Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра «Компьютерные технологии»

П.А. Петрошенко, Г.А. Корнеев, А.А. Шалыто

ИГРА “МОРСКОЙ БОЙ”

Объектно–ориентированное программирование
с явным выделением состояний

Проектная документация

Проект создан в рамках
«Движения за открытую проектную документацию»
http://is.ifmo.ru

Санкт–Петербург
2005

Оглавление

"*Морской бой* – вооруженное
противоборство соединений (групп),
частей и кораблей с противником…"[1]

1. Введение

Для алгоритмизации и программирования игры "Морской бой" была использована технология, называемая "Объектно-ориентированное программирование с явным выделением состояний", которая была предложена в работах Шалыто А. А. и Туккеля Н. И. [1, 2]. Она представляет собой совместное использование объектно-ориентированного программирования и SWITCH-технологии [3].

Настоящая работа содержит документацию, состоящую из диаграммы классов, структурных схем классов, схем связей и графов переходов автоматов, исходного текста программы и фрагментов протоколов.

Апплет, приложение и исходные коды программы можно найти на сайте http://is.ifmo.ru.

2. Реализация

Автоматы реализованы в виде методов классов. Всего классов в программе более сорока, из них пять автоматных, каждый из которых содержит по одному автомату. Четыре автомата являются вложенными — их вызов осуществляется из состояний других автоматов. Автоматы взаимодействуют между собой по вложенности и с помощью обмена номерами состояний, а с внешней средой — при обработке событий, которых всего два – "Нажатие кнопки мыши" (*MouseEvent*) и "Нажатие кнопки "Начать заново". Источниками событий (*EventProvider*), таким образом, являются, обработчик событий мыши на игровом поле и соответствующая кнопка (*ButtonEventProvider*).

Для обработки событий используется очередь необработанных событий (*EventStorage*). Каждый автомат, находясь в состоянии, в котором ожидается соответствующее событие, обращается к очереди необработанных событий, и, если в ней есть событие нужного типа, обрабатывает его, удаляя из очереди. Если такого события в очереди нет, то автомат ждет до тех пор, пока оно не произойдет и не поступит в очередь для обработки. Все автоматы используют единую очередь необработанных событий.

Для реализации данного проекта использован объектно-ориентированный язык *Java* и среда разработки *IntelliJ IDEA 4*.

При проектировании программы в части пользовательского интерфейса применялась стандартная архитектура "Модель-Вид-Контроллер" (*Model-View-Controller*). Одно из применений данной схемы — реализация игрового поля. Класс, представляющий модель поля, реализует внутреннюю структуру: физические характеристики и состояния ячеек игрового поля. В свою очередь, класс "Представление Игрового Поля" (*Field*) для этой модели определяет и реализует все фенкциональные особенности и детали визуализации. Изменения в текущие состояния полей вносят автоматные классы.

Схема "Модель-Вид-Контроллер" использована также и в механизме протоколирования состояний автоматов. Класс, реализующий модель системы протоколирования (*LoggingModel*), определяет ее функциональность и структуру. Состояния

[1] Поленин В.И. Морской бой, применение сил в морском бою или тактическая операция? // Военная мысль. 2003, № 10.

протоколов визуализируются средствами представлениями протоколов (классом, реализующим специальный интерфейс *LogView*).

В программе использованы следующие паттерны объектно-ориентированного проектирования: Одиночка (*Singleton*) и Наблюдатель (*Observer*).

При отладке программы применялись протоколы, являющиеся неотъемлемой частью использованной технологии. Отдельно поддерживается включение в протокол информации об объектах, автоматах, входных переменных и выходных воздействиях. При протоколировании входных переменных и выходных воздействий в протокол также добавляется информация об обрабатываемых автоматами событиях. В начале строки, содержащей информацию, относящейся к событию, ставится знак “#”.

Независимо ведется протоколирование основных этапов работы программы для логического разбиения всего протокола на отдельные части, такие, как “Расстановка кораблей Пользователя”, “Расстановка кораблей Робота”, “Сражение”. Протоколирование осуществляется только по изменению состояния хотя бы одного автомата. Поэтому несмотря на циклический вызов “Главного автомата”, протоколирование производится эпизодически. При наличии полного протокола можно восстановить ход работы программы. Полный протокол является “самописцем” внутреннего жизненного цикла программы, однако имеющейся в нем информации для восстановления хронологии боя недостаточно.

3. Описание правил игры “Морской бой”

На игровой площадке размером 10 на 10 клеток *Пользователь* расставляет один корабль из четырех клеток, два корабля из трех клеток, три корабля из двух клеток и четыре корабля размером в одну клетку. При этом корабль представляет собой последовательность соседних клеток, стоящих на одной вертикали или на одной горизонтали. Соседние корабли не должны иметь общих точек.

Противником *Пользователя* является *Робот* (Компьютер), который автоматически расставляет корабли на своем поле по указанным выше правилам.

После расстановки начинается бой. Он представляет собой поочередные выстрелы *Пользователя* и *Робота*. При попадании в корабль противника участник боя получает возможность проведения внеочередного выстрела. Игра заканчивается при уничтожении одним из участников всех кораблей противника.

4. Постановка задачи

Целью настоящей работы является апробирование объектно-ориентированного программирования с явным выделением состояний при создании игры — классического варианта игры “Морской бой”, которая имеет пользовательский интерфейс, представленный на рис. 1.

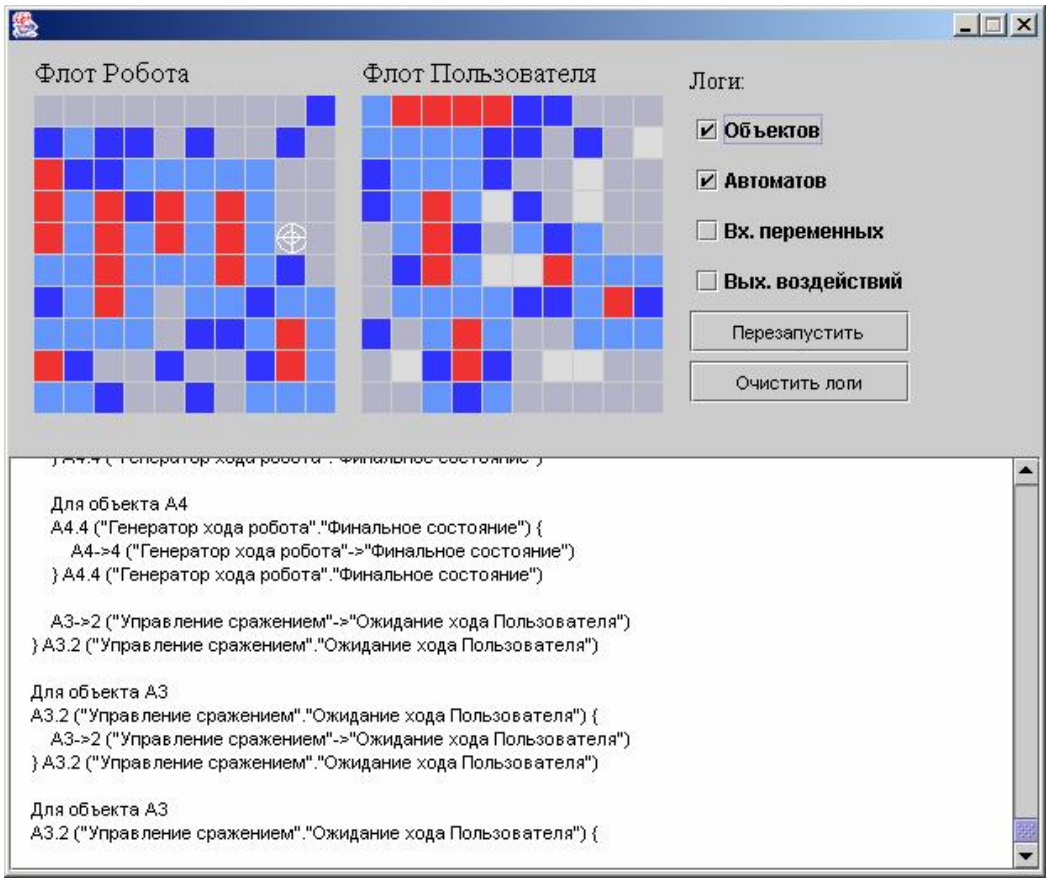

Рис. 1. Внешний вид программы

Отметим, что при создании игры не ставилась цель выбора оптимального алгоритма игры *Роботом*.

Основная идея алгоритма выбора роботом клетки для удара достаточна проста. Сначала ищется самая длинная вертикальная или горизонтальная последовательность непростреленных клеток. Затем из нее случайным образом выбирается клетка для удара.

5. Диаграмма автоматных классов

Диаграмма автоматных классов, с которыми работает аплет, представлена на рис. 2.

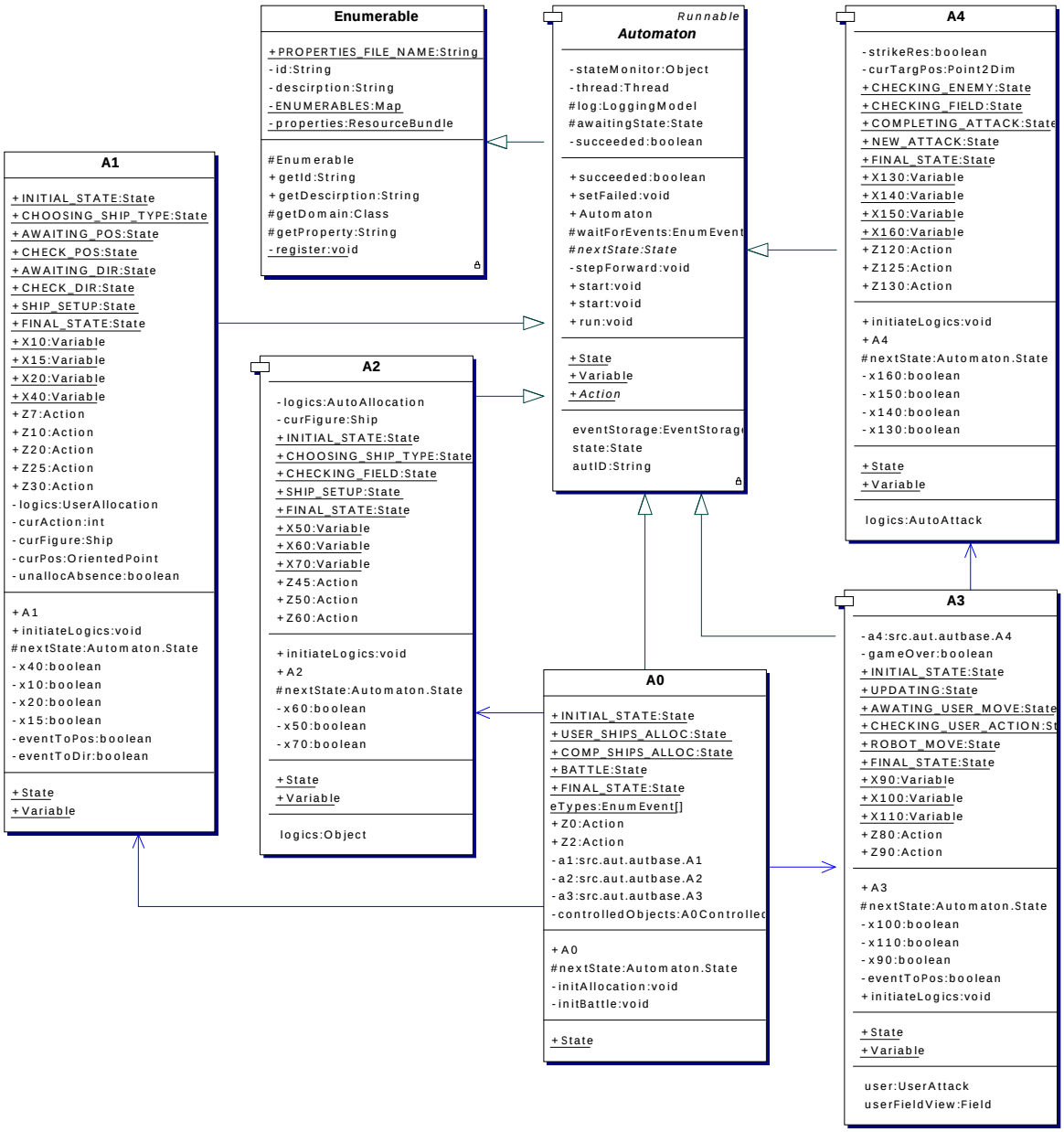

Рис. 2. Диаграмма автоматных классов

Диаграмма 2 иллюстрирует отношение автоматных классов. Из рисунка следует, что объект центрального автомата *A0* содержит в себе и зависит от объектов трех автоматных классов (*A1*, *A2*, *A3*). Принадлежность автоматных объектов *A1*, *A2* и *A3* объекту класса *A0* продиктована вложенностью автоматов.

Связка классов *Automaton→Enumerable* реализует паттерн объектно-ориентированного программирования *Enum*. Это гарантирует, что для каждого автоматного класса будет создан единственный объект. Класс *Automaton* является базовым автоматным классом. В нем реализована общая для всех автоматов функциональность.

6. Класс *Основной Процесс* (*MainProcess*)

6.1. Словесное описание

Класс *Основной Процесс* является главным автоматным классом. Он реализует втомат *A0*, в состояния которого вложены остальные автоматы. Для взаимодействия с внешней средой и фиксации действий *Пользователя* предусмотрены несколько вспомогательных методов.

6.2. Структурная схема класса

Структурная схема класса приведена на.рис. 3.

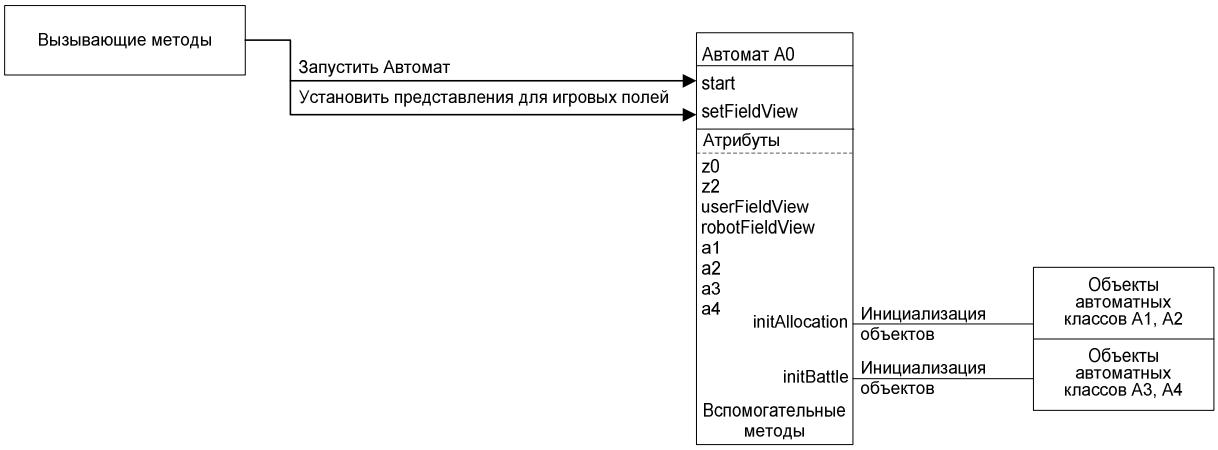

Рис. 3. Структурная схема автоматного класса A0

6.3. *Главный автомат* (*A0*)

6.3.1. Словесное описание

В состояниях главного автомата *A0* вложены автоматы расстановки кораблей *Пользователя*, расстановки кораблей *Робота* и игрового процесса.

6.3.2. Схема связей

На рис. 4 представлена схема связей автомата *A0*.

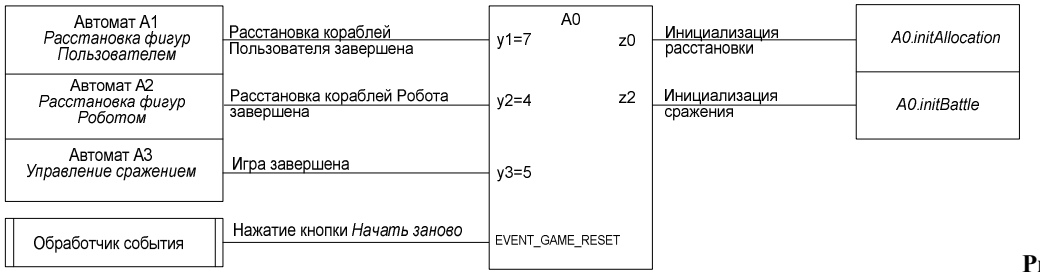

Рис. 4. Схема связей главного автомата *A0*

6.3.3. Граф переходов

На рис. 5 представлен граф переходов автомата *A0*.

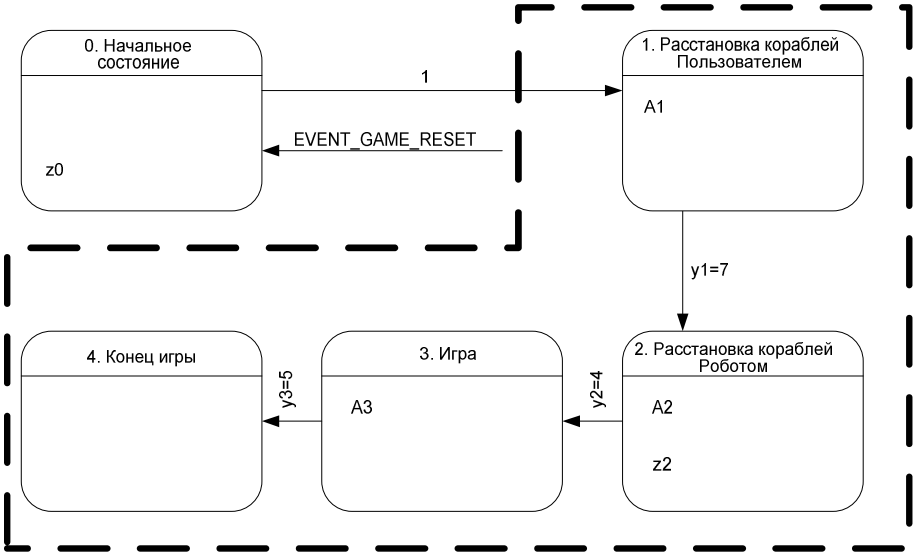

Рис. 5. Граф переходов главного автомата *A0*

7. Класс *Размещение Кораблей Пользователем* (*UserAlloc*)

7.1. Словесное описание

Класс *Размещение Кораблей Пользователем* содержит автомат расстановки кораблей *Пользователя* (*A1*).

7.2. Структурная схема класса

Структурная схема класса представлена на рис. 6.

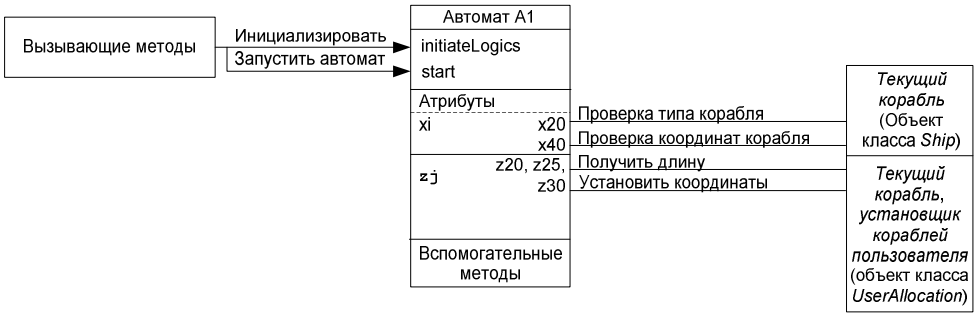

Рис. 6. Структурная схема класса *A1*

7.3. Автомат *Расстановка кораблей Пользователя* (*A1*)

7.3.1. Словесное описание

Этот автомат выполняет контроль расстановки *Пользователем* своих кораблей. Автомат производит выбор типа устанавливаемого корабля, проверку корректности выбора позиции и направления корабля. При успешном завершении проверки производится установка корабля.

7.3.2. Схема связей

На рис. 7 представлена схема связей этого автомата *A1*.

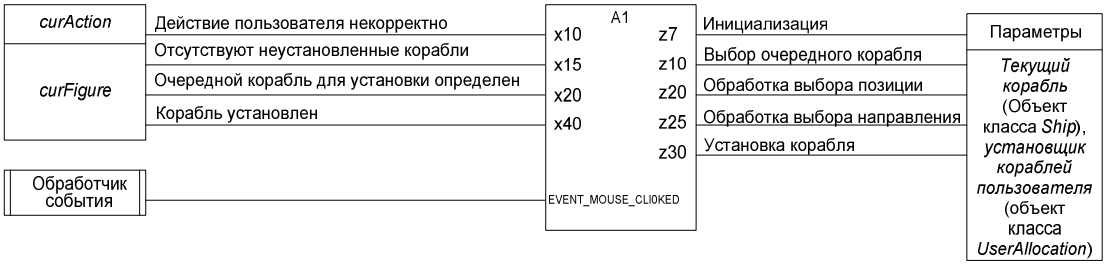

Рис. 7. Схема связей автомата *Расстановка кораблей Пользователя A1*

7.3.3. Граф переходов

На рис. 8 представлен граф переходов автомата *A1*.

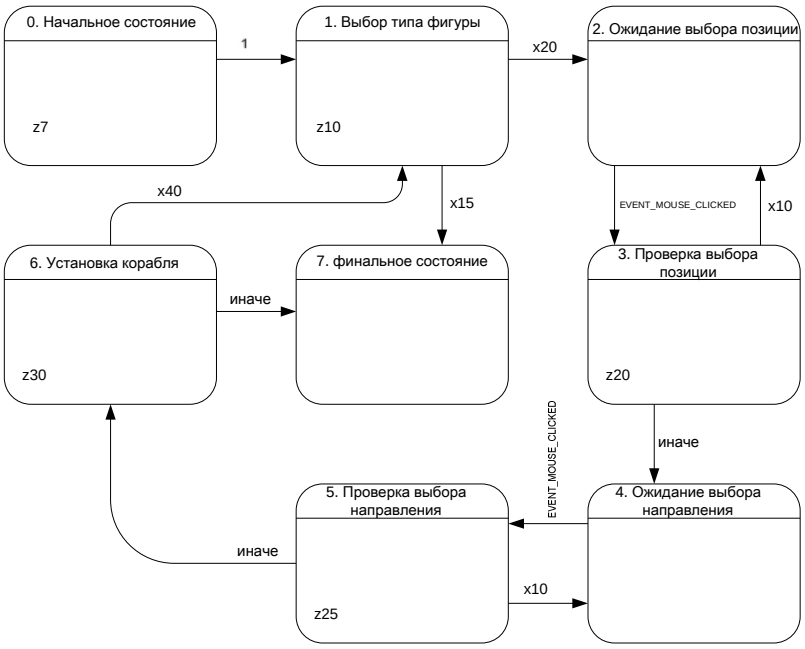

Рис. 8. Граф переходов автомата *Расстановка кораблей Пользователя A1*

8. Класс *Размещение кораблей Роботом* (*RobotAlloc*)

8.1. Словесное описание

Класс *Расстановка кораблей Роботом* функционально “похож” на класс *Расстановка кораблей Пользователем (A2)*. Процесс установки очередного корабля реализован в автомате расстановки кораблей *Робота*.

8.2. Структурная схема класса

Структурная схема класса представлена на рис. 9.

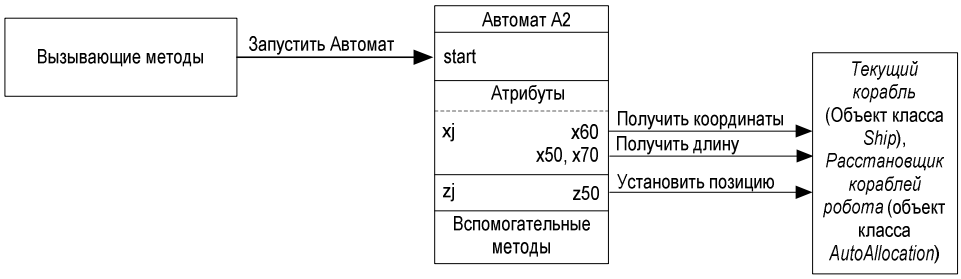

Рис. 9. Структурная схема класса *A2*

8.3. Автомат *Расстановка кораблей Робота* (*A2*)

8.3.1. Словесное описание

Автомат расстановки кораблей *Роботом* автоматически расставляет корабли на игровой площадке. Автомат производит выбор типа очередного устанавливаемого корабля, выбор позиции и направления установки, а также осуществляет саму установку. Данный автомат вызывается из главного автомата программы.

8.3.2. Схема связей

На рис. 10 представлена схема связей автомата *A2*.

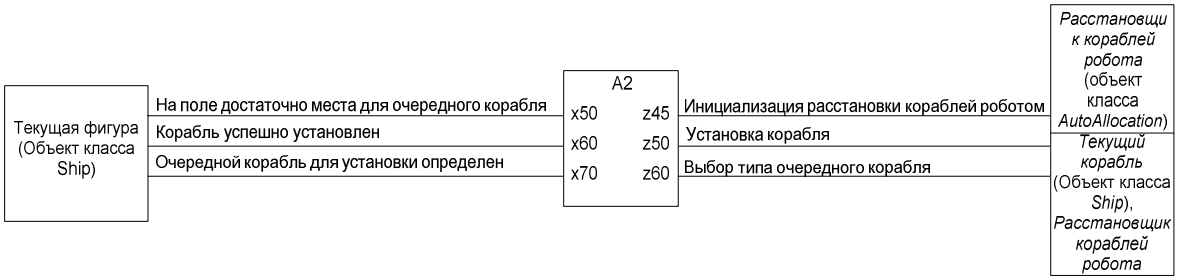

Рис. 10. Схема связей автомата *Расстановка кораблей Робота A2*

8.3.3. Граф переходов

На рис. 11 представлен граф переходов автомата A2.

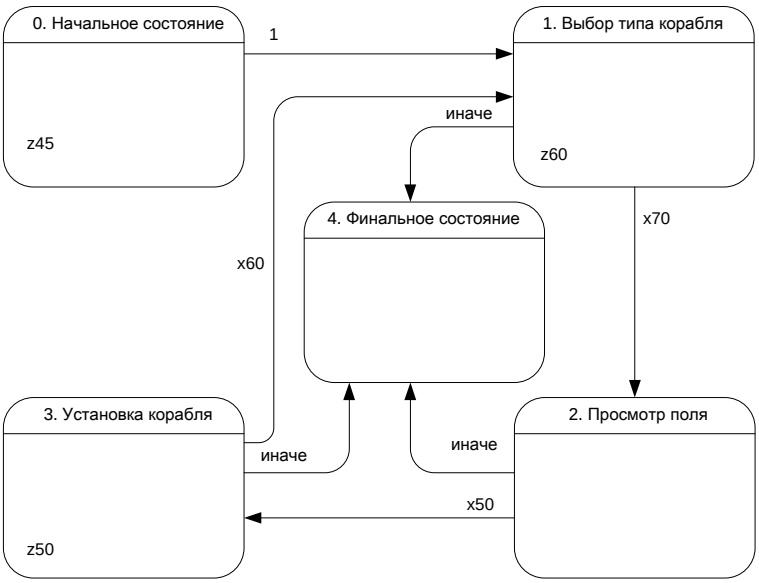

Рис. 11. Граф переходов автомата *Расстановка кораблей Робота А2*

9. Класс *Процесс игры* (*GameProcess*)

9.1. Словесное описание

Класс *Процесс Игры* реализует автомат управления игрой А3, обеспечивающий управление боем *Пользователя* и *Робота*.

9.2. Структурная схема класса

Структурная схема класса представлена на рис. 12.

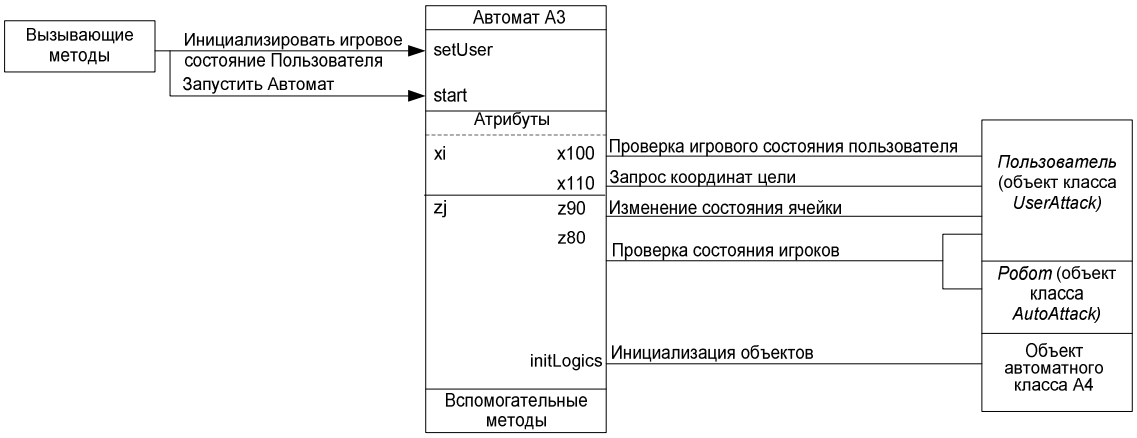

Рис. 12. Структурная схема класса *А3*

9.3. Автомат *Управление игрой* (*А3*)

9.3.1. Словесное описание

Автомат управления игрой реализует один такт игры, в который входит выстрел *Пользователя* и выстрел *Робота*. Процесс выполнения выстрела *Роботом* (один ход) реализован во вложенном в данный автомат автомате *А4*. При успешном выстреле *Пользователя* (при поражении им цели *Робота*) вложенный автомат *А4* не вызывается. Этим обеспечивается предоставление *Пользователю* внеочередного хода. Данный автомат вызывается из главного автомата программы.

9.3.2. Схема связей

На рис. 13 представлена схема связей переходов автомата *А3*.

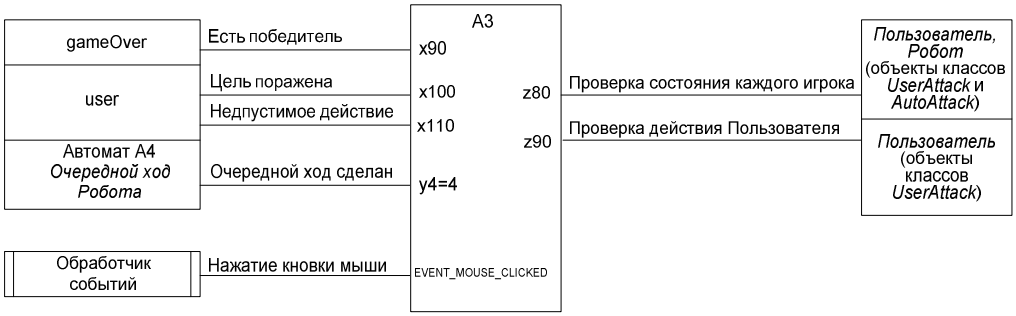

Рис. 13. Схема связей автомата *Управление игрой А3*

9.3.3. Граф переходов

На рис. 14 представлен граф переходов этого автомата.

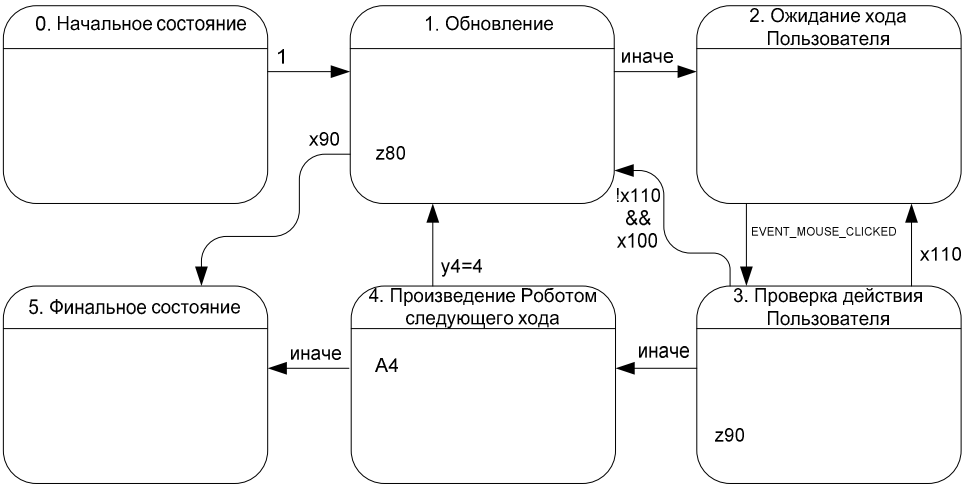

Рис. 14. Граф переходов автомата *Управление игрой А3*

10. Класс *Атака Робота* (*RobotAttack*)

10.1. Словесное описание

Класс *Атака Робота* реализует автомат, обеспечивающий выбор очередного хода *Робота*. Этот класс не принимает никаких внешних воздействий и использует только локальные методы, не воздействуя ни на какие другие объекты, кроме объекта класса *Робот (AutoAttack)*. Структурная схема класса представлена на рис. 15.

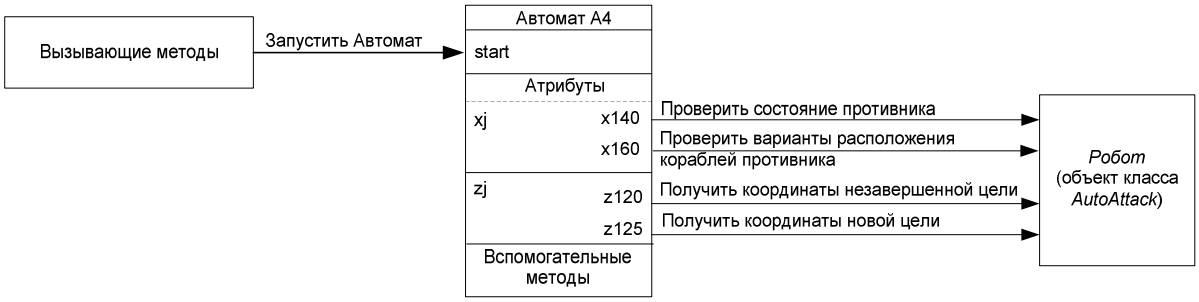

Рис. 15. Структурная схема класса *А4*

10.2. Автомат *Очередной ход Робота* (*А4*)

10.2.1. Словесное описание

Автомат выполнения очередного хода обеспечивает выбор поля для удара. При этом производится завершение незаконченной атаки. В случае поражения цели производится внеочередной выстрел. Автомат вызывается из автомата управления игрой.

10.2.2. Схема связей

Схема связей данного автомата представлена на рис. 16.

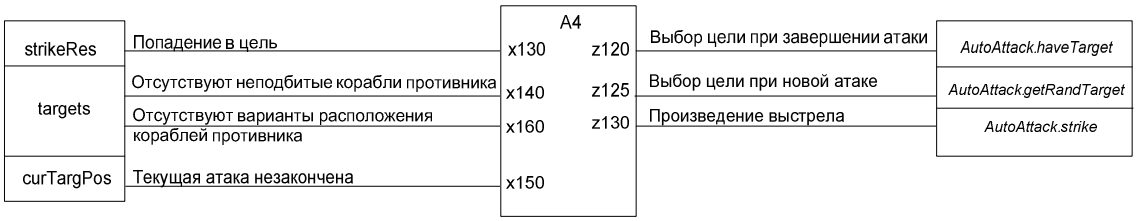

Рис. 16. Схема связей автомата *Очередной ход Робота A4*

10.2.3. Граф переходов

Схема граф переходов данного автомата представлен на рис. 17.

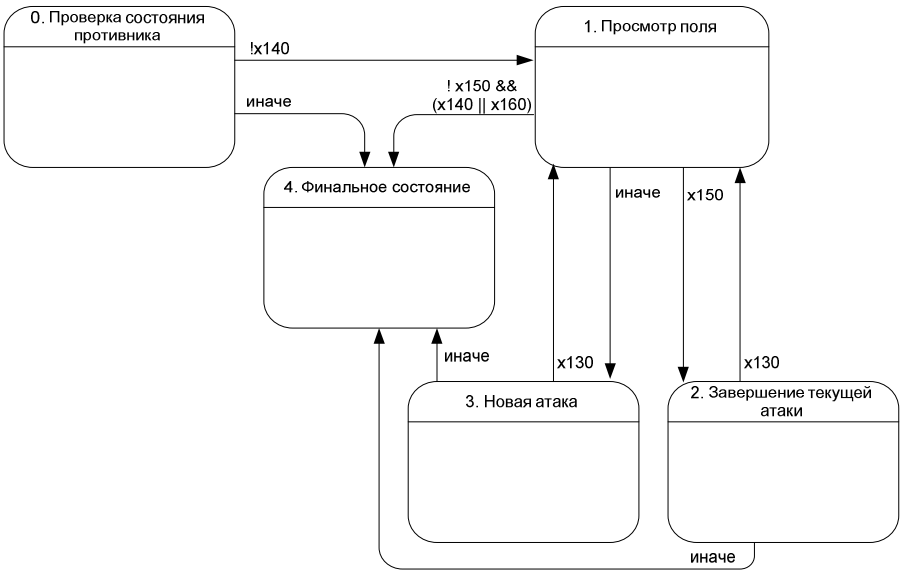

Рис. 17. Граф переходов автомата *Очередной ход Робота A4*

11. Заключение

Использование автоматного подхода значительно упростило построение логики программы, отладку и реализацию разработынных алгоритмов. Представление логики в виде графов переходов весьма наглядно и удобно для формального изоморфного перехода к тексту программ.

12. Источники

1. http://is.ifmo.ru
2. http://www.softcraft.ru
3. *Шалыто А. А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.

Приложение 1. Пример протоколирования

Фрагмент протокола для части первого этапа игры – “Расстановка кораблей Пользователя”.

```
Для объекта A0
A0.0 ("Контроллер игры"."Инициализация") {
   >>>>>>> Расстановка кораблей Пользователя <<<<<<<
   * z0: Инициализация атаки
   A0->1 ("Контроллер игры"->"Расстановка кораблей Пользователем")
} A0.1 ("Контроллер игры"."Расстановка кораблей Пользователем")

Для объекта A0
A0.1 ("Контроллер игры"."Расстановка кораблей Пользователем") {
   Для объекта A1
   A1.0 ("Расстановка кораблей Пользователем"."Начальное состояние") {
      * z7: Инициализация
      * z10: Выбор очередного корабля
      A1->1 ("Расстановка кораблей Пользователем"->"Выбор типа корабля")
   } A1.1 ("Расстановка кораблей Пользователем"."Выбор типа корабля")

   Для объекта A1
   A1.1 ("Расстановка кораблей Пользователем"."Выбор типа корабля") {
      x20: Очередной корабль для установки определен? - ДА
      A1->2 ("Расстановка кораблей Пользователем"->"Ожидание выбора позиции")
   } A1.2 ("Расстановка кораблей Пользователем"."Ожидание выбора позиции")

   Для объекта A1
   A1.2 ("Расстановка кораблей Пользователем"."Ожидание выбора позиции") {
      # Автомат "A1" обработал событие: e0
      * z20: Обработка выбора позиции
      A1->4 ("Расстановка кораблей Пользователем"->"Ожидание выбора направления")
   } A1.4 ("Расстановка кораблей Пользователем"."Ожидание выбора направления")

   Для объекта A1
   A1.4 ("Расстановка кораблей Пользователем"."Ожидание выбора направления") {
      x10: Действие Пользователя некорректно? - НЕТ
      A1->4 ("Расстановка кораблей Пользователем"->"Ожидание выбора направления")
   } A1.4 ("Расстановка кораблей Пользователем"."Ожидание выбора направления")

   Для объекта A1
   A1.4 ("Расстановка кораблей Пользователем"."Ожидание выбора направления") {
      # Автомат "A1" обработал событие: e0
      * z25: Обработка выбора направления
      A1->2 ("Расстановка кораблей Пользователем"->"Ожидание выбора позиции")
   } A1.2 ("Расстановка кораблей Пользователем"."Ожидание выбора позиции")
```

Для объекта A1
A1.2 ("Расстановка кораблей Пользователем"."Ожидание выбора позиции") {
x10: Действие Пользователя некорректно? - НЕТ
* z30: Установка корабля
A1->2 ("Расстановка кораблей Пользователем"->"Ожидание выбора позиции")
} A1.2 ("Расстановка кораблей Пользователем"."Ожидание выбора позиции")

Для объекта A1
A1.2 ("Расстановка кораблей Пользователем"."Ожидание выбора позиции") {
x40: Корабль установлен? - ДА
A1->2 ("Расстановка кораблей Пользователем"->"Ожидание выбора позиции")
} A1.2 ("Расстановка кораблей Пользователем"."Ожидание выбора позиции")

Для объекта A1
A1.2 ("Расстановка кораблей Пользователем"."Ожидание выбора позиции") {
* z7: Инициализация
* z10: Выбор очередного корабля
A1->2 ("Расстановка кораблей Пользователем"->"Ожидание выбора позиции")
} A1.2 ("Расстановка кораблей Пользователем"."Ожидание выбора позиции")

Приложение 2. Исходные тексты программы

```
////////////////////////////////////////////////////////////////////////////////
//                          Класс A0                                          //
////////////////////////////////////////////////////////////////////////////////

package aut;

import aut.autbase.Automaton;
import event.EnumEvent;
import event.EventStorage;
import field.Field;
import field.FieldModel;
import logics.AutoAllocation;
import logics.UserAttack;
import logs.LoggingModel;


/**
 * Автоматный класс, представляющий функциональность управления игрой.
 * Следующие автоматы: Польхователь (User), Робот (Robot),
 * Размещение Кораблей (Ships Allocation), Управление сражением
 * (BattleMain Manager) - вложены в данный автомат и вызываются их него.
 */
public final class A0 extends Automaton {
    /**
     * Состояния автомата
     */
    public final static class State extends Automaton.State {
        private State(String name) {
            super(name);
        }
    }

    public final static State INITIAL_STATE = new State("0");
    public final static State USER_SHIPS_ALLOC = new State("1");
    public final static State COMP_SHIPS_ALLOC = new State("2");
    public final static State BATTLE = new State("3");
    public final static State FINAL_STATE = new State("4");
```

```
final static EnumEvent eTypes[] = {EnumEvent.EVENT_GAME_RESET};

private final A1 a1;
private final A2 a2;
private final A3 a3;

private A0ControlledObjects controlledObjects;

/**
 * Конструктор автомата управления игрой.
 */
public A0(LoggingModel log, A1 a1, A2 a2, A3 a3,
          A0ControlledObjects controlledObjects,
          EventStorage eventStorage) {
    super(INITIAL_STATE, log, eventStorage);
    this.a1 = a1;
    this.a2 = a2;
    this.a3 = a3;
    this.controlledObjects = controlledObjects;

    Z0 = new Action("z0", log) {
        public void doIt() {
            initAllocation();
        }
    };

    Z2 = new Action("z5", log) {
        public void doIt() {
            initBattle();
        }
    };
}

protected final Automaton.State nextState() {
    final Automaton.State y0 = getState();
    Automaton.State resultState = getState();


    if (INITIAL_STATE == y0) {
        log.logSection("Расстановка фигур Пользователя");
        Z0.perform();
        resultState = USER_SHIPS_ALLOC;
    } else if (USER_SHIPS_ALLOC == y0) {
        a1.start(A1.INITIAL_STATE, A1.FINAL_STATE);
        if (!a1.succeeded()) {
            resultState = INITIAL_STATE;
        } else {
            log.logSection("Расстановка  фигур Робота");
            resultState = COMP_SHIPS_ALLOC;
        }
    } else if (COMP_SHIPS_ALLOC == y0) {
        a2.start(A2.INITIAL_STATE, A2.FINAL_STATE);
        if(!a2.succeeded()) {
            resultState = INITIAL_STATE;
        } else {
            log.logSection("Сражение");
            resultState = BATTLE;
        }
    } else if (BATTLE == y0) {
        a3.start(A3.INITIAL_STATE, A3.FINAL_STATE);
        if (!a3.succeeded()) {
            resultState = INITIAL_STATE;
        } else {
            log.logSection("Конец игры");
            resultState = FINAL_STATE;
        }
    } else if (FINAL_STATE == y0) {
        final EnumEvent event = waitForEvents(eTypes);
        if (event != null) {
            resultState = INITIAL_STATE;
            eventStorage.clean(); // clean event storage
        }
    }

    if (COMP_SHIPS_ALLOC == resultState) {
        Z2.perform();
    }
    return resultState;
}
```

```
    /* Производит инициализацию перед расстановкой фигур.*/
    private void initAllocation() {
        final FieldModel robotField = new FieldModel(FieldModel.ROBOT_FLEET_SIGN, false);
        /* Используется дополнительное пустое поле, которое будет
        видно в процессе расстановки кораблей роботом*/
        final FieldModel emptyField = new FieldModel(FieldModel.ROBOT_FLEET_SIGN, false);

        final FieldModel userField = new FieldModel(FieldModel.USER_FLEET_SIGN, true);

        a1.initiateLogics(userField);
        a2.initiateLogics(robotField);

        /** Очищаем хранилище сообщений перед началом новой игры */
        getEventStorage().clean();

        userField.setObserver(controlledObjects.getUserFieldView());
        emptyField.setObserver(controlledObjects.getRobotFieldView());

        // don't allow user to listen to the mouse
        controlledObjects.setUserFieldListener(getEventStorage());
        // make robot field listen to the mouse events
        controlledObjects.setRobotFieldListener(null);
    }

    /** Инициализирует автоматы a3 и a4 после расстановки кораблей
     * роботом и пользователем */
    private void initBattle() {
        /** Очищаем хранилище перед началом новой игровой сессии */
        getEventStorage().clean();

        final FieldModel userField = userFieldView.getModel();
        final FieldModel robotField = ((AutoAllocation) a2.getLogics()).getFieldModel();

        final UserAttack userAttack = new UserAttack(robotField);

        a3.initiateLogics(userField);
        a3.setUser(userAttack);

        userAttack.getFieldModel().setObserver(
                controlledObjects.getRobotFieldView());
        a3.setUserFieldView(controlledObjects.getUserFieldView());

        controlledObjects.setUserFieldListener(null);
        controlledObjects.setRobotFieldListener(getEventStorage());
    }
}

////////////////////////////////////////////////////////////////////////////
//                          Класс A1                                      //
////////////////////////////////////////////////////////////////////////////

package aut;

import aut.autbase.Automaton;
import event.EnumEvent;
import event.EventStorage;
import event.MouseEvent;
import field.FieldModel;
import logics.UserAllocation;
import logics.structs.OrientedPoint;
import logics.structs.Point2Dim;
import logics.structs.Ship;
import logs.LoggingModel;

/**
 * Автоматный класс <code>A1</code> представляет собой функциональность автомата
 * расстановки фигур Пользователя. Данный автомат имеет вложенных автоматов.
 */
public final class A1 extends Automaton {
    /**
     * <code>State</code> является перечислением состояний автомата.
     */
    public final static class State extends Automaton.State {
        private State(String name) {
            super(name);
        }
    }

    /**
```

```
 * <code>Variable</code> перечисление входных переменных автомата.
 */
public final static class Variable extends Automaton.Variable {
    private Variable(String id) {
        super(id);
    }
}

public final static State INITIAL_STATE = new State("0");
public final static State CHOOSING_SHIP_TYPE = new State("1");
public final static State AWAITING_POS = new State("2");
public final static State CHECK_POS = new State("3");
public final static State AWAITING_DIR = new State("4");
public final static State CHECK_DIR = new State("5");
public final static State SHIP_SETUP = new State("6");
public final static State FINAL_STATE = new State("7");

public static final Variable X10 = new Variable("x10");
public static final Variable X15 = new Variable("x15");
public static final Variable X20 = new Variable("x20");
public static final Variable X40 = new Variable("x40");

public final Action Z7;
public final Action Z10;
public final Action Z20;
public final Action Z25;
public final Action Z30;

private UserAllocation logics;
private int curAction;
private Ship curFigure;
private OrientedPoint curPos;
private boolean unallocAbsence;

public A1(final LoggingModel logging, EventStorage eventStorage) {
    super(INITIAL_STATE, logging, eventStorage);
    Z7 = new Action("z7", logging) {
        public void doIt() {
            unallocAbsence = false;
        }
    };

    Z10 = new Action("z10", logging) {
        public void doIt() {
            curFigure = logics.getUnalloc();
            unallocAbsence = (curFigure == null) ? true : false;
            curPos = null;
        }
    };
    Z20 = new Action("z20", logging) {
        public void doIt() {
            final FieldModel fieldVal = logics.getFieldModel();
            final int[][][] auxField = logics.getAuxField();


            if ((curPos.getY() == -1) || (curPos.getX() == -1)
                    || (!logics.posFit(curPos, curFigure.getLength()))) {
                // Действие Пользователя некорректно.
                curAction = -1;
                // поскольку действие некорректное,
                // пусть координаты точки "тычат в космос".
                curPos.setLocation(-1, -1);
            } else {

                // пользователь не "облажался".
                curAction = 1;

                fieldVal.setAt(
                        curPos.getY(), curPos.getX(),
                        FieldModel.CSTATE_FILLED);

                // Покажем подсказку Ползоветлю, куда он может ставить свои
                // корабли.
                // Для начала, покажем горизонтальную подсказку.
                if (auxField[0][curPos.getY()][curPos.getX()]
                        >= curFigure.getLength()) {
                    for (int c = curPos.getX(); c
                            < curPos.getX() + curFigure.getLength(); c++)
                        fieldVal.setAt(
                                curPos.getY(), c,
```

```
                                FieldModel.CSTATE_HINT_ALLOC);
                    }

                    // ... а теперь вертикальную.
                    if (auxField[1][curPos.getY()][curPos.getX()]
                            >= curFigure.getLength()) {
                        for (int r = curPos.getY(); r
                                < curPos.getY() + curFigure.getLength(); r++)
                            fieldVal.setAt(
                                    r, curPos.getX(),
                                    FieldModel.CSTATE_HINT_ALLOC);
                    }
                }
            }
        };
        Z25 = new Action("z25", logging) {
            public void doIt() {
                curAction = (!logics.directFit(curPos, curFigure.getLength()))
                        ? -1 : 1;
            }
        };

        Z30 = new Action("z30", logging) {
            public void doIt() {

                curFigure.setPosition(curPos);
                final FieldModel fieldVal = logics.getFieldModel();

                for (int row = 1; row <= logics.horDim(); row++)
                    for (int col = 1; col <= logics.verDim(); col++)
                        if (fieldVal.getAt(row, col)
                                == FieldModel.CSTATE_HINT_ALLOC)
                            fieldVal.setAt(row, col, FieldModel.CSTATE_EMPTY);

                logics.putShip(curFigure);
            }
        };
    }

    /**
     * Инициализирует контроль расстановки кораблей пользователем.
     * <code>FieldModel</code> поле для расстановки кораблей.
     */
    public final void initiateLogics(final FieldModel field) {
        logics = new UserAllocation(field);
    }

    protected final Automaton.State nextState() {
        final Automaton.State y1 = getState();
        Automaton.State resultState = getState();
        EnumEvent eTypes[] = {EnumEvent.EVENT_GAME_RESET,
                              EnumEvent.EVENT_MOUSE_CLICKED};

        if (y1 == INITIAL_STATE) {
            Z7.perform();
            resultState = CHOOSING_SHIP_TYPE;
        } else if (y1 == CHOOSING_SHIP_TYPE) {
            if (x20())
                resultState = AWAITING_POS;
            else if (x15()) {
                resultState = FINAL_STATE;
            }
        } else if (y1 == AWAITING_POS) {
            final EnumEvent event = waitForEvents(eTypes);
            if ((event != null) && event.equals(EnumEvent.EVENT_MOUSE_CLICKED)) {
                eventToPos(event);
                resultState = CHECK_POS;
            } else if ((event != null)
                    && event.equals(EnumEvent.EVENT_GAME_RESET)) {
                setFailed();
                resultState = awaitingState;
            }
        } else if (y1 == CHECK_POS) {
            if (x10())
                resultState = AWAITING_POS;
            else
                resultState = AWAITING_DIR;
        } else if (y1 == AWAITING_DIR) {
            final EnumEvent event = waitForEvents(eTypes);
            if ((event != null) && event.equals(EnumEvent.EVENT_MOUSE_CLICKED)) {
```

```
            eventToDir(event);
            resultState = CHECK_DIR;
        } else if ((event != null)
                && event.equals(EnumEvent.EVENT_GAME_RESET)) {
            setFailed();
            resultState = awaitingState;
        }
    } else if (y1 == CHECK_DIR) {
        if (x10())
            resultState = AWAITING_DIR;
        else {
            resultState = SHIP_SETUP;
        }
    } else if (y1 == SHIP_SETUP) {
        if (x40())
            resultState = INITIAL_STATE;
        else
            resultState = FINAL_STATE;
    }

    if (CHOOSING_SHIP_TYPE == resultState)
        Z10.perform();
    else if (CHECK_POS == resultState)
        Z20.perform();
    else if (CHECK_DIR == resultState)
        Z25.perform();
    else if (SHIP_SETUP == resultState)
        Z30.perform();
    return resultState;
}

private boolean x40() {
    final boolean result = (
            (curFigure.getY() > 0) && (curFigure.getX() > 0)
            );
    log.logVariable(X40, result);
    return result;
}

private boolean x10() {
    final boolean result = (curAction == -1) ? true : false;
    log.logVariable(X10, result);
    return result;
}

private boolean x20() {
    final boolean result = (
            (curFigure != null) && (curFigure.getLength() > 0)
            );
    log.logVariable(X20, result);
    return result;
}

private boolean x15() {
    final boolean result = unallocAbsence;
    log.logVariable(X15, result);
    return result;
}

private boolean eventToPos(final EnumEvent event) {
    log.logEvent(this, event);
    if (event == null
            || (
            (event != null) && event.equals(EnumEvent.EVENT_GAME_RESET)
            ))
        return false;
    curPos = new OrientedPoint(((MouseEvent) event).getPoint(), 0);
    return true;
}

private boolean eventToDir(final EnumEvent event) {
    log.logEvent(this, event);
    if (event == null
            || (
            (event != null) && event.equals(EnumEvent.EVENT_GAME_RESET)
            ))
        return false;

    final Point2Dim p = ((MouseEvent) event).getPoint();
    if (Math.abs(p.getY() - curPos.getY())
```

```
                > Math.abs(p.getX() - curPos.getX()))
            curPos.setVertical();
        else
            curPos.setHorisontal();
        return true;
    }
}

////////////////////////////////////////////////////////////////////////////////
//                          Класс A2                                          //
////////////////////////////////////////////////////////////////////////////////

package aut;

import aut.autbase.Automaton;
import event.EventStorage;
import field.FieldModel;
import logics.AutoAllocation;
import logics.structs.OrientedPoint;
import logics.structs.Ship;
import logs.LoggingModel;

/**
 * <code>A2</code> представляет собой автоматный класс автоматической
 * расстановки кораблей Роботом. Не имеет вложенных автоматов.
 */
public final class A2 extends Automaton {

    /**
     * <code>State</code> перечисление состояний автомата.
     */
    public final static class State extends Automaton.State {
        private State(String name) {
            super(name);
        }
    }

    /**
     * <code>Variable</code> перечисление входных переменных автомата.
     */
    public final static class Variable extends Automaton.Variable {
        private Variable(String id) {
            super(id);
        }
    }

    private AutoAllocation logics;
    private Ship curFigure;

    public final static State INITIAL_STATE = new State("0");
    public final static State CHOOSING_SHIP_TYPE = new State("1");
    public final static State CHECKING_FIELD = new State("2");
    public final static State SHIP_SETUP = new State("3");
    public final static State FINAL_STATE = new State("4");

    public static final Variable X50 = new Variable("x50");
    public static final Variable X60 = new Variable("x60");
    public static final Variable X70 = new Variable("x70");

    public final Action Z45;
    public final Action Z50;
    public final Action Z60;

    public final void initiateLogics(final FieldModel field) {
        logics = new AutoAllocation(field);
    }

    public A2(final LoggingModel logging, EventStorage storage) {
        super(INITIAL_STATE, logging, storage);
        Z45 = new Action("z45", logging) {
            public void doIt() {
                logics.getFieldModel().fillWith(FieldModel.CSTATE_EMPTY);
            }
        };
        Z50 = new Action("z50", logging) {
            public void doIt() {
                final OrientedPoint pos = logics.getAllocation(
                        curFigure.getLength());
                curFigure.setPosition(pos);
                logics.putShip(curFigure);
```

```
            }
        };
        Z60 = new Action("z60", logging) {
            public void doIt() {
                curFigure = logics.getUnalloc();
            }
        };
    }

    protected final Automaton.State nextState() {
        final Automaton.State y2 = getState();
        A2.State resultState = FINAL_STATE;

        if (y2 == INITIAL_STATE) {
            Z45.perform();
            resultState = CHOOSING_SHIP_TYPE;
        } else if (y2 == CHOOSING_SHIP_TYPE) {
            if (x70())
                resultState = CHECKING_FIELD;
            else {
                resultState = FINAL_STATE;
            }
        } else if (y2 == CHECKING_FIELD) {
            if (x50()) {
                resultState = SHIP_SETUP;
            } else {
                resultState = FINAL_STATE;
            }
        } else if (y2 == SHIP_SETUP) {
            if (x60())
                resultState = CHOOSING_SHIP_TYPE;
            else
                resultState = FINAL_STATE;
        }


        if (CHOOSING_SHIP_TYPE == resultState)
            Z60.perform();
        else if (SHIP_SETUP == resultState)
            Z50.perform();

        return resultState;
    }

    private boolean x60() {
        final boolean result = (curFigure != null) && curFigure.isAllocated();
        log.logVariable(X60, result);
        return result;
    }

    private boolean x50() {
        final boolean result = logics.checkAlloc(curFigure.getLength());
        log.logVariable(X50, result);
        return result;
    }

    private boolean x70() {
        final boolean result = (curFigure != null)
                && (curFigure.getLength() > 0);
        log.logVariable(X70, result);
        return result;
    }

    final Object getLogics() {
        return logics;
    }
}

////////////////////////////////////////////////////////////////////////////////
//                          Класс A3                                          //
////////////////////////////////////////////////////////////////////////////////

package aut;

import aut.autbase.Automaton;
import event.EnumEvent;
import event.EventStorage;
import event.MouseEvent;
import logics.UserAttack;
import logics.structs.Point2Dim;
```

```
import logs.LoggingModel;

/**
 * <code>A3</code> автоматный класс контроллера игры. Имеет вложенный автомат
 * Автоматическая Атака Роботом.
 */
public final class A3 extends Automaton {

    /**
     * <code>State</code> перечисление состояний автомата.
     */
    public final static class State extends Automaton.State {
        private State(String name) {
            super(name);
        }
    }

    /**
     * <code>Variable</code> перечисление входных переменных автомата.
     */
    public final static class Variable extends Automaton.Variable {
        private Variable(String id) {
            super(id);
        }
    }

    private UserAttack user;
    private final A4 a4;
    private boolean gameOver;

    public final static State INITIAL_STATE = new State("0");
    public final static State UPDATING = new State("1");
    public final static State AWATING_USER_MOVE = new State("2");
    public final static State CHECKING_USER_ACTION = new State("3");
    public final static State ROBOT_MOVE = new State("4");
    public final static State FINAL_STATE = new State("5");

    public static final Variable X90 = new Variable("x90");
    public static final Variable X100 = new Variable("x100");
    public static final Variable X110 = new Variable("x110");

    public final Action Z80;
    public final Action Z90;

    public A3(final LoggingModel logging, final A4 a4, EventStorage storage) {
        super(INITIAL_STATE, logging, storage);
        this.a4 = a4;
        Z80 = new Action("z80", logging) {
            public void doIt() {
                if (!a4.getLogics().enemyIsAlive()) {
                    log.logSection("Робот успешно выиграл сражение");
                } else if (!user.enemyIsAlive()) {
                    log.logSection("Пользователь успешно выиграл сражение");
                }
                gameOver = (!a4.getLogics().enemyIsAlive())
                        || (!user.enemyIsAlive());
                if (gameOver) {
                    a4.getLogics().getFieldModel().setDisabled();
                    user.getFieldModel().setDisabled();
                    user.showRobotFleet();
                }
            }
        };
        Z90 = new Action("z90", logging) {
            public void doIt() {
                final Point2Dim curTarget = user.getCurTarget();
                if (curTarget.getX() != -1 && curTarget.getY() != -1)
                    user.strike(curTarget.getY(), curTarget.getX());
            }
        };
    }

    /**
     * Устанавливает контроль атаки Польхователем.
     */
    public final void setUser(final UserAttack user) {
        this.user = user;
    }

    protected final Automaton.State nextState() {
```

```
    final Automaton.State y3 = getState();
    Automaton.State resultState = getState();
    EnumEvent eTypes[] = {EnumEvent.EVENT_GAME_RESET,
                          EnumEvent.EVENT_MOUSE_CLICKED};

    if (y3 == INITIAL_STATE) {
        resultState = UPDATING;
    } else if (y3 == UPDATING) {
        if (x90())
            resultState = FINAL_STATE;
        else
            resultState = AWATING_USER_MOVE;
    } else if (y3 == AWATING_USER_MOVE) {
        final EnumEvent event = waitForEvents(eTypes);
        if (eventToPos(event)) {
            resultState = CHECKING_USER_ACTION;
        } else if ((event != null)
                && event.equals(EnumEvent.EVENT_GAME_RESET)) {
            setFailed();
            resultState = awaitingState;
        }
    } else if (y3 == CHECKING_USER_ACTION) {
        if (x110())
            resultState = AWATING_USER_MOVE;
        else {
            if (x100())
                resultState = UPDATING;
            else
                resultState = ROBOT_MOVE;
        }
    } else if (y3 == ROBOT_MOVE) {
        a4.start(A4.CHECKING_ENEMY, A4.FINAL_STATE);
        resultState = UPDATING;
    }

    if (UPDATING == resultState)
        Z80.perform();
    else if (CHECKING_USER_ACTION == resultState)
        Z90.perform();
    return resultState;
}

private boolean x100() {
    final boolean result = user.getStrikeResult();
    log.logVariable(X100, result);
    return result;
}

private boolean x110() {
    final Point2Dim curTarget = user.getCurTarget();

    final boolean result = (curTarget.getY() < 1)
            || (curTarget.getY() > user.verDim())
            || (curTarget.getX() < 1)
            || (curTarget.getX() > user.horDim());
    log.logVariable(X110, result);
    return result;
}

private boolean x90() {
    log.logVariable(X90, gameOver);
    return gameOver;
}

private boolean eventToPos(final EnumEvent event) {
    log.logEvent(this, event);
    if (event == null ||
            ((event != null) && event.equals(EnumEvent.EVENT_GAME_RESET)) ||
            ((MouseEvent) event).getPoint() == null)
        return false;
    user.setCurTarget(((MouseEvent) event).getPoint());
    return true;
}

public void initiateLogics(FieldModel userField) {
    a4.initiateLogics(userField);
}

public void setUserFieldView(Field userFieldView) {
    a4.getLogics().getFieldModel().setObserver(userFieldView);
```

```
    }
}

////////////////////////////////////////////////////////////////////////////////
//                          Класс A4                                          //
////////////////////////////////////////////////////////////////////////////////

package aut;

import aut.autbase.Automaton;
import event.EventStorage;
import field.FieldModel;
import logics.AutoAttack;
import logics.structs.Point2Dim;
import logs.LoggingModel;

/**
 * <code>A4</code> автоматный объект представляющий функциональность автомата
 * Атаки Роботом. Не имеет вложенных автоматов.
 */
public final class A4 extends Automaton {
    /**
     * <code>State</code> перечисление состояний автомата.
     */
    public final static class State extends Automaton.State {
        private State(String name) {
            super(name);
        }
    }

    /**
     * <code>Variable</code> перечисление входных переменных автомата.
     */
    public final static class Variable extends Automaton.Variable {
        private Variable(String id) {
            super(id);
        }
    }

    private AutoAttack logics;
    private boolean strikeRes;
    private Point2Dim curTargPos;

    public final static State CHECKING_ENEMY = new State("0");
    public final static State CHECKING_FIELD = new State("1");
    public final static State COMPLETING_ATTACK = new State("2");
    public final static State NEW_ATTACK = new State("3");
    public final static State FINAL_STATE = new State("4");

    public static final Variable X130 = new Variable("x130");
    public static final Variable X140 = new Variable("x140");
    public static final Variable X150 = new Variable("x150");
    public static final Variable X160 = new Variable("x160");

    public final Action Z120;
    public final Action Z125;
    public final Action Z130;

    public final void initiateLogics(final FieldModel field) {
        logics = new AutoAttack(field);
    }

    public A4(final LoggingModel logging, EventStorage storage) {
        super(CHECKING_ENEMY, logging, storage);
        Z120 = new Action("z120", logging) {
            public void doIt() {
                curTargPos = logics.haveTarget();
            }
        };
        Z125 = new Action("z125", logging) {
            public void doIt() {
                curTargPos = logics.getRandTarget();
            }
        };

        Z130 = new Action("z130", logging) {
            public void doIt() {
                if (curTargPos != null)
                    strikeRes
                            = logics.strike(
```

```
                                curTargPos.getY(), curTargPos.getX()
                            );
                }
            };
        }

        protected final Automaton.State nextState() {
            Automaton.State resultState = getState();
            final Automaton.State y4 = getState();

            if (y4 == CHECKING_ENEMY) {
                if (!x140()) {
                    resultState = CHECKING_FIELD;
                } else
                    resultState = FINAL_STATE;
            } else if (y4 == CHECKING_FIELD) {
                if (x150()) {
                    resultState = COMPLETING_ATTACK;
                } else {
                    if (x140() || x160()) {
                        resultState = FINAL_STATE;
                    } else {
                        resultState = NEW_ATTACK;
                    }
                }
            } else if (y4 == COMPLETING_ATTACK) {
                if (x130())
                    resultState = CHECKING_FIELD;
                else
                    resultState = FINAL_STATE;
            } else if (y4 == NEW_ATTACK) {
                if (x130())
                    resultState = CHECKING_FIELD;
                else
                    resultState = FINAL_STATE;
            }

            if (CHECKING_FIELD == resultState)
                Z120.perform();
            else if (COMPLETING_ATTACK == resultState)
                Z130.perform();
            else if (NEW_ATTACK == resultState) {
                Z125.perform();
                Z130.perform();
            }

            return resultState;
        }

        private boolean x160() {
            final boolean result = !logics.checkAlloc(logics.getMaxTarget());
            log.logVariable(X160, result);
            return result;
        }

        private boolean x150() {
            final boolean result = (curTargPos != null);
            log.logVariable(X150, result);
            return result;
        }

        private boolean x140() {
            final boolean result = logics.getMaxTarget() == 0;
            log.logVariable(X140, result);
            return result;
        }

        private boolean x130() {
            log.logVariable(X130, strikeRes);
            return strikeRes;
        }

        final AutoAttack getLogics() {
            return logics;
        }
    }
```

```
////////////////////////////////////////////////////////////////////////////////
//                          Класс Automaton                                   //
////////////////////////////////////////////////////////////////////////////////

package aut.autbase;

import event.EnumEvent;
import event.EventStorage;
import logs.LoggingModel;

/**
 * Базоввй автоматный класс
 */
public abstract class Automaton extends Enumerable implements Runnable {
    /**
     * <code>State</code> перечисление состояний автомата.
     */
    public static class State extends Enumerable {
        protected State(String name) {
            super(name);
        }
    }

    /**
     * <code>Variable</code> перечисление входных переменных автомата.
     */
    public static class Variable extends Enumerable {
        protected Variable(String name) {
            super(name);
        }
    }

    /**
     * <code>Action</code> перечисление выходных воздействий.
     */
    public static abstract class Action extends Enumerable {
        private final LoggingModel log;

        public Action(String name, LoggingModel log) {
            super(name);
            this.log = log;
        }

        abstract public void doIt();

        public void perform() {
            log.logAction(this);
            doIt();
        }
    }

    private final Object stateMonitor = new Object();

    private Thread thread;
    private State state;
    protected final LoggingModel log;
    protected State awaitingState;
    private boolean succeeded;


    public boolean succeeded() {
        return succeeded;
    }

    public void setFailed() {
        succeeded = false;
    }

    public EventStorage getEventStorage() {
        return eventStorage;
    }

    protected EventStorage eventStorage;
```

```
/**
 * Автоматы предполагаются наделенными знанием своих идентифиматоров, также
 * идентификаторов своих входных переменных и выходных воздействий и нет
 * необходимости "протаскивать" их через иераркию конструкторов автоматных
 * объектов
 */
public Automaton(
        State initialState, LoggingModel log, EventStorage eventStorage
        ) {
    super(null);
    this.log = log;
    state = initialState;
    this.eventStorage = eventStorage;
}

public final State getState() {
    return state;
}

protected EnumEvent waitForEvents(EnumEvent[] events) {
    return eventStorage.waitForEvents(events);
}

private void setState(State state) {
    if (state == this.state)
        return;
    this.state = state;
    log.logAutSwitching(this);
}

protected abstract State nextState();

/**
 *  Производит (один) шаг автомата
 */
private void stepForward() {
    log.logObject(getAutID());
    log.logAutRunning(this);
    setState(nextState());
    log.logAutTermination(this);
}

public final String getAutID() {
    String[] arr = getDomain().getName().split("\\.");
    return (arr.length != 0) ? arr[arr.length - 1] : "";
}

public final void start() {
    thread = new Thread(this);
    thread.start();
}

public final void start(State initialState, State finalState) {
    succeeded = true;
    this.state = initialState;
    awaitingState = finalState;
    thread = new Thread(this);
    synchronized (stateMonitor) {
        thread.start();
        try {
            stateMonitor.wait();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

/**
 * Реализация Thread.run() метода
 */
public final void run() {
    while (thread != null && !thread.isInterrupted()) {
        stepForward();
        if (state == awaitingState) {
            thread.interrupt();
            synchronized (stateMonitor) {
                stateMonitor.notify();
            }
        }
    }
```

```
    }
}
////////////////////////////////////////////////////////////////////////////////
//                          Класс Enumerable                                  //
////////////////////////////////////////////////////////////////////////////////

package aut.autbase;

import java.util.*;

/**
 * Класс у которого может быть только конечное, заранее определенное
 * (перечисляемое) количество объектов. Вариант реализации образца
 * объектно-ориентированного проектирования Одиночка (Singleton)
 */
public class Enumerable {
    public final static String PROPERTIES_FILE_NAME = "resources.Texts";
    private final String id;
    private final String descirption;

    protected Enumerable(String id) {
        this.id = id;
        try {
            register(this);
        } catch (IllegalStateException e) {
            e.printStackTrace();
        }
        descirption = getProperty(null);
    }

    public final String getId() {
        return id;
    }

    public String getDescirption() {
        return descirption;
    }

    protected Class getDomain() {
        return getClass();
    }

    /**
     * Возвращает строковое наименование перечисляемого объекта,
     * хранимое в ресурсном файле
     */
    protected String getProperty(String suffix) {
        String[] arr = getDomain().getName().split("\\$");
        String lastClassName = (arr.length != 0) ? arr[arr.length - 1] : "";
        String key;

        try {
            Integer.parseInt(lastClassName);
            lastClassName = new String("");
        } catch (NumberFormatException exception) {
            // do nothing
        }

        if (id == null)
            key = lastClassName + ((null == suffix) ? "" : "." + suffix);
        else if (arr.length != 2) {
            if (lastClassName.compareTo("") != 0)
                key = lastClassName + "." + id
                        + ((null == suffix) ? "" : "." + suffix);
            else
                key = id + ((null == suffix) ? "" : "." + suffix);
        } else {
            if (lastClassName.compareTo("") != 0)
                key = arr[0] + "." + lastClassName + "." + id
                        + ((null == suffix) ? "" : "." + suffix);
            else
                key = arr[0] + "." + id
                        + ((null == suffix) ? "" : "." + suffix);
        }

        try {
            return properties.getString(key);
        } catch (MissingResourceException exc) {
            System.err.println(exc.getMessage());
            return null;
```

```
        }

    }

    private final static Map ENUMERABLES = new HashMap();
    private final static ResourceBundle properties;

    static {
        properties = ResourceBundle.getBundle(PROPERTIES_FILE_NAME);
    }

    /**
     * Производит регистрацию перечисляемых объектов
     */
    private static void register(Enumerable enumerable) throws IllegalStateException{
        synchronized (ENUMERABLES) {
            Class domain = enumerable.getDomain();
            String id = enumerable.getId();
            Set set = (Set) ENUMERABLES.get(domain);
            if (null == set) {
                set = new HashSet();
                ENUMERABLES.put(domain, set);
            }
            if (set.contains(id)) {
                throw new IllegalStateException(
                        "Enumerable " + id + "@" + domain + " already exists"
                );
            } else {
                set.add(id);
            }
        }
    }
}

////////////////////////////////////////////////////////////////////////////////
//                          Класс ButtonEventProvider                         //
////////////////////////////////////////////////////////////////////////////////

package event;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
  * Класс поставщик сообщений от кнопок
 */
public class ButtonEventProvider extends EventProvider
        implements ActionListener {
    public ButtonEventProvider() {
    }

    public void actionPerformed(ActionEvent e) {
        eventHappened(EnumEvent.EVENT_GAME_RESET);
    }
}

////////////////////////////////////////////////////////////////////////////////
//                          Класс EnumEvent                                   //
////////////////////////////////////////////////////////////////////////////////

package event;


/**
 * Класс - перечисление событий. Содержит тип события, содержание (описание)
 * события (например, координаты точки щелчка мыши).
 */
public class EnumEvent {
    public static final EnumEvent EVENT_MOUSE_CLICKED = new EnumEvent("e0");
    public static final EnumEvent EVENT_GAME_RESET = new EnumEvent("e10");

    private final String myName; // Данные для отладки

    protected EnumEvent(String id) {
        myName = id;
    }

    public String toString() {
        return myName;
    }
```

```
    public boolean equals(Object o) {
        if (this == o)
            return true;
        if (!(o instanceof EnumEvent))
            return false;
        final EnumEvent event = (EnumEvent) o;
        if (myName != null
                ? !myName.equals(event.myName) : event.myName != null)
            return false;
        return true;
    }

    public int hashCode() {
        return (myName != null ? myName.hashCode() : 0);
    }
}

////////////////////////////////////////////////////////////////////////////////
//                          Класс EventProvider                               //
////////////////////////////////////////////////////////////////////////////////

package event;


/**
 * Класс, реализующий "поставщика" событий.
 */
public class EventProvider {
    private ProviderListener listener = null;

    /**
     * Устанавливает обработчика;
     * согласно дизайну, провайдер не может иметь более одного обработчика;
     */
    public final void setListener(final ProviderListener listener) {
        this.listener = listener;
    }

    public final void eventHappened(final EnumEvent e) {
        notifyProviderListener(e);
    }

    private void notifyProviderListener(final EnumEvent event) {
        if (listener != null)
            listener.addEvent(event);
    }

    public boolean hasListener() {
        return listener != null;
    }
}

////////////////////////////////////////////////////////////////////////////////
//                          Класс EventStorage                                //
////////////////////////////////////////////////////////////////////////////////

package event;

import java.util.ArrayList;

/**
 * Класс, реализующий очередь необработанных событий
 */
public class EventStorage implements ProviderListener {

    private final ArrayList eventStorage;

    public EventStorage() {
        eventStorage = new ArrayList();
    }
```

```
    /**
     * Каждый автомат, находясь в состоянии, в котором ожидается соответствующее
     * событие, обращается к очереди необработанных событий, и, если в ней есть
     * событие нужного типа, обрабатывает его, удаляя из очереди. Если такого
     * события в очереди нет, то автомат ждет до тех пор, пока оно не произойдет
     * и не поступит в очередь для обработки.
     */
    public final EnumEvent waitForEvents(EnumEvent[] events) {
        synchronized (eventStorage) {
            while (true) {
                for (int i = 0; i < events.length; i++) {
                    EnumEvent event = events[i];
                    int localInd = eventStorage.indexOf(event);
                    if (localInd != -1) {
                        return (EnumEvent) eventStorage.remove(localInd);
                    }
                }

                try {
                    eventStorage.wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }

    /**
     * Добавляет событие в очередь необработанных событий и оповещает
     * ожидающие потоки о факте свершения очередного события.
     */
    public final void addEvent(final EnumEvent event) {
        synchronized (eventStorage) {
            eventStorage.add(event);
            eventStorage.notify();
        }
    }

    /**
     * Очищает очередь необработанных событий.
     */
    public final void clean() {
        synchronized (eventStorage) {
            eventStorage.clear();
        }
    }
}
```

```
////////////////////////////////////////////////////////////////////////////////
//                          Класс MouseEvent                                  //
////////////////////////////////////////////////////////////////////////////////

package event;

import logics.structs.Point2Dim;

/**
 * Событие - "клик" кнопкой мыши.
 */
public class MouseEvent extends EnumEvent {
    private final Point2Dim point;

    public MouseEvent(Point2Dim point) {
        super(EVENT_MOUSE_CLICKED.toString());
        this.point = point;
    }

    public final Point2Dim getPoint() {
        return point;
    }
}
```

```
////////////////////////////////////////////////////////////////////////////////
//                          Класс ProviderListener                            //
////////////////////////////////////////////////////////////////////////////////

package event;

public interface ProviderListener {

    public void addEvent(EnumEvent event);
}
```

```
////////////////////////////////////////////////////////////////////////////////
//                          Класс Field                                       //
////////////////////////////////////////////////////////////////////////////////

package field;

import event.EnumEvent;
import event.EventProvider;
import logics.structs.Point2Dim;

import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionAdapter;

/**
 * Класс, реализующий вид (view) для модели игрового поля (FieldModel).
 */
public final class Field extends JComponent {

    private FieldModel model;
    private final FieldMouseListener mouseListener;
    private final FieldMouseMotionListener mouseMotionListener;
    // Размеры отступов для каждой клетки
    private static final Dimension cellIndent = new Dimension(1, 1);
    // Размеры контрола "Поле"
    private static final Dimension fieldControlSize = new Dimension(180, 225);
    // Высота отступа для надписи на поле
    private static final int textHeight = fieldControlSize.height / 7;

    public Field() {
        mouseListener = new FieldMouseListener();
        this.addMouseListener(mouseListener);
        mouseMotionListener = new FieldMouseMotionListener();
        this.addMouseMotionListener(mouseMotionListener);
        this.setPreferredSize(fieldControlSize); // set the preferred size of the control
    }

    private void cleadFieldFromSightedCells() {
        Dimension fieldSize = model.getFieldSize();
        for (int h = 1; h <= fieldSize.getHeight(); h++)
            for (int w = 1; w <= fieldSize.getWidth(); w++)
                if (model.getAt(h, w) == FieldModel.CSTATE_SIGHTED)
                    model.setAt(h, w, FieldModel.CSTATE_EMPTY);
    }

    public final class FieldMouseMotionListener extends MouseMotionAdapter {
        public void mouseMoved(MouseEvent e) {
            super.mouseMoved(e);
            Point2Dim movedPoint = getPressedCell(new Point2Dim(e.getPoint()));
            if (movedPoint.getY() != -1 && movedPoint.getX() != -1) {
                cleadFieldFromSightedCells();
                if ((model.getAt(movedPoint.getY(), movedPoint.getX()) ==
                        FieldModel.CSTATE_EMPTY) &&
                        (getMouseEventProvider() != null) &&
                        (getMouseEventProvider().hasListener()))
                    model.setAt(
                            movedPoint.getY(), movedPoint.getX(),
                            FieldModel.CSTATE_SIGHTED);
            }
        }
    }
```

```
public final class FieldMouseListener extends MouseAdapter {
    final EventProvider provider;

    public FieldMouseListener() {
        provider = new EventProvider();
    }

    public final EventProvider getProvider() {
        return provider;
    }

    public final void mousePressed(final MouseEvent e) {
        super.mousePressed(e);
        Point2Dim pressedPoint = getPressedCell(
                new Point2Dim(e.getPoint()));
        final EnumEvent event = new event.MouseEvent(pressedPoint);
        provider.eventHappened(event);
    }

    public void mouseExited(MouseEvent e) {
        super.mouseExited(e);
        cleadFieldFromSightedCells();
    }
}

public final EventProvider getMouseEventProvider() {
    return mouseListener.getProvider();
}

private Point2Dim getPressedCell(final Point2Dim pointClicked) {
    final double x = pointClicked.getX();
    final double y = pointClicked.getY() - textHeight;

    final Dimension controlSize = this.getSize();
    final Dimension fieldDims = model.getFieldSize();
    final double cellWidth = controlSize.width / fieldDims.width;
    final double cellHeight = (controlSize.height - textHeight)
            / fieldDims.height;
    final Point2Dim result = new Point2Dim();

    double box = x / (cellWidth);

    if (box - Math.floor(box) < cellWidth / (cellWidth))
        result.setX((int) Math.ceil(box));
    else
        result.setX(-1);


    box = y / (cellHeight);
    if (box - Math.floor(box) < cellHeight / (cellHeight))
        result.setY((int) Math.ceil(box));
    else
        result.setY(-1);
    return result;
}

public final void update(FieldModel fmodel) {
    if (fmodel != null) {
        this.model = fmodel;
        repaint();
    }
}

public final void paint(final Graphics g) {
    super.paint(g);
    if (g == null || model == null)
        return;

    g.setFont(new Font("Serif", Font.PLAIN, 17));
    g.drawString(model.getFieldName(), 0, textHeight * 3 / 4);

    final Dimension controlSize = this.getSize();
    final Dimension fieldDims = model.getFieldSize();
    final Dimension cIndent = cellIndent;
    final int cellWidth = controlSize.width / (fieldDims.width)
            - cIndent.width;
    final int cellHeight = (controlSize.height - textHeight)
            / (fieldDims.height)
            - cIndent.height;
```

```
        for (int row = 1; row <= fieldDims.getHeight(); row++) {
            for (int col = 1; col <= fieldDims.getWidth(); col++) {
                final int curX = (cIndent.width + cellWidth) * (col - 1);
                final int curY = textHeight
                        + (cIndent.height + cellHeight) * (row - 1);

                g.setColor(model.getAt(row, col).getColor());
                g.fillRect(curX, curY, cellWidth, cellHeight);
                drawSight(g, row, col, curX, curY, cellHeight, cellWidth);
            }
        }
    }

    private void drawSight(Graphics g, int row, int col,
                           int curX, int curY, int cellWidth, int cellHeight) {
        if (model.getAt(row, col) == FieldModel.CSTATE_SIGHTED &&
                !model.isDisabled()) {
            g.setColor(Color.white);
            g.drawOval(
                    curX + cellWidth / 4, curY + cellHeight / 4,
                    cellWidth / 2, cellHeight / 2);
            g.drawOval(curX, curY, cellWidth, cellHeight);

            g.drawLine(
                    curX, curY + cellHeight / 2, curX + cellWidth,
                    curY + cellHeight / 2);
            g.drawLine(
                    curX + cellWidth / 2, curY, curX +
                    cellWidth / 2,
                    curY + cellHeight);
        }
    }

    public FieldModel getModel() {
        return model;
    }
}

////////////////////////////////////////////////////////////////////////////////
//                          Класс FieldModel                                  //
////////////////////////////////////////////////////////////////////////////////

package field;


import aut.autbase.Enumerable;

import java.awt.*;

/** Класс, представляющий модель игрового поля. */
public final class FieldModel {
    private boolean isDisabled = false;

    public static final String ROBOT_FLEET_SIGN = "Флот Робота";
    public static final String USER_FLEET_SIGN = "Флот Пользователя";

    public final static CellState CSTATE_EMPTY = new CellState("CSTATE_EMPTY");
    public final static CellState CSTATE_SIGHTED = new CellState(
            "CSTATE_SIGHTED");
    public final static CellState CSTATE_FILLED = new CellState(
            "CSTATE_FILLED");
    public final static CellState CSTATE_ATTACKED_EMPTY = new CellState(
            "CSTATE_ATTACKED_EMPTY");
    public final static CellState CSTATE_HINT_ALLOC = new CellState(
            "CSTATE_HINT_ALLOC");
    public final static CellState CSTATE_HINT_EMPTY = new CellState(
            "CSTATE_HINT_EMPTY");
    public final static CellState CSTATE_DISABLED = new CellState(
            "CSTATE_DISABLED");
    public final static CellState CSTATE_HINT_FILLED = new CellState(
            "CSTATE_HINT_FILLED");

    public final static class CellState extends Enumerable {
        private Color color;
```

```
    /**
     * Помимо прочего, конструктор восстанавливает значения цветов из
     * ресурсного файла. <code>id</code> является идентификатором
     * перечисляемого объекта - строка.
     */
    protected CellState(String id) {
        super(id);
        String strColor = getProperty("Color");
        try {
            color = new Color(Integer.parseInt(strColor, 16));
        } catch (NumberFormatException exc) {
            System.err.println(exc.getMessage());
            color = new Color(200, 200, 200);
        }
    }

    public Color getColor() {
        return color;
    }
}

private CellState[][] cells;
private final Dimension fieldSize;   // Размеры поля
private final String fieldName;      // Надпись над полем при отрисовке
private Field observer;


public FieldModel(final String fieldName, final boolean initEnabled) {
    this(null, fieldName, initEnabled, null);
}

public FieldModel(final FieldModel model, final boolean initEnabled) {
    this(null, model.getFieldName(), initEnabled, null);

}

private FieldModel(final CellState[][] values, final String fieldName,
                   final boolean initEnabled, final Dimension fieldDim) {
    this.fieldSize = (fieldDim != null) ? fieldDim : new Dimension(10, 10);
    this.cells
            = (values != null)
            ? values
            : (
            (initEnabled)
            ? fillVals(CSTATE_EMPTY) : fillVals(CSTATE_DISABLED)
            );
    this.fieldName = fieldName;
}

public final String getFieldName() {
    return fieldName;
}

public void setObserver(Field field) {
    if (field != null) {
        observer = field;
        observer.update(this);
    }
}


public final Dimension getFieldSize() {
    return fieldSize;
}

/** Возвращает состояние ячейки поля */
public final CellState getAt(final int row, final int column) {
    return cells[row][column];
}

/** Устанавливает состояние ячейки поля. */
public final void setAt(final int row, final int column,
                        final CellState value) {
    cells[row][column] = value;
    if (observer != null)
        observer.update(this);
}

public final int horDim() {
    return (int) fieldSize.getWidth();
```

```
    }

    public final int verDim() {
        return (int) fieldSize.getHeight();
    }

    public final void fillWith(final CellState valToFill) {
        cells = fillVals(valToFill);
    }

    private CellState[][] fillVals(final CellState valToFill) {
        final CellState[][] values = new CellState[verDim() + 2][horDim() + 2];
        for (int row = 0; row < horDim() + 2; row++) {
            for (int col = 0; col < verDim() + 2; col++) {
                values[row][col] = valToFill;
            }
        }
        return values;
    }

    public boolean isDisabled() {
        return isDisabled;
    }

    public void setDisabled() {
        isDisabled = true;
        for (int row = 1; row <= horDim(); row++)
            for (int col = 1; col <= verDim(); col++) {
                CellState sellState = getAt(row, col);
                if ((sellState != CSTATE_FILLED) &&
                        (sellState != CSTATE_HINT_FILLED))
                    setAt(row, col, CSTATE_EMPTY);
            }

        if (observer != null)
            observer.update(this);
    }
}
```

Класс Allocation

```
package logics;

import field.FieldModel;
import logics.structs.Ship;

/**
 * Базовый класс для установщиков кораблей
 */
abstract public class Allocation extends BaseProcessor {
    // Количество кораблей на поле
    private final int TOTAL_SHIPS_NUM = 10;
    // Максимальный размер корабля
    private final int MAX_SHIP_SIZE = 4;
    private Ship[] ships;

    public Allocation(final FieldModel fieldModel) {
        super(fieldModel);
        initAllocation();
    }

    private void initAllocation() {
        ships = new Ship[TOTAL_SHIPS_NUM];
        int ind = TOTAL_SHIPS_NUM - 1;
        for (int shipSize = 0; shipSize < MAX_SHIP_SIZE; shipSize++) {
            for (int i = 0; i <= shipSize; i++) {
                ships[ind--] = new Ship(MAX_SHIP_SIZE - shipSize);
            }
        }
    }

    public final Ship getUnalloc() {
        for (int i = TOTAL_SHIPS_NUM - 1; i >= 0; i--) {
            if (!ships[i].isAllocated())
                return ships[i];
        }
        return null;
    }
```

```
}

////////////////////////////////////////////////////////////////////////////////
//                            Класс Attack                                    //
////////////////////////////////////////////////////////////////////////////////

package logics;

import field.FieldModel;

/**
 * Базовый класс для атакующего.
 */
public class Attack extends BaseProcessor {
    int[] targets;
    protected final FieldModel enemyMap;

    public Attack(final FieldModel enemy) {
        super(new FieldModel(enemy, true));
        this.enemyMap = enemy;
        initAttack();
    }

    private void initAttack() {
        targets = new int[4];
    }

    /**
     * Индикатор "жизнеспособности" противника.
     */
    public final boolean enemyIsAlive() {
        for (int t = 0; t < 4; t++) {
            if (targets[t] < 4 - t)
                return true;
        }
        return false;
    }

    /**
     * Возвращает длину последовательности занятых клетов в даном месте.
     */
    private int getTargetLength(final int y, final int x,
                                final FieldModel field) {

        if (field.getAt(y, x) != FieldModel.CSTATE_FILLED) {
            return 0;
        }

        final boolean isLevel;
        int length = 0;

        if ((field.getAt(y + 1, x) == FieldModel.CSTATE_FILLED)
                || (field.getAt(y - 1, x) == FieldModel.CSTATE_FILLED))
            isLevel = false;
        else {
            if ((field.getAt(y, x + 1) == FieldModel.CSTATE_FILLED)
                    || (field.getAt(y, x - 1) == FieldModel.CSTATE_FILLED))
                isLevel = true;
            else
                return 1;
        }

        if (isLevel) // Если цель ориентирована горизонтально
        {
            int col = x;
            while ((field.getAt(y, col - 1) == FieldModel.CSTATE_FILLED)
                    && (col > 1))
                col--;

            while ((field.getAt(y, col) == FieldModel.CSTATE_FILLED)
                    && (col <= horDim)) {
                col++;
                length++;
            }
        } else        // Если цель ориентирована вертикально
        {
            int row = y;
            while ((field.getAt(row - 1, x) == FieldModel.CSTATE_FILLED)
                    && (row > 1))
                row--;
```

```
        while ((field.getAt(row, x) == FieldModel.CSTATE_FILLED)
                && (row <= verDim)) {
            row++;
            length++;
        }
    }
    return length;
}

/**
 * Помечает все соседние с кораблем клетки, как простреленные,
 * чтобы не делать лишних ударов.
 */
private void markTarget(final int y, final int x, final boolean finished) {
    if (fieldModel.getAt(y, x) != FieldModel.CSTATE_FILLED)
        return;
    final boolean isLevel;

    if ((fieldModel.getAt(y + 1, x) == FieldModel.CSTATE_FILLED)
            || (fieldModel.getAt(y - 1, x) == FieldModel.CSTATE_FILLED))
        isLevel = false;
    else {
        if ((fieldModel.getAt(y, x + 1) == FieldModel.CSTATE_FILLED)
                ||
                (fieldModel.getAt(y, x - 1) == FieldModel.CSTATE_FILLED))
            isLevel = true;
        else // В случае единичной длины корабля
        {
            for (int i = -1; i < 2; i++) {
                for (int j = -1; j < 2; j++) {
                    if ((
                            finished && ((i != 0) || (j != 0))
                            || ((!finished) && ((i != 0) && (j != 0)))
                            ))
                        setCellState(
                                y + i, x + j, FieldModel.CSTATE_HINT_EMPTY);
                }
            }
            return;
        }
    }

    if (isLevel) {
        int col = x;
        while ((fieldModel.getAt(y, col - 1) == FieldModel.CSTATE_FILLED)
                && (col > 1))
            col--;

        setCellState(y + 1, col - 1, FieldModel.CSTATE_HINT_EMPTY);
        setCellState(y - 1, col - 1, FieldModel.CSTATE_HINT_EMPTY);

        if (finished)
            setCellState(y, col - 1, FieldModel.CSTATE_HINT_EMPTY);

        while ((fieldModel.getAt(y, col) == FieldModel.CSTATE_FILLED)
                && (col <= horDim)) {
            setCellState(y - 1, col, FieldModel.CSTATE_HINT_EMPTY);
            setCellState(y + 1, col, FieldModel.CSTATE_HINT_EMPTY);
            col++;
        }

        setCellState(y - 1, col, FieldModel.CSTATE_HINT_EMPTY);
        setCellState(y + 1, col, FieldModel.CSTATE_HINT_EMPTY);
        if (finished)
            setCellState(y, col, FieldModel.CSTATE_HINT_EMPTY);
    } else {

        int row = y;
        while ((fieldModel.getAt(row - 1, x) == FieldModel.CSTATE_FILLED)
                && (row > 1))
            row--;

        setCellState(row - 1, x - 1, FieldModel.CSTATE_HINT_EMPTY);
        setCellState(row - 1, x + 1, FieldModel.CSTATE_HINT_EMPTY);

        if (finished)
            setCellState(row - 1, x, FieldModel.CSTATE_HINT_EMPTY);

        while ((fieldModel.getAt(row, x) == FieldModel.CSTATE_FILLED)
```

```
                    && (row <= verDim)) {
                setCellState(row, x - 1, FieldModel.CSTATE_HINT_EMPTY);
                setCellState(row, x + 1, FieldModel.CSTATE_HINT_EMPTY);
                row++;
            }

            setCellState(row, x - 1, FieldModel.CSTATE_HINT_EMPTY);
            setCellState(row, x + 1, FieldModel.CSTATE_HINT_EMPTY);

            if (finished)
                setCellState(row, x, FieldModel.CSTATE_HINT_EMPTY);
        }
    }

    public boolean strike(final int y, final int x) {
        if (enemyMap.getAt(y, x) == FieldModel.CSTATE_EMPTY)
            setCellState(y, x, FieldModel.CSTATE_ATTACKED_EMPTY);

        if ((fieldModel.getAt(y, x) == FieldModel.CSTATE_FILLED)
                || (enemyMap.getAt(y, x) == FieldModel.CSTATE_EMPTY))
            return false;

        setCellState(y, x, FieldModel.CSTATE_FILLED);
        if (getTargetLength(y, x, fieldModel)
                == getTargetLength(y, x, enemyMap)) {
            targets[getTargetLength(y, x, fieldModel) - 1]++;
            markTarget(y, x, true);
        } else
            markTarget(y, x, false);
        return true;
    }
}

////////////////////////////////////////////////////////////////////////////////
//                          Класс AutoAllocation                              //
////////////////////////////////////////////////////////////////////////////////

package logics;

import field.FieldModel;
import logics.structs.OrientedPoint;

public final class AutoAllocation extends Allocation {
    public AutoAllocation(final FieldModel fieldModel) {
        super(fieldModel);
    }

    public final OrientedPoint getAllocation(final int length) {
        return getRandomPos(length);
    }
}

////////////////////////////////////////////////////////////////////////////////
//                           Класс AutoAttack                                 //
////////////////////////////////////////////////////////////////////////////////

package logics;

import field.FieldModel;
import logics.structs.OrientedPoint;
import logics.structs.Point2Dim;

/**
 * Класс автоматической атаки.
 */
public final class AutoAttack extends Attack {

    public AutoAttack(final FieldModel enemy) {
        super(enemy);

        for (int row = 0; row < verDim + 2; row++) {
            for (int col = 0; col < horDim + 2; col++) {
                if (enemy.getAt(row, col) == FieldModel.CSTATE_FILLED)
                    fieldModel.setAt(row, col, FieldModel.CSTATE_HINT_FILLED);
            }
        }
    }

    public final int getMaxTarget() {
        for (int t = 3; t >= 0; t--)
```

```
        if (targets[t] < 4 - t)
            return (t + 1);
    return 0;
}

/**
 * Возвращает возможное положение неподбитого корабля.
 */
public final Point2Dim getRandTarget() {
    final OrientedPoint pos = getRandomPos(getMaxTarget());
    return new Point2Dim(pos.getX(), pos.getY());
}

/**
 * Возвращает очередную координату цели при условии незаконченной атаки.
 */
public final Point2Dim haveTarget() {
    final Point2Dim target = new Point2Dim();
    final int[][] targs = new int[verDim + 2][horDim + 2];

    for (int row = 0; row < verDim + 2; row++)
        for (int col = 0; col < horDim + 2; col++)
            targs[row][col] = 0;

    int targsNum = 0;

    for (int row = 1; row <= verDim; row++) {
        for (int col = 1; col <= horDim; col++) {
            if (fieldModel.getAt(row, col) == FieldModel.CSTATE_FILLED) {
                if ((
                        (
                        fieldModel.getAt(row - 1, col)
                        == FieldModel.CSTATE_EMPTY
                        )
                        || (
                        fieldModel.getAt(row - 1, col)
                        == FieldModel.CSTATE_HINT_FILLED
                        )
                        )
                        && (row > 1)) {
                    targsNum++;
                    targs[row - 1][col] = 1;
                }

                if ((
                        (
                        fieldModel.getAt(row + 1, col)
                        == FieldModel.CSTATE_EMPTY
                        )
                        || (
                        fieldModel.getAt(row + 1, col)
                        == FieldModel.CSTATE_HINT_FILLED
                        )
                        )
                        && (row < verDim)) {
                    targsNum++;
                    targs[row + 1][col] = 1;
                }

                if ((
                        (
                        fieldModel.getAt(row, col - 1)
                        == FieldModel.CSTATE_EMPTY
                        )
                        || (
                        fieldModel.getAt(row, col - 1)
                        == FieldModel.CSTATE_HINT_FILLED
                        )
                        )
                        && (col > 1)) {
                    targsNum++;
                    targs[row][col - 1] = 1;
                }

                if ((
                        (
                        fieldModel.getAt(row, col + 1)
                        == FieldModel.CSTATE_EMPTY
                        )
                        || (
```

```
                        fieldModel.getAt(row, col + 1)
                        == FieldModel.CSTATE_HINT_FILLED
                        )
                        )
                        && (col < horDim)) {
                    targsNum++;
                    targs[row][col + 1] = 1;
                }
            }
        }
    }

    if (targsNum == 0)
        return null;

    final int rn = (int) Math.ceil(Math.random() * (targsNum - 1) + 1);
    int count = 0;

    for (int row = 1; row <= verDim; row++)
        for (int col = 1; col <= horDim; col++)
            if (targs[row][col] == 1) {
                count++;
                if (count == rn) {
                    target.setLocation(col, row);
                    return target;
                }
            }

    return null;
  }
}
```

```
////////////////////////////////////////////////////////////////////////////////
//                          Класс BaseProcessor                               //
////////////////////////////////////////////////////////////////////////////////

package logics;

import field.FieldModel;
import logics.structs.OrientedPoint;
import logics.structs.Point2Dim;
import logics.structs.Ship;

/**
 * Базовый класс для всех расставляющих и атакующих объектов, реализующий их
 * общую функциональность.
 */
abstract public class BaseProcessor {
    int horDim;
    int verDim;
    int[][][] auxField;
    FieldModel fieldModel;


    public BaseProcessor() {
    }

    public BaseProcessor(final FieldModel fieldModel) {
        initBaseField(fieldModel);
    }

    private final void initBaseField(final FieldModel fieldModel) {

        this.verDim = fieldModel.verDim();
        this.horDim = fieldModel.horDim();
        auxField = new int[2][verDim + 2][horDim + 2];

        // Заполнение горизонтальных полос.
        for (int row = 1; row <= verDim; row++)
            for (int col = horDim; col >= 1; col--)
                this.auxField[0][row][col] = horDim - col + 1;

        // Заполнение вертикальных полос.
        for (int col = 1; col <= horDim; col++)
            for (int row = verDim; row >= 1; row--)
                this.auxField[1][row][col] = verDim - row + 1;

        this.fieldModel = fieldModel;
    }
```

```
/**
 * Проверяет возможность установки фигуры заданной длины на доску.
 */
public final boolean checkAlloc(final int length) {
    for (int s = 0; s < 2; s++)
        for (int row = 1; row <= verDim; row++)
            for (int col = 1; col <= horDim; col++)
                if (auxField[s][row][col] >= length)
                    return true;
    return false;
}

/**
 * Возвращает длину максимальной последовательности незанятых клетов
 * в данной колонке.
 */
private int getColMax(final int col) {
    int max = -1;
    for (int row = 1; row <= verDim; row++)
        if (max < auxField[1][row][col])
            max = auxField[1][row][col];
    return max;
}

/**
 * Возвращает длину максимальной последовательности незанятых клетов
 * в данном ряду.
 */
private int getRowMax(final int row) {
    int max = -1;
    for (int col = 1; col <= horDim; col++)
        if (auxField[0][row][col] > max)
            max = auxField[0][row][col];
    return max;
}

/**
 * Возвращает список номеров колонок, содержащих
 * последовательность свободных клеток максимальной длины.
 */
private int[] getMaxiCols() {
    int max = -1;
    final int[] cols;
    int count = 0;

    for (int col = 1; col <= horDim; col++)
        if (max < getColMax(col))
            max = getColMax(col);

    for (int row = 1; row <= verDim; row++)
        if (max < getRowMax(row))
            max = getRowMax(row);

    if (max == -1)
        return null;

    for (int col = 1; col <= horDim; col++)
        if (max == getColMax(col))
            count++;

    cols = new int[count];
    count = 0;
    for (int col = 1; col <= horDim; col++)
        if (max == getColMax(col))
            cols[count++] = col;

    return cols;
}

/**
 * Возвращает список номеров рядов, содержащих
 * последовательность свободных клеток максимальной длины.
 */
private int[] getMaxiRows() {
    int count = 0;
    final int[] rows;
    int max = -1;

    for (int col = 1; col <= horDim; col++)
        if (max < getColMax(col))
```

```
            max = getColMax(col);

        for (int row = 1; row <= verDim; row++)
            if (max < getRowMax(row))
                max = getRowMax(row);

        if (max == -1)
            return null;

        for (int row = 1; row <= verDim; row++)
            if (max == getRowMax(row))
                count++;

        rows = new int[count];
        count = 0;
        for (int row = 1; row <= verDim; row++)
            if (getRowMax(row) == max)
                rows[count++] = row;

        return rows;
    }

    /**
     * Возвращает координаты следующего корабля для установки (удара)
     */
    final OrientedPoint getRandomPos(final int Length) {
        final OrientedPoint result = new OrientedPoint();

        final int[] MaxiRows = getMaxiRows();
        final int[] MaxiCols = getMaxiCols();

        final int rowsNumber = MaxiRows.length;
        final int colsNumber = MaxiCols.length;

        int rlNum = (int) Math.ceil(Math.random() * (colsNumber + rowsNumber));
        if (rlNum == 0)
            rlNum = 1;

        final int randLine;
        int randPos = 1;

        if (rlNum <= rowsNumber) // Рассматриваем горизонтальную установку
        {
            randLine = MaxiRows[rlNum - 1];

            final int max;
            int cellQuant = 0;
            final int[] line = new int[horDim + 2];
            for (int col = 0; col < horDim + 2; col++)
                line[col] = 0;

            max = getRowMax(randLine);

            for (int col = 1; col <= horDim; col++)
                if (auxField[0][randLine][col] == max)
                    for (int i = col; i < col + max; i++) {
                        line[i] = 1;
                        cellQuant++;
                    }

            final int rn = (int) Math.ceil(Math.random() * cellQuant);

            int count = 0;
            while (count != rn) {
                for (int col = 1; col <= horDim; col++) {
                    if ((line[col] == 1)
                            && (auxField[0][randLine][col] >= Length)
                            && (Length > 0)) {
                        count++;
                        if (rn == count) {
                            randPos = col;
                            break;
                        }
                    }
                }
            }
        } else                    // Рассматриваем вертикальную установку
        {
            randLine = MaxiCols[rlNum - rowsNumber - 1];
```

```
        final int max;
        int cellQuant = 0;
        final int[] line = new int[verDim + 2];
        for (int row = 0; row < verDim + 2; row++)
            line[row] = 0;

        max = getColMax(randLine);

        for (int row = 1; row <= verDim; row++)
            if (auxField[1][row][randLine] == max)
                for (int i = row; i < row + max; i++) {
                    line[i] = 1;
                    cellQuant++;
                }

        final int rn = (int) Math.ceil(Math.random() * cellQuant);

        int count = 0;
        while (count != rn) {
            for (int row = 1; row <= verDim; row++) {
                if ((line[row] == 1)
                        && (auxField[1][row][randLine] >= Length)
                        && (Length > 0)) {
                    count++;
                    if (rn == count) {
                        randPos = row;
                        break;
                    }
                }
            }
        }
    }

    if (rlNum <= rowsNumber) {
        result.setHorisontal();
        result.setLocation(randPos, randLine);
    } else {
        result.setVertical();
        result.setLocation(randLine, randPos);
    }

    return result;
}

/**
 * Помечает ячейку на вспомогательной доске как "подбитую"
 */
private void markCell(final Point2Dim p, final FieldModel.CellState state) {
    final int y = p.getY();
    final int x = p.getX();

    if ((x == horDim + 1) || (x == 0) || (y == verDim + 1) || (y == 0))
        return;

    auxField[0][y][x] = 0; // Помечаем как пустые
    auxField[1][y][x] = 0;

    if (state == FieldModel.CSTATE_FILLED) // Цель поражена
    {
        // Обнлюляем все ближайшие соседние клетки
        for (int t = 0; t < 2; t++) {
            for (int i = y - 1; i < y + 2; i++) {
                for (int j = x - 1; j < x + 2; j++) {
                    auxField[t][i][j] = 0;
                }
            }
        }
        // Заполняем вспомогательные поля для вертикальных полос
        if (y > 1) {
            for (int i = -1; i < 2; i++) {
                int row = y - 2;
                while ((auxField[1][row][x + i] != 0) && (row > 0)) {
                    auxField[1][row][x + i] = y - row - 1;
                    row--;
                }
            }
        }

        if (x > 1) // Заполняем вспомогательные поля для горизонтальных полос
        {
```

```
            for (int i = -1; i < 2; i++) {
                int col = x - 2;
                while ((auxField[0][y + i][col] != 0) && (col > 0)) {
                    auxField[0][y + i][col] = x - col - 1;
                    col--;
                }
            }
        }
    } else {
        int row = y - 1;
        // Заполняем вспомогательные поля для вертикальных полос
        while ((auxField[1][row][x] != 0) && (row > 0)) {
            auxField[1][row][x] = y - row;
            row--;
        }

        int col = x - 1;
        // Заполняем вспомогательные поля для горизонтальных полос
        while ((auxField[0][y][col] != 0) && (col > 0)) {
            auxField[0][y][col] = x - col;
            col--;
        }
    }
}

/**
 * Устанавливает корабль на доску и делает при это все необходимое.
 */
public final void putShip(final Ship ship) {
    final int y = ship.getY();
    final int x = ship.getX();
    final int length = ship.getLength();
    final boolean isVertical = ship.isVerticallyOriented();

    if ((x == -1) || (y == -1) || (length == -1))
        return;

    // Проверяем возможность установки фигуры начиная с указанной клетки
    if (!isVertical) {
        if (auxField[0][y][x] < length)
            return;
    } else if (auxField[1][y][x] < length)
        return;

    // Устанавливем фигуру на доску и помечаем соответствующим образом
    // вспомогательные поля

    if (isVertical) // Если фигура ориентирована горизонтально
    {
        // Помечаем все соседние с фигурой клетки
        for (int row = y; row < y + length; row++)
            setCellState(new Point2Dim(x, row), FieldModel.CSTATE_FILLED);
    } else {
        for (int col = x; col < x + length; col++)
            setCellState(new Point2Dim(col, y), FieldModel.CSTATE_FILLED);
    }
}

final void setCellState(final int y, final int x,
                        final FieldModel.CellState state) {
    setCellState(new Point2Dim(x, y), state);
}

/**
 * Выставляет состояние ячейки.
 */
private void setCellState(final Point2Dim p,
                          final FieldModel.CellState state) {
    if ((state == FieldModel.CSTATE_HINT_EMPTY)
            && (
            fieldModel.getAt(p.getY(), p.getX())
            == FieldModel.CSTATE_ATTACKED_EMPTY
            ))
        return;

    fieldModel.setAt(p.getY(), p.getX(), state);
    markCell(p, state);
}

public final FieldModel getFieldModel() {
```

```
        return fieldModel;
    }

    public final int[][][] getAuxField() {
        return auxField;
    }

    public int horDim() {
        return horDim;
    }

    public int verDim() {
        return verDim;
    }
}

////////////////////////////////////////////////////////////////////////////////
//                          Класс UserAllocation                              //
////////////////////////////////////////////////////////////////////////////////

package logics;

import field.FieldModel;
import logics.structs.OrientedPoint;
import logics.structs.Point2Dim;

/**
 * Класс Расстановки кораблей польхователя.
 */
public final class UserAllocation extends Allocation {

    public UserAllocation(final FieldModel fieldModel) {
        super(fieldModel);
    }

    public final boolean directFit(final OrientedPoint pos, final int length) {
        if (pos.isHorisontallyOriented())
            if (auxField[0][pos.getY()][pos.getX()] >= length)
                return true;
        if (pos.isVerticallyOriented())
            if (auxField[1][pos.getY()][pos.getX()] >= length)
                return true;
        return false;
    }

    public final boolean posFit(final Point2Dim p, final int length) {
        if ((auxField[0][p.getY()][p.getX()] >= length)
                || (auxField[1][p.getY()][p.getX()] >= length))
            return true;

        return false;
    }
}

////////////////////////////////////////////////////////////////////////////////
//                          Класс UserAttack                                  //
////////////////////////////////////////////////////////////////////////////////

package logics;

import field.FieldModel;
import logics.structs.Point2Dim;

/**
 * Класс атаки робота.
 */
public final class UserAttack extends Attack {
    private boolean strikeResult;
    private Point2Dim curTarget;

    public UserAttack(final FieldModel enemy) {
        super(enemy);
    }

    public final boolean strike(final int y, final int x) {
        strikeResult = super.strike(y, x);
        return strikeResult;
    }

    public final boolean getStrikeResult() {
```

```
        return strikeResult;
    }

    public final Point2Dim getCurTarget() {
        return curTarget;
    }

    public final void setCurTarget(final Point2Dim curTarget) {
        this.curTarget = curTarget;
    }

    public void showRobotFleet() {
        for (int row = 0; row < verDim + 2; row++) {
            for (int col = 0; col < horDim + 2; col++) {
                if ((enemyMap.getAt(row, col) == FieldModel.CSTATE_FILLED) &&
                        (fieldModel.getAt(row, col) == FieldModel.CSTATE_EMPTY))
                    fieldModel.setAt(row, col, FieldModel.CSTATE_HINT_FILLED);
            }
        }
    }
}

////////////////////////////////////////////////////////////////////////////////
//                          Класс OrientedPoint                               //
////////////////////////////////////////////////////////////////////////////////

package logics.structs;

/**
 * Класс - расширение <code>Point2Dim</code> до (направленного) вектора.
 * Направлений может быть два: горизонтальное (<code>HORISONTAL</code>) и
 * вертикальное (<code>VERTICAL</code>).
 */
public final class OrientedPoint extends Point2Dim {
    private final int HORISONTAL = 0;
    private final int VERTICAL = 1;
    private int orientation;

    public OrientedPoint() {
        orientation = -1;
    }

    public OrientedPoint(final OrientedPoint pos) {
        super(pos.getX(), pos.getY());
        orientation = pos.orientation;
    }

    public OrientedPoint(final Point2Dim p, final int orientation) {
        super(p);
        this.orientation = orientation;
    }

    public final int getOrientation() {
        return orientation;
    }

    public final boolean isVerticallyOriented() {
        return orientation == VERTICAL;
    }

    public final boolean isHorisontallyOriented() {
        return orientation == HORISONTAL;
    }

    public final void setHorisontal() {
        orientation = HORISONTAL;
    }

    public final void setVertical() {
        orientation = VERTICAL;
    }

    public final void setOrientation(final int orientation) {
        this.orientation = orientation;
    }
}
```

```
////////////////////////////////////////////////////////////////////////////////
//                          Класс Point2Dim                                   //
////////////////////////////////////////////////////////////////////////////////

package logics.structs;

import java.awt.*;

/**
 * Класс представляет собой реализацию стандарстной двумерной точки. В отличие
 * от {@link java.awt.Point} не имеет public атрибутов.
 */
public class Point2Dim {
    private int x; // Координаты
    private int y; // Двумерной точки

    public Point2Dim(final Point2Dim p) {
        this.x = (p != null) ? p.getX() : -1;
        this.y = (p != null) ? p.getY() : -1;
    }

    public Point2Dim() {
        this(-1, -1);
    }

    public Point2Dim(final int x, final int y) {
        this.x = x;
        this.y = y;
    }

    public Point2Dim(final Point p) {
        this(p.x, p.y);
    }

    public final int getX() {
        return x;
    }

    public final void setX(final int x) {
        this.x = x;
    }

    public final int getY() {
        return y;
    }

    public final void setY(final int y) {
        this.y = y;
    }

    public final void setLocation(final Point2Dim newLoc) {
        setLocation(newLoc.getX(), newLoc.getY());
    }

    public final void setLocation(final int x, final int y) {
        this.x = x;
        this.y = y;
    }
}

////////////////////////////////////////////////////////////////////////////////
//                             Класс Ship                                     //
////////////////////////////////////////////////////////////////////////////////

package logics.structs;

/** Класс, являющийся реализацией корабля. Определяется
 * позицией, вектором направления и длиной. */
public final class Ship {
    private final OrientedPoint position;
    private final int length;

    public Ship(final int length) {
        this(new OrientedPoint(), length);
    }

    private Ship(final OrientedPoint pos, final int length) {
        this.length = length;
        position = new OrientedPoint(pos);
    }
```

```
    public final int getLength() {
        return length;
    }

    /** Индикатор того, что корабль уже установлен. */
    public final boolean isAllocated() {
        if ((position.getX() > -1) && (position.getY() > -1) && (length > -1))
            return true;
        return false;
    }

    /**  Устанавливает новые положение и ориентацию корабля. */
    public final void setPosition(final OrientedPoint newPos) {
        position.setLocation(newPos);
        position.setOrientation(newPos.getOrientation());
    }

    /** Возвращает Y-координату положения корабля. */
    public final int getY() {
        return position.getY();
    }

    /** Возвращает X-координату положения корабля. */
    public final int getX() {
        return position.getX();
    }

    public final boolean isVerticallyOriented() {
        return position.isVerticallyOriented();
    }
}

////////////////////////////////////////////////////////////////////////////////
//                          Класс LogBuffer                                   //
////////////////////////////////////////////////////////////////////////////////

package logs;

import java.util.ArrayList;

public class LogBuffer extends ArrayList {
    // Количестко объектов в буфере
    private int loggedItems;

    public LogBuffer(int initialCapacity) {
        super(initialCapacity);
        loggedItems = 0;
    }

    public LogBuffer() {
        loggedItems = 0;
    }

    public int getLastLogged() {
        return loggedItems;
    }

    public boolean isFullyLogged() {
        return (size() != 0) && (loggedItems == size());
    }

    public void setFullyUpdated() {
        loggedItems = 0;
    }

    public void setFullyLogged() {
        loggedItems = size();
    }
}

////////////////////////////////////////////////////////////////////////////////
//                         Класс LoggingModel                                 //
////////////////////////////////////////////////////////////////////////////////

package logs;

import aut.autbase.Automaton;
import event.EnumEvent;
import logs.logatoms.*;
```

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;

/**
 * Класс, реализующий модель протоколов.
 */
public final class LoggingModel implements ActionListener {

    private LogBuffer logBuffer = null;

    private int indent = 0;
    private LogView listener = null;

    public LoggingModel(final LogView listener) {
        logBuffer = new LogBuffer(1000);
        this.listener = listener;
    }

    /**
     * Оповещает обработчика. Должен быть <code>syncrhonized</code>, чтобы
     * избежать ситуации оповещения несуществующих обработчиков.
     */
    private synchronized void notifyListener(LogBuffer logBuffer) {
        if (listener == null)
            return;
        listener.logUpdated(logBuffer);
    }

    private synchronized void notifyListener() {
        if (listener == null)
            return;
        listener.logUpdated();
    }

    private synchronized void putLog(LogAtom logAtom) {
        synchronized (logBuffer) {
            logBuffer.add(logAtom);
            listener.logUpdated(logBuffer);
        }
    }

    /**
     * Протоколирование запуска автоматов.
     */
    public final void logAutRunning(final Automaton aut) {
        putLog(new LogAutRunAtom(indent, aut));
        increaseIndent();
    }

    /**
     * Протоколирование останова автоматов.
     */
    public final void logAutTermination(final Automaton aut) {
        decreaseIndent();
        putLog(new LogAutTerminationAtom(indent, aut));
    }

    /**
     * Протоколирование событий.
     */
    public final void logEvent(final Automaton aut, final EnumEvent event) {
        putLog(new LogEventAtom(indent, aut, event));
    }

    /**
     * Протоколирование выходных воздействий.
     */
    public final void logAction(Automaton.Action action) {
        putLog(new LogActionAtom(indent, action));
    }

    /**
     * Протоколирование входных переменных.
     */
    public final void logVariable(Automaton.Variable variable,
                                  final boolean value) {
        putLog(new LogVarAtom(indent, variable, value));
    }
```

```
    /**
     * Протоколирование основных этапов работы программы (секций).
     */
    public final void logSection(final String section) {
        putLog(new LogSectionAtom(indent, section));
    }

    /**
     * Протоколирование объектов.
     */
    public final void logObject(final String objectName) {
        putLog(new LogObjectAtom(indent, objectName));
    }

    /**
     * Протоколирование изменения состояний автоматов.
     */
    public final void logAutSwitching(final Automaton aut) {
        putLog(new LogSwitchingAtom(indent, aut));
    }

    public final ArrayList getLogAtoms() {
        return logBuffer;
    }

    private void increaseIndent() {
        this.indent++;
    }

    private void decreaseIndent() {
        this.indent--;
    }

    private void clearLogs() {
        logBuffer.clear();
        logBuffer.setFullyUpdated();
        notifyListener(logBuffer);
    }

    public final void actionPerformed(final ActionEvent e) {
        clearLogs();
        notifyListener();
    }
}
```

```
////////////////////////////////////////////////////////////////////////////////
//                          Класс LogView                                     //
////////////////////////////////////////////////////////////////////////////////

package logs;


public interface LogView {
    public void logUpdated(LogBuffer logBuffer);

    public void logUpdated();
}
```

```
////////////////////////////////////////////////////////////////////////////////
//                          Класс LogActionAtom                               //
////////////////////////////////////////////////////////////////////////////////

package logs.logatoms;

import aut.autbase.Automaton;
import ui.LoggingViewArea;

public class LogActionAtom extends LogAtom {
    private final Automaton.Action action;

    public LogActionAtom(int indent, Automaton.Action action) {
        super(indent);
        this.action = action;
    }

    protected String getMsgContent() {
        return "* " + action.getId() + ": " + action.getDescirption() + "\n";
    }
```

```
    public boolean enabledLogging(LoggingViewArea.LoggingPermissions logPerms) {
        return logPerms.isLoggingActions();
    }
}


////////////////////////////////////////////////////////////////////////////////
//                            Класс LogAtom                                   //
////////////////////////////////////////////////////////////////////////////////

package logs.logatoms;

import ui.LoggingViewArea;

/**
 * Класс, представляющий атомарное сообщение протокола.
 */
public abstract class LogAtom {
    private static final String INDENT_STRING = "    ";
    private String stringCache = null;
    /* отступ строки (в <code>INDENT_STRING</code>)*/
    private int indent = 0;

    public LogAtom(int indent) {
        this.indent = indent;
    }

    /**
     * Конвертирует численное представление отступа в строковое.
     */
    private StringBuffer indentToString(int indent) {
        final StringBuffer result = new StringBuffer();
        for (int i = 0; i < indent; i++)
            result.append(INDENT_STRING);
        return result;
    }

    /**
     * Возвращает строковое представление сообщения.
     */
    abstract protected String getMsgContent();

    /**
     * Возвращает полное строковое представление сообщения с отступом.
     */
    public String toString() {
        if (stringCache == null)
            stringCache = indentToString(indent) +
                    getMsgContent();
        return stringCache;
    }

    public abstract boolean enabledLogging(
            LoggingViewArea.LoggingPermissions logPerms);
}

////////////////////////////////////////////////////////////////////////////////
//                            Класс LogAutRunAtom                             //
////////////////////////////////////////////////////////////////////////////////

package logs.logatoms;

import aut.autbase.Automaton;
import ui.LoggingViewArea;

public class LogAutRunAtom extends LogAtom {
    private final Automaton aut;

    public LogAutRunAtom(int indent, Automaton aut) {
        super(indent);
        this.aut = aut;
    }

    protected String getMsgContent() {
        return aut.getAutID() + "." + aut.getState().getId() + " (\""
                + aut.getDescirption()
                + "\".\""
                + aut.getState().getDescirption() + "\")" + " {\n";
    }
```

```
    public boolean enabledLogging(LoggingViewArea.LoggingPermissions logPerms) {
        return logPerms.isLoggingAutomatons();
    }
}

////////////////////////////////////////////////////////////////////////////////
//                          Класс LogAutTerminationAto                        //
////////////////////////////////////////////////////////////////////////////////

package logs.logatoms;

import aut.autbase.Automaton;
import ui.LoggingViewArea;

public class LogAutTerminationAtom extends LogAtom {
    private final Automaton aut;

    public LogAutTerminationAtom(int indent, final Automaton aut) {
        super(indent);
        this.aut = aut;
    }

    protected String getMsgContent() {
        return "} " + aut.getAutID() + "." + aut.getState().getId() + " (\""
                + aut.getDescirption()
                + "\".\""
                + aut.getState().getDescirption() + "\")\n\n";
    }

    public boolean enabledLogging(LoggingViewArea.LoggingPermissions logPerms) {
        return logPerms.isLoggingAutomatons();
    }
}

////////////////////////////////////////////////////////////////////////////////
//                          Класс LogEventAtom                                //
////////////////////////////////////////////////////////////////////////////////

package logs.logatoms;

import aut.autbase.Automaton;
import event.EnumEvent;
import ui.LoggingViewArea;

public class LogEventAtom extends LogAtom {
    private final Automaton aut;
    private final EnumEvent event;

    public LogEventAtom(int indent, Automaton aut, EnumEvent event) {
        super(indent);
        this.aut = aut;
        this.event = event;
    }

    protected String getMsgContent() {
        if (event == null)
            return new String();
        final String result = "# Автомат " + "\"" + aut.getAutID()
                + "\" обработал событие: "
                + event.toString()
                + "\n";
        return result;
    }

    public boolean enabledLogging(LoggingViewArea.LoggingPermissions logPerms) {
        return logPerms.isLoggingEvents();
    }
}

////////////////////////////////////////////////////////////////////////////////
//                          Класс LogObjectAtom                               //
////////////////////////////////////////////////////////////////////////////////

package logs.logatoms;

import ui.LoggingViewArea;

public class LogObjectAtom extends LogAtom {
    final private String objectName;
```

```
    public LogObjectAtom(int indent, String objectName) {
        super(indent);
        this.objectName = objectName;
    }


    protected String getMsgContent() {
        return "Для объекта " + objectName + "\n";
    }

    public boolean enabledLogging(LoggingViewArea.LoggingPermissions logPerms) {
        return logPerms.isLoggingObjects();
    }
}

////////////////////////////////////////////////////////////////////////////////
//                          Класс LogSectionAtom                              //
////////////////////////////////////////////////////////////////////////////////

package logs.logatoms;

import ui.LoggingViewArea;

public class LogSectionAtom extends LogAtom {
    final private String section;

    public LogSectionAtom(int indent, String section) {
        super(indent);
        this.section = section;
    }

    protected String getMsgContent() {
        return ">>>>>>> " + section + " <<<<<<<\n";
    }

    /**
     * Всегда возвращает <code>true</code>, потому что протоколирование
     * секций разрешено всегда.
     */
    public boolean enabledLogging(LoggingViewArea.LoggingPermissions logPerms) {
        return true;
    }
}

////////////////////////////////////////////////////////////////////////////////
//                          Класс LogSwitchingAtom                            //
////////////////////////////////////////////////////////////////////////////////

package logs.logatoms;

import aut.autbase.Automaton;
import ui.LoggingViewArea;

public class LogSwitchingAtom extends LogAtom {
    final private Automaton aut;

    public LogSwitchingAtom(int indent, Automaton automaton) {
        super(indent);
        this.aut = automaton;
    }

    protected String getMsgContent() {
        return aut.getAutID() + "->" + aut.getState().getId() + " (\""
                + aut.getDescirption()
                + "\"->\""
                + aut.getState().getDescirption() + "\")" + "\n";
    }

    public boolean enabledLogging(LoggingViewArea.LoggingPermissions logPerms) {
        return logPerms.isLoggingAutomatons();
    }
}


////////////////////////////////////////////////////////////////////////////////
//                          Класс LogVarAtom                                  //
////////////////////////////////////////////////////////////////////////////////

package logs.logatoms;
```

```
import aut.autbase.Automaton;
import ui.LoggingViewArea;

public class LogVarAtom extends LogAtom {
    private final Automaton.Variable variable;
    private final boolean value;

    public LogVarAtom(int indent, Automaton.Variable variable, boolean value) {
        super(indent);
        this.value = value;
        this.variable = variable;
    }

    protected String getMsgContent() {
        final String strValue = (value) ? "ДА" : "НЕТ";
        final String result = variable.getId() + ": "
                + variable.getDescirption()
                + "? - "
                + strValue
                + "\n";
        return result;
    }

    public boolean enabledLogging(LoggingViewArea.LoggingPermissions logPerms) {
        return logPerms.isLoggingVariables();
    }
}

////////////////////////////////////////////////////////////////////////////////
//                          Класс BattleApplet                                //
////////////////////////////////////////////////////////////////////////////////

package ui;

import javax.swing.*;
import java.awt.*;

/**
 * Апплет
 */
public class BattleApplet extends JApplet {

    public final void start() {
        BattleMain battle = new BattleMain();
        setSize(new Dimension(600, 480));
        battle.initInterfaceAndAutomatons(getContentPane());
        doLayout();
        repaint();
    }
}

////////////////////////////////////////////////////////////////////////////////
//                          Класс BattleApplication                           //
////////////////////////////////////////////////////////////////////////////////

package ui;

import javax.swing.*;

/**
 * Отдельное приложение
 */
public class BattleApplication {

    public static void main(String[] args) {
        JFrame frame = new JFrame();
        BattleMain battle = new BattleMain();
        battle.initInterfaceAndAutomatons(frame.getContentPane());
        frame.setSize(620, 520);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setResizable(false);
        frame.show();
    }
}
```

```
////////////////////////////////////////////////////////////////////////////////
//                          Класс BattleMain                                  //
////////////////////////////////////////////////////////////////////////////////

package ui;

import aut.*;
import event.EventStorage;
import field.Field;
import logs.LoggingModel;

import javax.swing.*;
import java.awt.*;

public final class BattleMain {
    private A0 a0;
    private TopPane topPane;
    private LoggingViewArea logsArea;

    public void initInterfaceAndAutomatons(Container contentPane) {
        contentPane.setLayout(new GridLayout(2, 1));
        topPane = new TopPane(new Field(), new Field());
        contentPane.add(topPane);
        logsArea = new LoggingViewArea(new JTextArea());
        contentPane.add(logsArea);
        // Создание модели протоколирования
        final LoggingModel logs = new LoggingModel(logsArea);

        logsArea.start(); // Запускаем отдельных поток для обработки протоколов

        EventStorage eventStorage = new EventStorage();
// LOGGING CONNECTIONS /BEGIN/

        final A1 a1 = new A1(logs, eventStorage);
        final A2 a2 = new A2(logs, eventStorage);
        final A4 a4 = new A4(logs, eventStorage);
        final A3 a3 = new A3(logs, a4, eventStorage);

// LOGGING CONNECTIONS /END/

// EVENT SOURCE - EVENT SOURCE CONNECTIONS /BEGIN/
        topPane.getControls().getBtnEventProvider().setListener(eventStorage);
        topPane.getControls().getBtnClearLogs().addActionListener(logs);
// EVENT SOURCE - EVENT SOURCE CONNECTIONS /END/

        a0 = new A0(logs, a1, a2, a3, a4, eventStorage);

// LOGGING CONTROL CONNECTIONS /BEGIN/
        topPane.getControls().getCbLogActs().addItemListener(logsArea);
        topPane.getControls().getCbLogAuts().addItemListener(logsArea);
        topPane.getControls().getCbLogObjects().addItemListener(logsArea);
        topPane.getControls().getCbLogVars().addItemListener(logsArea);

        topPane.getControls().getCbLogAuts().setSelected(true);
        topPane.getControls().getCbLogObjects().setSelected(true);
        topPane.getClearBtn().addActionListener(logs);
// LOGGING CONTROL CONNECTIONS /END/

        a0.setFieldViews(topPane.getUserFleet(), topPane.getRobotFleet());

        a0.start();
    }
}

////////////////////////////////////////////////////////////////////////////////
//                          Класс Button                                      //
////////////////////////////////////////////////////////////////////////////////

package ui;

import javax.swing.*;
import java.awt.*;

public final class Button extends JButton {
    public Button(final String name, final String text) {
        super(text);
        setName(name);
        setFont(new Font("Dialog", Font.TRUETYPE_FONT, 11));
    }
}
```

```
////////////////////////////////////////////////////////////////////////////////
//                          Класс CheckBox                                    //
////////////////////////////////////////////////////////////////////////////////

package ui;

import javax.swing.*;

public final class CheckBox extends JCheckBox {
    public CheckBox(final String text, final boolean selected,
                    final String name) {
        super(text, selected);
        setName(name);
    }
}

////////////////////////////////////////////////////////////////////////////////
//                          Класс ControlsPane                                //
////////////////////////////////////////////////////////////////////////////////

package ui;

import event.ButtonEventProvider;

import javax.swing.*;
import java.awt.*;

public final class ControlsPane extends JPanel {
    private final Button BtnRestart;
    private final Button BtnClearLogs;
    private final CheckBox cbLogObjects;
    private final CheckBox cbLogAuts;
    private final CheckBox cbLogVars;
    private final CheckBox cbLogActs;
    private final ButtonEventProvider btnEventProvider;

    public ButtonEventProvider getBtnEventProvider() {
        return btnEventProvider;
    }

    public final Button getBtnClearLogs() {
        return BtnClearLogs;
    }

    public ControlsPane() {
        setLayout(new GridLayout(7, 1, 0, 5));
        BtnRestart = new Button("Restart", "Перезапустить");
        BtnClearLogs = new Button("ClearLogs", "Очистить логи");
        final JLabel label = new JLabel("Логи:");
        label.setFont(new Font("Serif", Font.PLAIN, 14));

        cbLogObjects = new CheckBox("Объектов", false, "LogObjects");
        cbLogAuts = new CheckBox("Автоматов", false, "LogAutomatons");
        cbLogVars = new CheckBox("Вх. переменных", false, "LogVaribles");
        cbLogActs = new CheckBox("Вых. воздействий", false, "LogActions");

        add(label);
        add(cbLogObjects);
        add(cbLogAuts);
        add(cbLogVars);
        add(cbLogActs);
        add(BtnRestart);
        add(BtnClearLogs);

        btnEventProvider = new ButtonEventProvider();
        BtnRestart.addActionListener(btnEventProvider);
    }

    public final CheckBox getCbLogObjects() {
        return cbLogObjects;
    }

    public final CheckBox getCbLogAuts() {
        return cbLogAuts;
    }

    public final CheckBox getCbLogVars() {
        return cbLogVars;
    }
```

```
    public final CheckBox getCbLogActs() {
        return cbLogActs;
    }
}

////////////////////////////////////////////////////////////////////////////////
//                          Класс LoggingViewArea                             //
////////////////////////////////////////////////////////////////////////////////

package ui;

import logs.LogBuffer;
import logs.LogView;
import logs.logatoms.LogAtom;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.util.ArrayList;
import java.util.Iterator;

/**
 * Класс, реализует область отображения протоколов.
 */
public final class LoggingViewArea extends JScrollPane implements LogView,
        ItemListener, Runnable {

    private LogBuffer logBuffer = null;
    private Thread thread = null;

    public class LoggingPermissions {
        private boolean loggingActions;     // Показывать протоколы выходных воздействий
        private boolean loggingVariables;   // Показывать протоколы входных переменных
        private boolean loggingObjects;     // Показывать протоколы объектов
        private boolean loggingAutomatons;  // Показывать протоколы автоматов
        private boolean loggingEvents;      // Показывать протоколы событий

        private void setLoggingActions(final boolean loggingActions) {
            this.loggingActions = loggingActions;
        }

        private void setLoggingVariables(final boolean loggingVariables) {
            this.loggingVariables = loggingVariables;
        }

        private void setLoggingObjects(final boolean loggingObjects) {
            this.loggingObjects = loggingObjects;
        }

        private void setLoggingAutomatons(final boolean loggingAutomatons) {
            this.loggingAutomatons = loggingAutomatons;
        }

        private void setLoggingEvents() {
            this.loggingEvents = loggingVariables || loggingActions;
        }

        public boolean isLoggingActions() {
            return loggingActions;
        }

        public boolean isLoggingEvents() {
            return loggingEvents;
        }

        public boolean isLoggingVariables() {
            return loggingVariables;
        }

        public boolean isLoggingObjects() {
            return loggingObjects;
        }

        public boolean isLoggingAutomatons() {
            return loggingAutomatons;
        }
    };
```

```
private LoggingPermissions logPerms;

private final JTextArea textArea;
private String lastStringLogged;

public LoggingViewArea(final JTextArea textArea) {
    super(textArea);
    logPerms = new LoggingPermissions();
    setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

    lastStringLogged = "";

    this.textArea = textArea;
    textArea.setColumns(35);
    textArea.setRows(5);
    textArea.setText("");
    textArea.setFont(new Font("Arial", Font.PLAIN, 11));
    textArea.setLineWrap(true);
    textArea.setWrapStyleWord(true);
}

public final void logUpdated(LogBuffer logBuffer) {
    synchronized (logBuffer) {
        this.logBuffer = logBuffer;
        logBuffer.notify();
    }
}

public final void logUpdated() {
    textArea.setText("");

    logBuffer.setFullyUpdated();
    logUpdated(logBuffer);
}

public final void start() {
    thread = new Thread(this);
    thread.start();
}

public void run() {
    while (thread != null && !thread.isInterrupted()) {
        ArrayList a_tmpList = new ArrayList();
        if (logBuffer != null) {
            synchronized (logBuffer) {
                for (int i = logBuffer.getLastLogged(); i <
                        logBuffer.size(); i++) {
                    a_tmpList.add(logBuffer.get(i));
                }
                if (logBuffer.isFullyLogged()) {
                    try {
                        logBuffer.wait();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                } else {
                    logBuffer.setFullyLogged();
                }
            }
        }

        StringBuffer logContent = new StringBuffer();
        for (Iterator iterator = a_tmpList.iterator(); iterator.hasNext();) {
            final LogAtom logAtom = (LogAtom) iterator.next();
            final String filteredStr = logAtom.enabledLogging(logPerms)
                    ? logAtom.toString() : "";
            if (filteredStr.equals(lastStringLogged)
                    && filteredStr.trim().equals(""))
                continue;
            logContent.append(filteredStr);
            lastStringLogged = (filteredStr != "")
                    ? filteredStr : lastStringLogged;
        }
        addString(logContent);
    }
}
```

```
     /**
     * Добавляет строку в протокол.
     */
    private void addString(final StringBuffer str) {
        textArea.append(str.toString());
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
//                textArea.append(str.toString());
                String a_str = textArea.getText();
                if (a_str != null) {
                    try {
                        textArea.setCaretPosition(a_str.length());
                    } catch (IllegalArgumentException e) {
                        e.printStackTrace();
                    }
                }
            }
        });
    }

    public final void itemStateChanged(final ItemEvent e) {
        final String name = ((CheckBox) e.getSource()).getName();
        if (name.equals("LogObjects")) {
            logPerms.setLoggingObjects(
                    e.getStateChange() == ItemEvent.SELECTED);
        } else if (name.equals("LogAutomatons")) {
            logPerms.setLoggingAutomatons(
                    e.getStateChange() == ItemEvent.SELECTED);
        } else if (name.equals("LogVaribles")) {
            logPerms.setLoggingVariables(
                    e.getStateChange() == ItemEvent.SELECTED);
        } else if (name.equals("LogActions")) {
            logPerms.setLoggingActions(
                    e.getStateChange() == ItemEvent.SELECTED);
        }
        logPerms.setLoggingEvents();
        if (logBuffer != null) {
            logUpdated();
        }
    }
}

////////////////////////////////////////////////////////////////////////////////
//                          Класс MyButton                                    //
////////////////////////////////////////////////////////////////////////////////

package ui;

import javax.swing.*;
import java.awt.*;

public final class MyButton extends JButton {
    public MyButton(final String name, final String text) {
        super(text);
        setName(name);
        setFont(new Font("Dialog", Font.TRUETYPE_FONT, 11));
    }
}

////////////////////////////////////////////////////////////////////////////////
//                          Класс MyCheckBox                                  //
////////////////////////////////////////////////////////////////////////////////

package ui;

import javax.swing.*;

public final class MyCheckBox extends JCheckBox {
    public MyCheckBox(final String text, final boolean selected,
                      final String name) {
        super(text, selected);
        setName(name);
    }
}
```

```
////////////////////////////////////////////////////////////////////////////////
//                          Класс TopPane                                     //
////////////////////////////////////////////////////////////////////////////////

package ui;

import field.Field;

import javax.swing.*;
import java.awt.*;

final class TopPane extends JPanel {
    private final Field robotFleet;
    private final Field userFleet;
    private final ControlsPane controls;

    public final ControlsPane getControls() {
        return controls;
    }

    public final Button getClearBtn() {
        return controls.getBtnClearLogs();
    }

    public TopPane(final Field robotFleet, final Field userFleet) {
        super();
        setLayout(new FlowLayout(FlowLayout.LEFT, 15, 0));
        this.robotFleet = robotFleet;
        this.userFleet = userFleet;
        controls = new ControlsPane();
        add(robotFleet);
        add(userFleet);
        add(controls);
    }

    public final Field getRobotFleet() {
        return robotFleet;
    }

    public final Field getUserFleet() {
        return userFleet;
    }
}

////////////////////////////////////////////////////////////////////////////////
//                          Класс A0ControlledObjects                         //
////////////////////////////////////////////////////////////////////////////////
package aut;

import event.ProviderListener;
import field.Field;

public class A0ControlledObjects {

    private Field userFieldView;
    private Field robotFieldView;

    public A0ControlledObjects(Field userFieldView, Field robotFieldView) {
        this.userFieldView = userFieldView;
        this.robotFieldView = robotFieldView;
    }

    public Field getUserFieldView() {
        return userFieldView;
    }

    public Field getRobotFieldView() {
        return robotFieldView;
    }

    public void setUserFieldListener(ProviderListener listener) {
        userFieldView.getMouseEventProvider().setListener(listener);
    }

    public void setRobotFieldListener(ProviderListener listener) {
        robotFieldView.getMouseEventProvider().setListener(listener);
    }
}
```