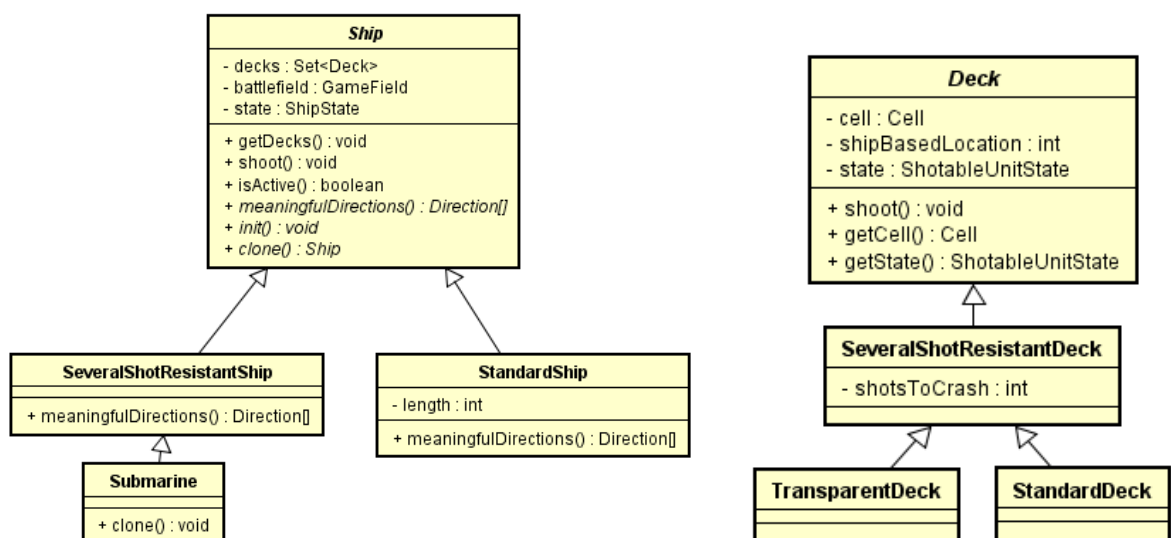
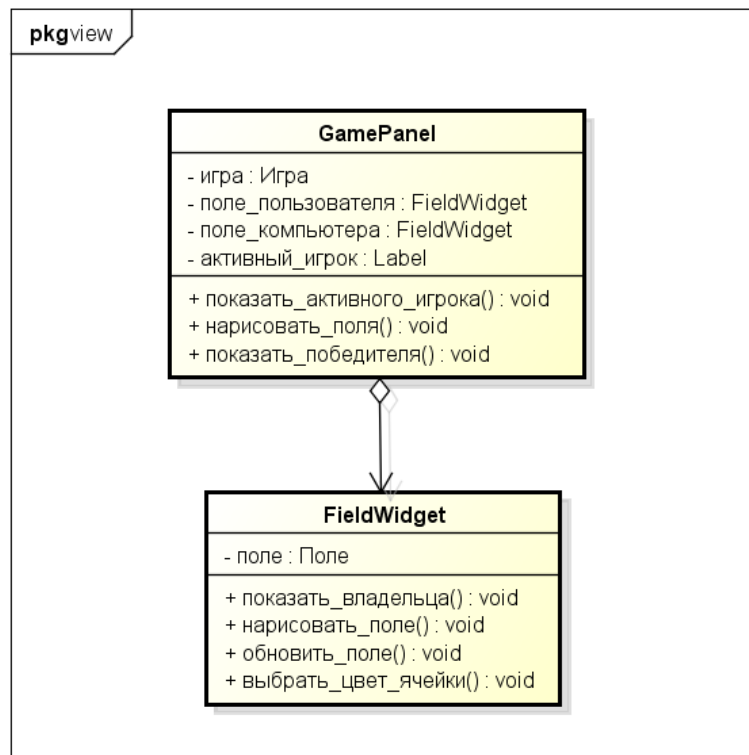


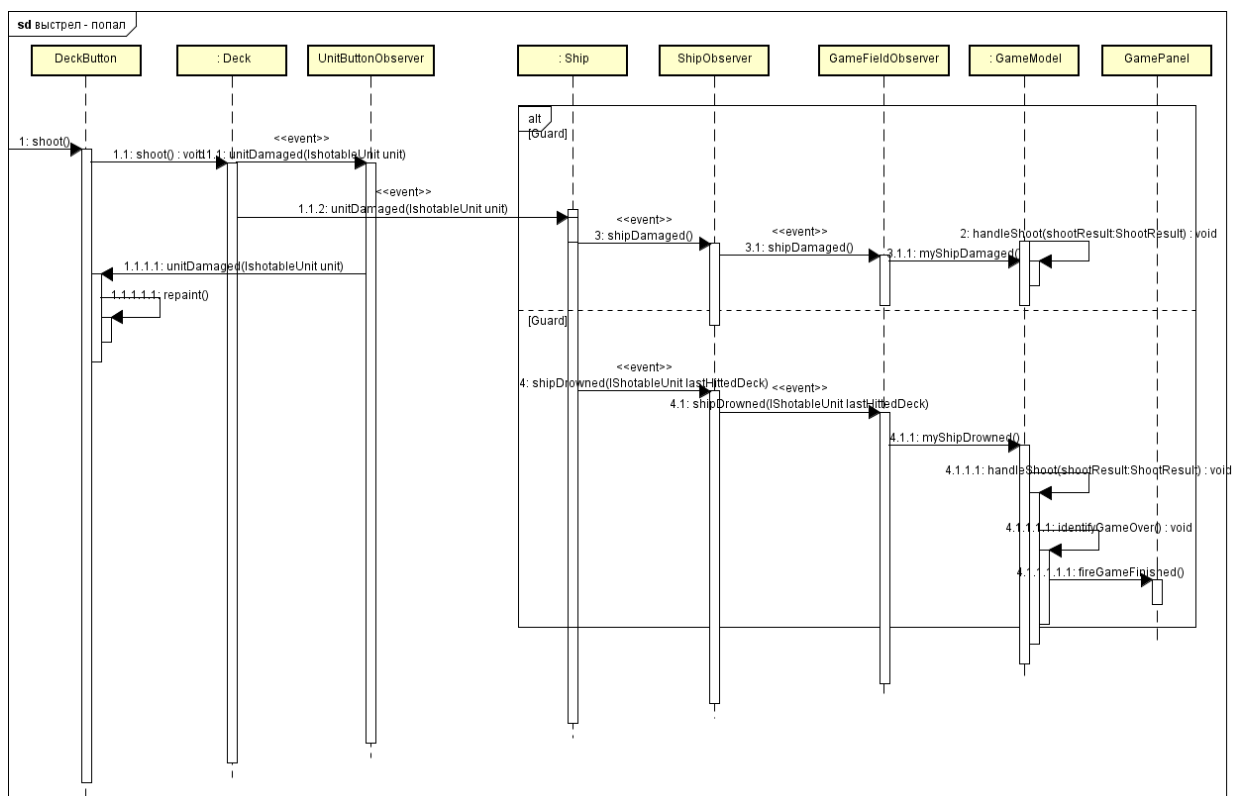
Диаграмма классов точки расширения



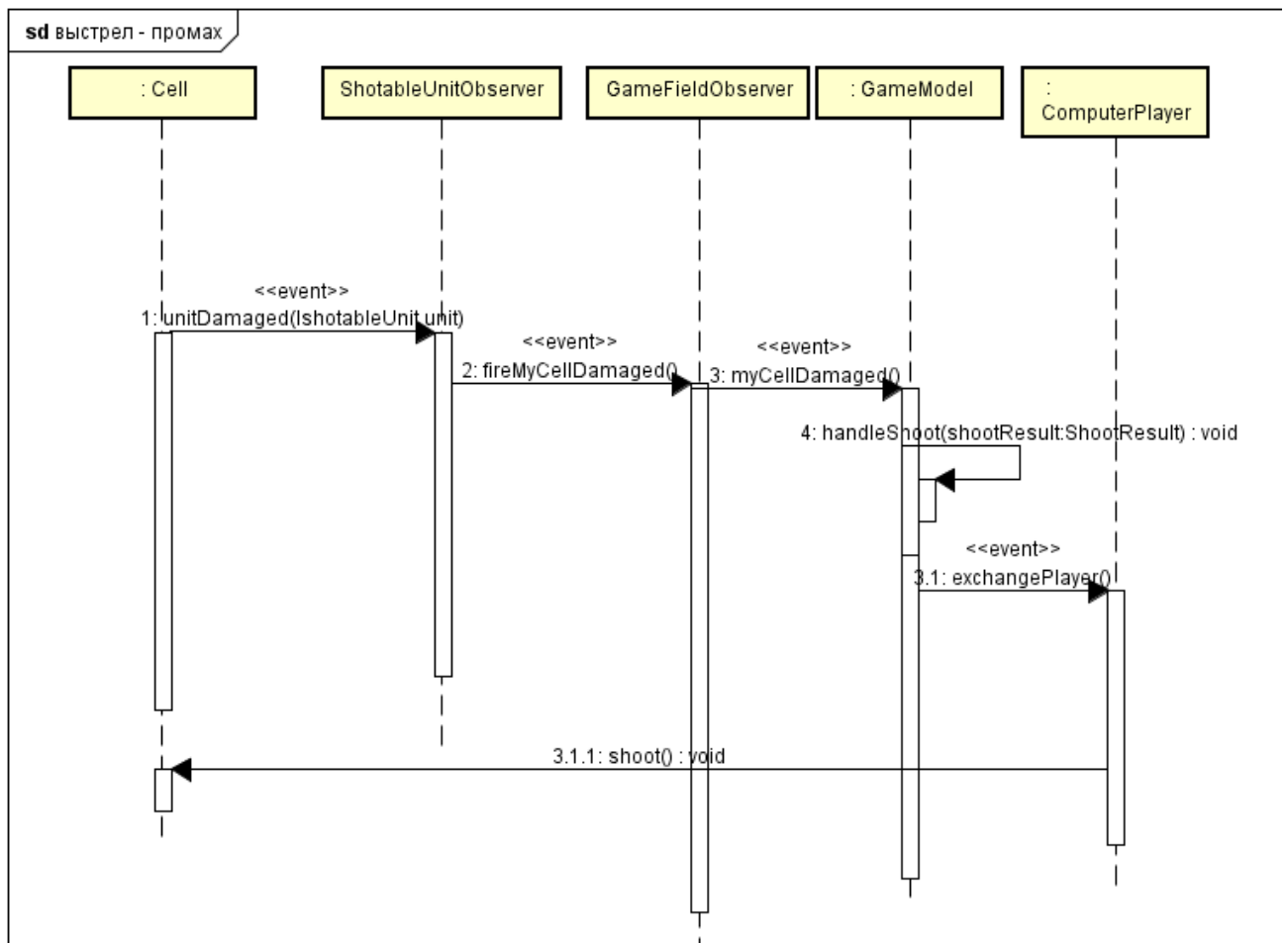
powered by Astah

Диаграмма классов представления

## 4.4 Типовые процессы в программе



Выстрел - попадание



Выстрел – промах

## 4.5 Человеко-машинное взаимодействие

При запуске игры пользователь видит экран настроек (рис. 1), на котором предлагается выбрать размер игрового поля и тип игры.

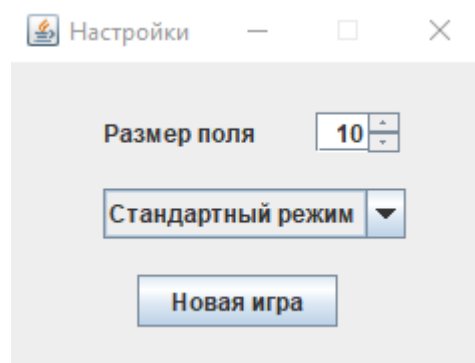


Рисунок 1 – Начальный экран программы

Общий вид экрана игры представлен на рисунке 2. На нем располагаются поля с пользователя и компьютера. Поле с кораблями пользователя открыто, на

нем цветом отмечены корабли (черный) и море (голубой). В данном режиме на поле находятся только подводные лодки. Синяя обводка вокруг поля показывает, что сейчас нужно стрелять по нему.

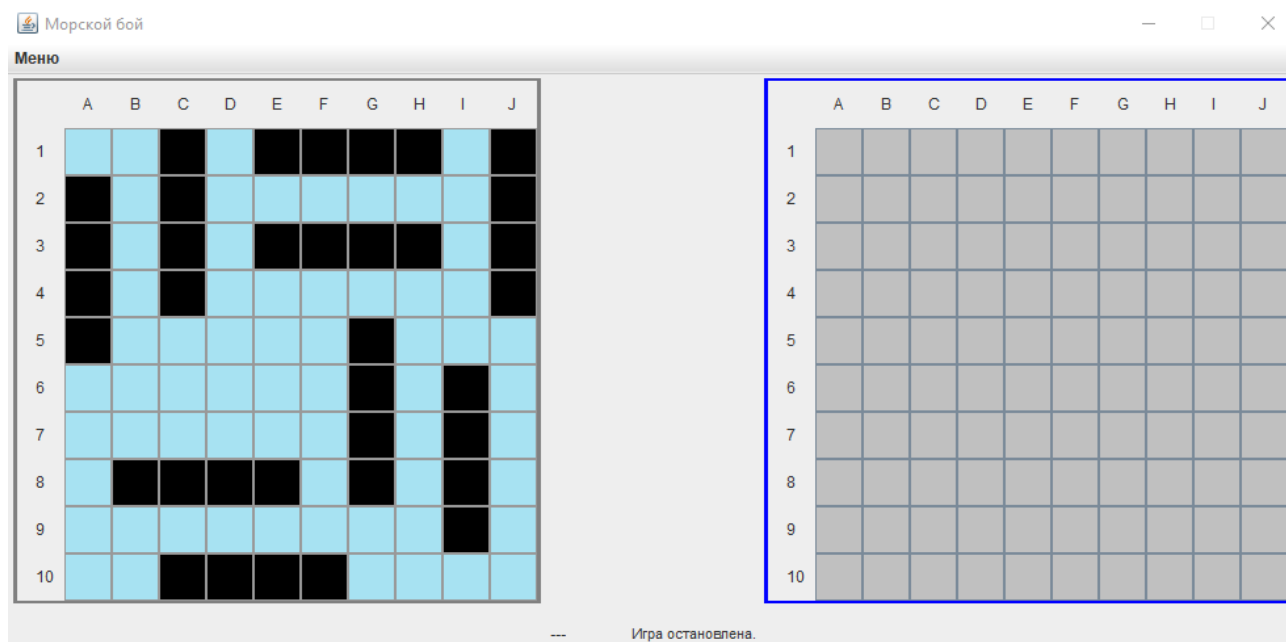


Рисунок 2 – Экран игры

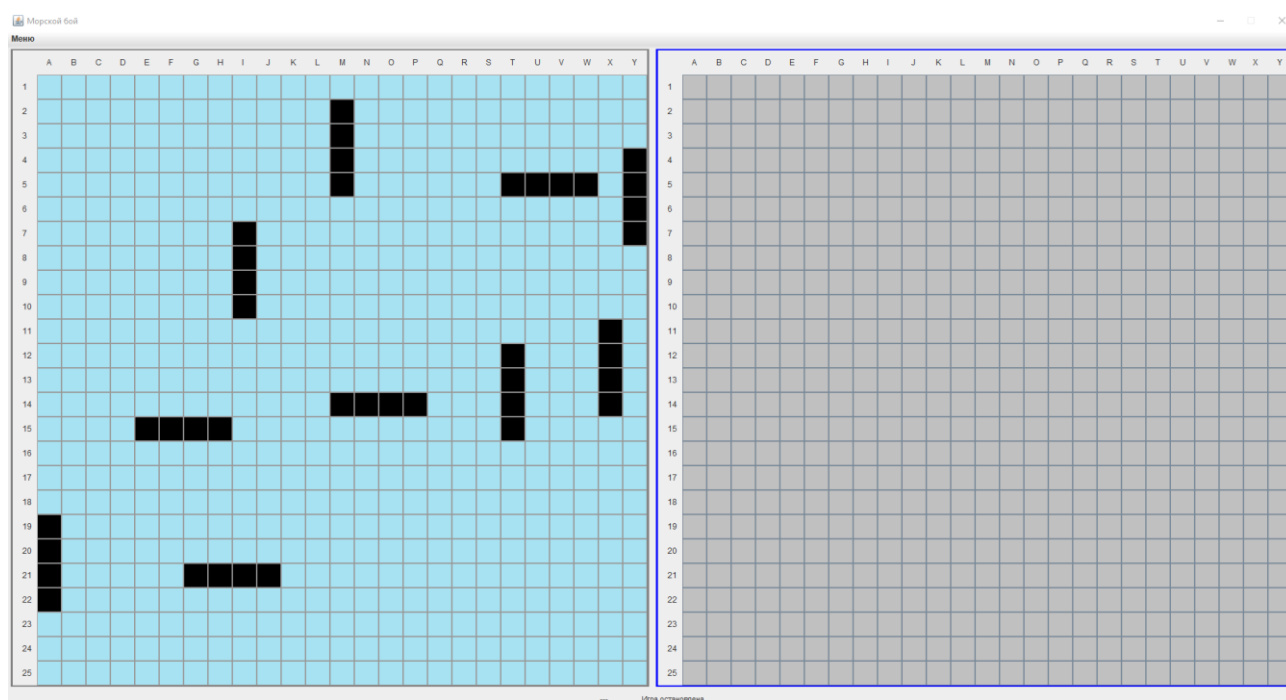


Рисунок 3 – Экран игры с полями 25x25

Для выстрела пользователь кликает по ячейке, в которую он хочет выстрелить. Ячейка отмечается пораженной: если попали в море, то внутри клетки рисуется синий круг, если попали по палубе впервые, то попадание



засчитывается, но клетка никак не помечается, если попали по рубке, то внутри клетки рисуется красный крест (рис. 4-5).



Рисунок 4 – Клетка корабля

а) не поражена

б) поражена

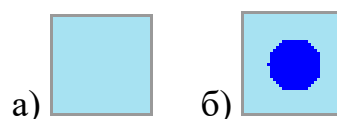


Рисунок 5 – Клетка моря

а) не поражена

б) поражена

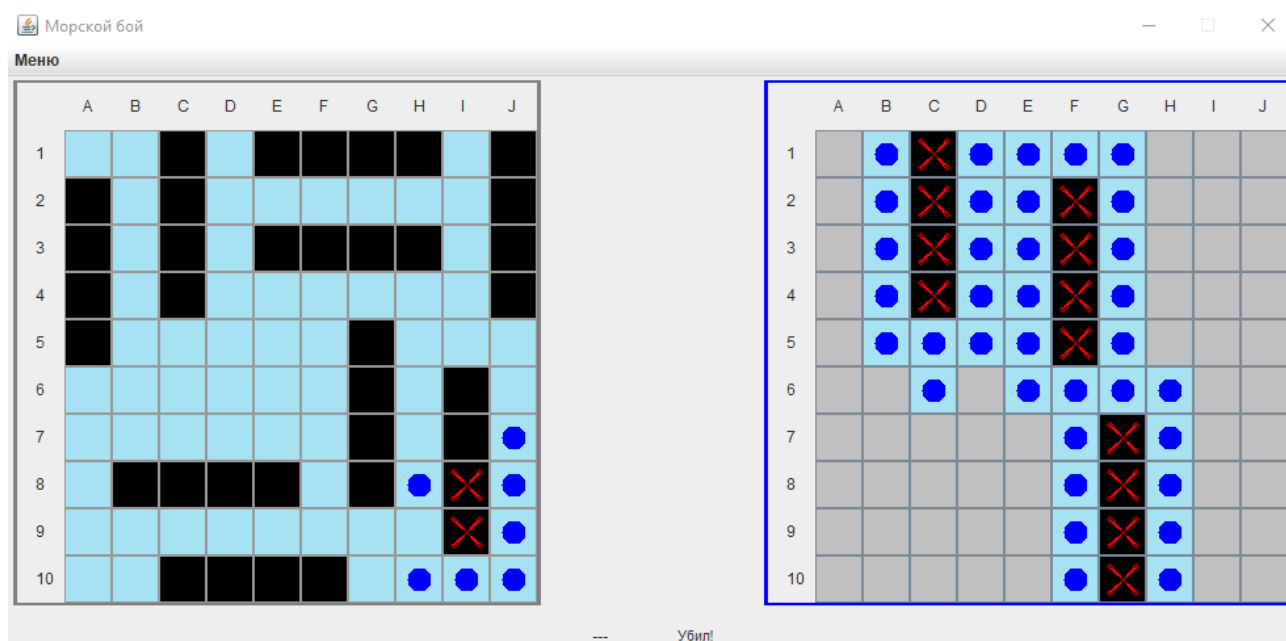


Рис. 6 – Общий вид игры с отметками выстрелов

Пользователь не может совершить выстрел по полю со своими кораблями.

Выстрел компьютера происходит автоматически, результат выстрела отображается на соответствующем поле.

При завершении игры выводится окно с сообщением о результате игры, представленное на рисунке 7. При этом оба поля открываются.

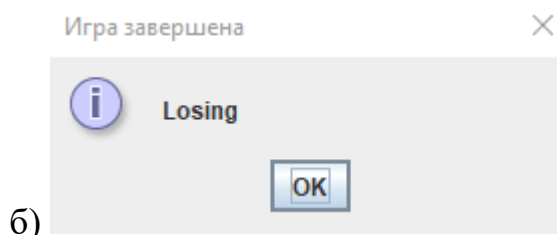
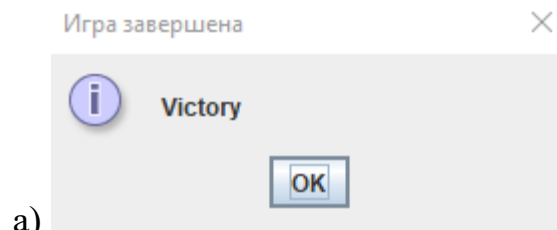


Рисунок 7 – Окно с сообщением о результате игры

а) победа человека

б) победа компьютера

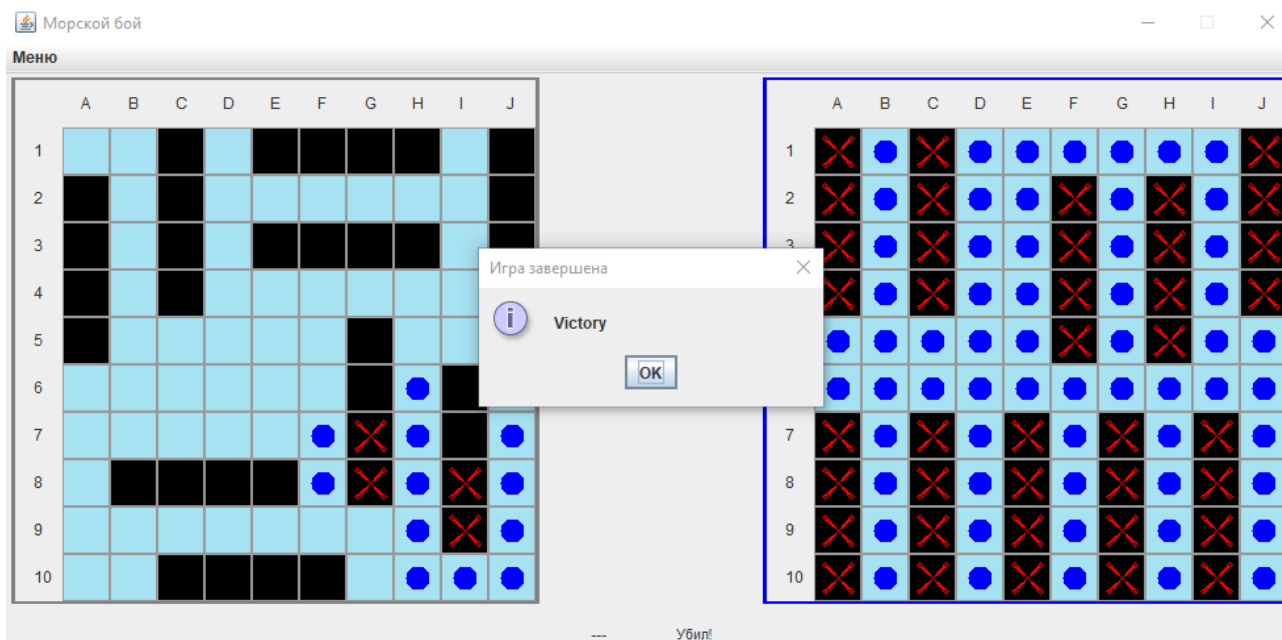


Рисунок 8 – Общий вид игры при завершении

После закрытия окна с сообщением о победе можно начать новую игру через меню «Игра» или выйти из программы.

## 4.6 Реализация ключевых классов

```
public abstract class SeveralShotResistantShip extends Ship {

    SeveralShotResistantShip(){
        super();
        init();
    }
    @Override
    public Direction[] meaningfulDirections() {
        return new Direction [] {Direction.NORTH, Direction.EAST, Direction.SOUTH,
Direction.WEST} ;
    }
}

public class Submarine extends SeveralShotResistantShip {
    public Submarine() {
        super();
    }

    private Submarine(Submarine other) {
        this();
    }

    @Override
    protected void init() {
        Deck deck;

        for (int i = 0; i < 4; i++) {
            if (i != 1) {
                deck = new TransparentDeck(this, new Location(i, 0));
                this.decks.add(deck);
            } else {
                deck = new StandardDeck(this, new Location(i, 0));
                this.decks.add(deck);
            }
            //теперь корабль следит за этой палубой//
            deck.addModelUnitListener(shipObserver);
        }
    }

    @Override
    public Ship clone() {
        return new Submarine(this);
    }
}

public class TransparentDeck extends SeveralShotResistantDeck {

    public TransparentDeck(Ship ship, Location shipBasedLocation) {
        super(ship, shipBasedLocation);
        this.shotsToCrash = 2;
    }
}

public class ShipPoolFactory {

    private ArrayList<Ship> allNonStandardShips = new ArrayList<>();

    ShipPoolFactory(){
```

```

        allNonStandardShips.add(new Submarine());
    }

    public ArrayList<Ship> createStandardShipPool(){
        ArrayList<Ship> standardShips= new ArrayList<>();
        standardShips.add(new StandardShip(4));

        standardShips.add(new StandardShip(3));
        standardShips.add(new StandardShip(3));

        standardShips.add(new StandardShip(2));
        standardShips.add(new StandardShip(2));
        standardShips.add(new StandardShip(2));

        standardShips.add(new StandardShip(1));
        standardShips.add(new StandardShip(1));
        standardShips.add(new StandardShip(1));
        standardShips.add(new StandardShip(1));

        return standardShips;
    }

    public ArrayList<Ship> createNonStandardShipPool(){
        ArrayList <Ship> nonStandardShips = new ArrayList<>();
        for (int i=0; i < 10; i++){
            nonStandardShips.add(Objects.requireNonNull(Utils.chooseOne(allNonStandardShips)).clone());
        }
        return nonStandardShips;
    }
}

```

## 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>