МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

01.03.02 ПРИКЛАДНАЯ МАТЕМАТИКА И ИНФОРМАТИКА

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ И КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ

БАКАЛАВРСКАЯ РАБОТА

на тему: **Использование игрового подхода в задачах выбора средств защиты информации**

Студент А. М. Васильев _____

Руководитель Г. А. Тырыгина _____

Консультант по аннотации Н. В. Ященко _____

Допустить к защите

Заведующий кафедрой к.т.н., доцент А.В. Очеповский _____

«_____» _____ 2017 г.

Тольятти 2017

АННОТАЦИЯ

Темой данной выпускной квалификационной работы является «Использование игрового подхода в задачах выбора средств защиты информации».

Работа выполнена студентом Тольяттинского Государственного Университета, института математики, физики и информационных технологий, группы ПМИб – 1301, Васильевым Александром Михайловичем.

Объектом исследования является, матричная модель конфликта «злоумышленник – администратор».

Целью работы является, разработка программы для выбора средств защиты информации на основе игрового подхода.

Предметом исследования является, определение оптимальных стратегий.

Задачами работы являются:

1. теоретические основания игрового подхода;
2. построение математических моделей, реализующих игровой подход;
3. реализация программы выбора оптимальной стратегии.

Выпускная квалификационная работа состоит из введения, трех глав, заключения, списка литературы и приложения.

Первая глава посвящена исследованию теоретических основ матричной теории игр.

Во второй главе описаны возможные стратегии злоумышленника и администратора. Рассматривается построение математических моделей конфликта двух лиц «злоумышленник – администратор».

В третьей главе описан жизненный цикл процесса разработки программы.

Результатом работы является программа, которая позволяет выбрать наиболее оптимальный набор средств защиты информации.

Выпускная квалификационная работа содержит пояснительную записку объемом 59 страниц, включая 8 иллюстраций, 6 таблиц, список литературы из 20 наименований, 2 приложения.

ABSTRACT

The title of the graduation work is “Application of the Game Approach for Selection of Optimal Set of Means for Protection of Information”.

The object of the graduation work is a matrix model of conflict: “hacker – administrator”.

The subject is the selection of optimal strategies for protection of information.

The aim of the work is the application of the game approach for tasks which are connected with selection of an optimal strategy for the protection of information.

The relevance of the work is to search new approaches for protection of information systems in view of their intensive development.

This work consists of introduction, three parts, conclusion, references, and 2 appendices.

The first part is devoted to the study of theoretical basis of the matrix game theory.

In the second part there is the description of possible strategies of hacker and administrator and to the analysis of mathematical models of conflict “hacker – administrator”.

The third part gives the description of the program development process.

The result of this work is a program that can find an effective set of means for protection of information.

The graduation work consists of 59 pages, 6 tables, 8 figures, and 20 references.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В XXI веке информационные технологии переживают новый этап инноваций, причиной которой становится проникновении Интернет технологий во многие сферы деятельности.

Большинство крупных компаний активно применяют и развивают интернет-коммерцию. Важным фактором роста конкурентоспособности компании считается применении информационных систем и интернет-технологий для управления бизнесом, что кардинально меняет подход в построении информационной структуры так, как все больше корпоративных систем имеют доступ к Глобальной сети. Подключение к сети интернет этих систем, может привести к тому, что хранящаяся информация становиться объектом нападения различных злоумышленников или хакеров. Злоумышленник при помощи несанкционированного доступа может похитить информацию, а посредством компьютерной атаки либо уничтожить ее, либо временно ограничить доступ к ней. Ежедневно злоумышленники подвергают угрозам различные информационные ресурсы, с целью заполучить доступ к ним с помощью определенной атаки. Данные атаки становятся все более искусными и упрощенными в исполнении.

Основной причиной атак послужил ряд ограничений, свойственных протоколу TCP/IP. В самых первых версиях в архитектуре протокола IP отсутствовали спецификации требований безопасности, что послужило изначальным уязвимостям. Только несколько лет спустя, были предложены и описаны технические спецификации и стандарты (RFC).

Защита информационных систем – это ключевая задача любой службы безопасности на предприятии или в организации. Защита информации способна уменьшить вероятность утечки (разглашения), модификации (умышленного искажения) или утраты (уничтожения) информации, представляющей определенную ценность для ее владельца. Этим и определяется актуальность выпускной квалификационной работы.

На сегодняшний день рынок переполнен огромным количеством средств защиты компьютерных систем, и администратор безопасности может принимать субъективное решение для выбора программных продуктов. Решение этой проблемы в настоящее время приобретает исключительно важное значение ввиду интенсивного развития информационных систем и, как следствие, необходимости поиска новых подходов к их защите.

В данной работе рассматривается и применяется игровой подход, для защиты информационных ресурсов.

Объект выпускной квалификационной работы: матричная модель конфликта: злоумышленник – администратор.

Предмет выпускной квалификационной работы: определение оптимальных стратегий.

Целью выпускной квалификационной работы: разработка программы для выбора средств защиты информации на основе игрового подхода.

Для достижения цели необходимо реализовать следующие задачи:

1. теоретические основания игрового подхода;
2. построение математических моделей, реализующих игровой подход;
3. реализация программы выбора оптимальной стратегии.

ГЛАВА 1 ОСНОВНЫЕ ПОЛОЖЕНИЯ ТЕОРИИ МАТРИЧНЫХ ИГР

1.1 Общие элементы матричных игр

Теория игр – это один из разделов прикладной математики, в котором исследуются математические модели оптимальных решений в условиях конфликта, т.е. в ситуации, в которой сталкиваются интересы двух сторон, каждая сторона пытается достичь наибольшей выгоды для себя.

Например, с помощью, теории игр можно описать конфликт противостояние администратора и злоумышленника в виде математической модели, которую называют игрой, а участников конфликта игроками.

Игровая модель конфликта может помочь спрогнозировать и дать оценку достигнутому уровню защиты информационной системы с учетом выбора оптимальных стратегий поведения игроков. Причем предполагается, что игра повторяется многократно, что позволит выбрать оптимальную стратегию администратору, обеспечивающую максимально возможную защиту информационной системы.

Очевидно, что приобретение и установка всевозможных средств защиты информационной системы, ведет к материальным затратам организации или предприятия, а также требует наличия инструкций по эффективному использованию программ. Поэтому необходимо вооружить системного администратора или службу безопасности компьютерной сети оптимальной стратегией обеспечения защиты информационной системы.

Конфликт – это ситуация, в которой сталкиваются интересы двух сторон, где каждая сторона пытается достичь наибольшей выгоды для себя, сводя к минимуму свои потери. Человеческая практика наполнена всевозможными конфликтными ситуациями, которые требуют разрешения.

Для того, чтобы разрешить конфликт описывается математическая модель такой ситуации. Для разрешения конфликта используется математическая теория, помогающая принимать рациональные решения в определенных условиях.

Игра может состоять из двух или более игроков, которые поочередно или одновременно делают ходы. Игра осуществляется по установленному набору правил игры. Правила игры описывают:

1. что разрешается или что требуется делать игроку, при множестве всех возможных обстоятельств;
2. сведения действий, которые получает каждый игрок;
3. момент окончания игры, сумму, которую уплачивает или получает каждый игрок, и цель каждого игрока;
4. число ходов, число игроков и платеж (платежные функции, или функции выигрышей игроков), который является количественной оценкой результатов игры.

Действия каждого игрока, его поведение, т.е. выбор хода, определяется набором правил, которые составляют стратегию игрока. Делая ход, игрок выбирает стратегию из множества возможных своих стратегий. Выигравший игрок получает выигрыш.

Стратегия игрока – это доскональное описание действий игрока при всех возможных ситуациях, сформировавшихся в процессе игры.

Чистая стратегия – это конечный набор стратегий игрока $A_1, A_2, \dots, A_n$, которая описывает конкретное действие игрока (ход), которое он выполняет в зависимости от сведений, которые, возможно, он может получить в ходе игры.

Смешанная стратегия – это набор неотрицательных чисел $p_1, \dots, p_n$, сумма которых равна единице и которые поставлены в однозначное соответствии чистым стратегиям $A_1, A_2, \dots, A_n$ этого игрока. Стратегия A_n выбрана с вероятностью p_n.

Стратегия, при которой возможно достичь максимального ожидаемого среднего выигрыша называется оптимальной стратегией при условии, что игра многократно повторяется. При этом под «максимальным ожидаемым

средним выигрышем» понимается тот выигрыш, который дает выбор стратегии на основе принятого критерия оптимальности.

Рассмотрим конечную игру двух лиц с нулевой суммой. Очевидно, что в такой игре выигрыши одного игрока равны проигрышам другого игрока.

Матричная игра G_Z – это игра, где игроки A и B играют в игру с нулевой суммой, имея конечное число чистых стратегий $i \in \{A_1, A_2, \dots, A_n\}$ и $j \in \{B_1, B_2, \dots, B_m\}$ соответственно. Для каждых пар стратегий (i, j) задается платеж a_{ij} второго игрока первому. Матрица $Z = (a_{ij})$ представляется в виде выигрышей первого игрока и проигрышей второго.

Матричная игра, в которой оба игрока делают свой ход, сразу же распределяя выигрыши, называется одноходовой. Поскольку это игра называется игрой с нулевой суммой, то выигрыш игрока A равен a_{ij}, а выигрыш игрока B равен $(-a_{ij})$.

Таблица 1.1 – Платежная матрица Z

$A\backslash B$	B_1	B_2	…	B_m
A_1	a_{11}	a_{12}	…	a_{1m}
A_2	a_{21}	a_{22}	…	a_{2m}
…	…	…	…	…
A_n	a_{n1}	a_{n2}	…	a_{nm}

Игра заключается в том, что каждый из игроков, не имея информацию о действиях противника, делает один ход (выбирает одну из своих стратегий). Каждая выбранная пара стратегий – по одной стратегии для каждого игрока – определяет партию игры. Партия в свою очередь определяет платеж каждому игроку. Результатом партии является выигрыш одного игрока и проигрыш другого.

Партия игры состоит в том, что каждый игрок принимает одно решение, а именно делает свой выбор стратегии. Второй партией или вторым шагом игры считается повторный выбор определенных стратегий игроками в

этой игре. Возможен и третий, и четвертый шаг в игре. Иначе говоря, игра может быть многошаговой, т.е. состоять из нескольких партий.

Матричную или антагонистическую игру можно представить в виде тройки:

$$G_Z =< i, j, H >,$$

где i и j множества чистых стратегий игроков, H – функция от двух переменных $i \in A_1, \dots, A_n$, $j \in \{B_1, \dots, B_m\}$, которая называется функция выигрышей, а пара (i, j) – ситуация в чистых стратегиях.

Процесс разыгрывания между двумя игроками конечной антагонистической игры состоит в том, что игроки, независимо друг от друга, определяют для себя некоторые чистые стратегии i и j, в результате чего формируется ситуация (i, j). Затем игроки могут получить либо проигрыш, либо выигрыш $H(i, j)$.

Минимаксный подход состоит в следующем: игру G_Z можно представить следующим образом. Игрок A выбирает стратегию i (номер строки платежной матрицы), а игрок B – стратегию j (номер столбца платежной матрицы).В случае выигрыша игрок A получает сумму a_{ij} от игрока B.

В такой матричной игре целью игрока A является, достижение по возможности наибольшего выигрыша. Цель игрока B наоборот, заключается в том, чтобы игрок A по возможности получил наименьший выигрыш. Поэтому нужно строить рассуждения, основываясь на том, что поведение игроков в такой матричной игре будет осторожным и разумным.

Пусть игрок A выбрал некоторую свою стратегию i. Тогда в наихудшем случае (предполагается, что игроки ведут себя весьма осторожно и рассчитывают для себя наименее благоприятный исход, такой исход может наступить для игрока A, если выбранная им стратегия i станет известна игроку B) он получит выигрыш $\min_j a_{ij}$, поскольку второй игрок старается проиграть как можно меньше. Предвидя такие обстоятельства, игроку A

лучше всего выбрать свою стратегию i_0 так, чтобы максимизировать этот свой минимальный выигрыш:

$$\min_j a_{i_0 j} = \max_i \min_j a_{ij}$$

Следовательно, «максимин», стоящий в правой части равенства будет являться гарантированным выигрышем игрока A. Стратегия i_0 игрока A называется максиминной.

Симметричные рассуждения, проводимые за игрока B, показывают, что игрок B должен выбирать такую свою стратегию j_0, что

$$\max_i a_{i j_0} = \min_j \max_i a_{ij}$$

Здесь стоящий справа «минимакс» является тем выигрышем игрока B, больше которого он при правильных действиях противника получить не может. Стратегия j_0 игрока B называется минимаксной.

Поэтому фактический выигрыш v_1 игрока A должен при разумных действиях партнеров лежать между правыми частями $\min_j a_{i_0 j}$ и $\max_i a_{i j_0}$:

$$\max_i \min_j a_{ij} \le v_1 \le \min_j \max_i a_{ij}$$

Принцип осторожности, заставляющий игроков придерживаться максиминной и минимаксной стратегий соответственно, называют «принципом минимакса», а минимаксную и максиминную стратегии объединяют общим термином «минимаксные стратегии».

Игра G_Z называется игрой с седловой точкой, если выполняется равенство:

$$\max_i \min_j a_{ij} = \min_j \max_i a_{ij}$$

В игре с седловой точкой существуют стратегии i_0 и j_0 такие, что

$$a_{i_0 j_0} = \max_i \min_j a_{ij} = \min_j \max_i a_{ij}$$

Элемент платежной матрицы $a_{i_0 j_0}$ называется седловой точкой. Выигрыш игрока A в игре с седловой точкой равен элементу $a_{i_0 j_0}$ и называется значением игры.

В случае игры с седловой точкой игрокам выгодно придерживаться максиминной и минимаксной стратегий и невыгодно отклоняться от них, они являются оптимальными стратегиями. Таким образом, в этом случае говорят, что имеется ситуация равновесия.

Если игра G_Z не имеет седловой точки, то игроки в игре вынуждены выбирать стратегии независимо друг от друга, случайным образом, отдавая тем самым предпочтение смешанным стратегиям

$$s = \ p_1, \dots, p_n \ \text{ и } \delta = (q_1, \dots, q_m)$$

игроков A и B соответственно. В качестве платежа берется

$$sZ\delta^T = \sum_{i,j} a_{ij} p_i q_j$$

Теорема Фон Неймана имеет место равенства

$$\max_s \min_\delta sZ\delta^T = \min_\delta \max_s sZ\delta^T$$

Смешанные стратегии s_* и δ_*, для которых выполняется равенство

$$\max_s \min_\delta sZ\delta^T = \min_\delta \max_s sZ\delta^T,$$

называются оптимальными. Пара оптимальных стратегий (s_*, δ_*) называется также ситуацией равновесия в смешанных стратегиях. Число $v = s_* Z \delta_*^T$ называется значением игры G_Z. Этот максимальный выигрыш, который обеспечивает себе игрок A (и это максимальные потери, понесенные игроком B). Значение игры G_Z может быть представлено формулами:

$$v = \max_s \min_{1 \le j \le m} sZe_j^T = \min_\delta \max_{1 \le i \le n} f_i Z \delta^T,$$

где

$$e_j = \ 0, \dots, 0, 1, 0, \dots, 0 \ \in \mathbb{R}^m, f_i = (0, \dots, 0, 1, 0, \dots, 0) \in \mathbb{R}^n.$$

1.2 Методы решения матричных игр

Решением игры G_Z называется пара оптимальных стратегий. Таким образом, для разрешения конфликта, описанного как игра необходимо найти (смешанные) оптимальные стратегии игроков.

Чаще встречаются матричные игры без седловой точки. Тогда для нахождения их решений используются смешанные стратегии.

Поиск решения игр в смешанных стратегиях производится путем сведения к задаче линейного программирования и решения ее симплекс-методом.

Приведем только два метода решения матричной игры – метод доминирования и сведение к линейному программированию.

Метод доминирования. Пусть дана матрица $Z = (a_{ij})$. Строка i доминирует -ю строку, если

$$a_{ij} \geq a_{kj}$$

для всех j и

$$a_{ij} > a_{kj}$$

хотя бы для одного j.

Аналогично, столбец j доминирует -й столбец, если

$$a_{ij} \leq a_{il}$$

для всех i и

$$a_{ij} < a_{il}$$

хотя бы для одного i.

Пусть G_Z – игра с матрицей Z, строки $i_1, \dots, i_k$ которой доминируются. Тогда игрок A имеет оптимальную смешанную стратегию $s = (p_1, \dots, p_n)$, для которой $p_{i1} = \cdots = p_{i_k} = 0$. При этом оптимальная смешанная стратегия для игры G'_Z с матрицей Z', получаемой вычеркиванием из матрицы Z доминируемых строк $i_1, \dots, i_k$, является оптимальной и для первоначальной игры G_Z.

Аналогичная теорема справедлива и для доминируемых столбцов. Таким образом, учет доминирования позволяет игроку сводить игру к игре с меньшей матрицей.

В ряде игр удобно использовать в качестве решение игры следующий принцип.

Принцип доминирования. Стратегия x_i является доминирующей, если строка i доминирует остальные строки. Оптимальной в таком случае считаем доминирующую стратегию.

Метод линейного программирования. Нахождение оптимальных смешанных стратегий s, δ игры G_Z сводится к задаче линейного программирования.

К каждому элементу матрицы Z можно добавить число a так, что получается матрица Z^a и игры G_Z и G_{Z^a} имеют одинаковые оптимальные стратегии, и для значений двух игр справедливо равенство $u\ Z^a\ = u\ Z\ + a$. Ясно, что число a можно взять так, что для игры G_{Z^a}

$$v = u\ Z^a\ > 0.$$

Следовательно, без ограничения общности можем считать, что для игры G_Z ее значение $u\ Z\ > 0$.

Оптимальная смешанная стратегия $s = (p_1, \dots, p_n)$ игрока A может быть определена как решение максиминной задачи

$$v = \max_s \min_{1 \le j \le m} sZe_j^T,$$

где

$$s = \ p_1, \dots, p_n\ , e_j = \ 0, \dots, 0, 1, 0, \dots, 0\ \in \mathbb{R}^m.$$

Полагаем

$$v\ s\ = \min_j sZe_j^T.$$

Тогда

$$sZe_j^T = \sum_{i=1}^{n} a_{ij} p_i\ \partial \ge v\ s\ , j = 1, \dots, m.$$

Поэтому задачу игрока A можно представить с помощью вспомогательной величины u как задачу линейного программирования, максимизирующую величину

$$v = \max u$$

при ограничениях

$$\sum_{i=1}^{n} a_{ij}p_i - u \geq 0, j = 1, \dots, m,$$

$$\sum_{i=1}^{n} p_i = 1, \forall i \ p_i \geq 0 \ .$$

Одновременно, нахождение оптимальной смешанной стратегии δ игроком B – это минимаксная задача:

$$v = \min_{\delta} \max_{1 \leq i \leq n} f_i Z \delta^T,$$

где

$$\delta = \ q_1, \dots, q_m \ , f_i = \ 0, \dots, 0, \underset{i}{1}, 0, \dots, 0 \ .$$

Следовательно, если взять

$$v \ \delta \ = \max_{j} f_i Z \delta^T,$$

то

$$f_i Z \delta^T = \sum_{j=1}^{m} a_{ij}q_{ij} \leq v \ \delta \ , j = 1, \dots, m,$$

и тогда задачу для игрока B можно представить с помощью вспомогательной величины u как задачу линейного программирования, минимизирующую величину

$$v = \min w$$

при ограничениях

$$\sum_{j=1}^{m} a_{ij}q_j - w \leq 0, i = 1, \dots, n,$$

$$\sum_{j=1}^{m} q_j = 1, \forall j \ q_j \geq 0 \ .$$

Сделаем следующую замену переменных:

$$u_i = \frac{p_i}{u}, v_j = \frac{q_j}{w}.$$

Так как

$$\frac{1}{u} = \sum_i u_i \,, \frac{1}{w} = \sum_j u_j,$$

задачи сводятся к

$$\sum_{i=1}^{n} u_i \to \min, \tag{1}$$

$$\sum_{i=1}^{n} a_{ij} u_i \geq 1, j = 1, \dots, m,$$

$$u_i \geq 0$$

и

$$\sum_{j=1}^{n} v_j \to \max, \tag{2}$$

$$\sum_{j=1}^{m} a_{ij} v_j \leq 1, i = 1, \dots, n,$$

$$u_j \geq 0.$$

Задачи (1) и (2) являются двойственными друг другу.

Решение матричной игры G_Z эквивалентно решению пары двойственных задач линейного программирования.

Если $u^* = (u_1^*, \dots, u_n^*)$ – решение задачи линейного программирования (1), то

$$v = \max u = 1 / \sum_{i=1}^{n} u_i^*,$$

и, следовательно,

$$s^* = \left(p_1^*, \dots, p_n^*\right) = v\left(u_1^*, \dots, u_n^*\right) = vu^*$$

– оптимальная стратегия игрока A.

Аналогично, $v^* = (v_1^*, \dots, v_m^*)$ – решение задачи линейного программирования (2), то

$$v = \min w = 1/\sum_{j=1}^{m} v_j^*,$$

и, следовательно,

$$\sigma^* = (q_1^*, \dots, q_m^*) = v\,(v_1^*, \dots, v_n^*) = vv^*$$

– оптимальная стратегия игрока B.

Тогда (s^*, σ^*) – решение игры G_Z.

ГЛАВА 2 ПОСТРОЕНИЕ МАТЕМАТИЧЕСКИХ МОДЕЛЕЙ «ЗЛОУМЫШЛЕННИК – АДМИНИСТРАТОР»

2.1 Описание предметной области

Для построения математической модели охарактеризуем возможные угрозы злоумышленника и возможные средства защиты администратора.

Возможные угрозы разобьем на 4 категории:

1. атаки доступа (получение злоумышленником доступа к конфиденциальной информации);
2. атаки модификации (изменение информации или нарушение ее целостности);
3. атаки на отказ в обслуживании (делает ресурс недоступным для использования, превышая допустимые пределы пропускной способности);
4. комбинированные атаки (применение злоумышленником нескольких атак).

К атакам доступа относятся:

1. подслушивание;
2. перехват;
3. перехват сеанса.

Для подслушивания данных компьютерных сетей используется сниффер. Он является прикладной программой, перехватывающей все данные, которые передаются по определенному домену. Злоумышленник, который получил доступ к потоку передачи данных сети, имеет возможность подслушать или считать трафик.

В современной жизни программы снифферы не являются незаконными. Например, их применяют как анализатор трафика при диагностике неисправностей. Однако так как некоторые приложения при передаче данных переводят их в текстовый формат (Telnet, FTP, SMTP, POP3 и т.д.),

при помощи сниффера можно добыть, к примеру, имена пользователей и пароли, что является важной и конфиденциальной информацией.

Перехват данных является активным видом атаки, так как злоумышленник способен захватить данные в момент передачи. Довольно большую опасность создает перехват имен пользователей и паролей, так как зачастую пользователем используется одно имя и пароль в различных приложениях и системах.

В качестве перехвата сеанса, можно представить ситуацию, когда пользователь завершает начальную процедуру аутентификации при подключении, например, к почтовому серверу, злоумышленник может переводит сеанс на другой сервер, а исходному хосту отдает команду прервать соединение. В результате данной атаки исходный хост заменяется незаметно для пользователя. При получении доступа злоумышленник получает следующие возможности:

1. отправлять некорректную информацию приложению или сетевой службе;
2. пропускать через ресурс или компьютер большое количество трафика, которое приведет к остановке системы;
3. заблокировать трафик, что приведет к потере доступа пользователей, которые зарегистрированы в определенном ресурсе сети.

К атакам модификации относятся:

1. изменение данных;
2. добавление данных;
3. удаление данных.

Изменение данных, может произойти, когда злоумышленник получает доступ к ним. Даже если злоумышленник не имеет никакой информации об отправителе или получателе, он может изменить пакет данных.

Для второго типа атаки применяют добавление данных. Примером подобного типа атак является добавление информации об истории предыдущих сеансов.

При удалении данных происходит перемещение существующей информации, в недоступную область для владельца.

Атака типа отказа в обслуживании Dos (Denial-of-Service) закрывает доступ обычных пользователей к сети, превышая допустимые пределы функционирования операционной системы. На самом деле, этот тип атаки делает ресурс или компьютерную сеть недоступной обычным пользователям.

Большая часть атак типа DoS рассчитана на то, чтобы опираться на общие слабости архитектуры системы. При использовании определенных серверных приложений (например, веб- или FTP-сервер) атаки типа DoS занимают все соединения, которые доступны для данных приложений, и держат их в таком состоянии, при котором обслуживания других пользователей не допускается возможным. При применении DoS атак применяются обычные интернет-протоколы, например, TCP и ICMP (Internet Control Message Protocol).

Единовременное применение атак типа DoS на нескольких устройствах можно назвать распределенной атакой отказа в обслуживании DDoS (Distributed DoS). Такой вид атак способен нанести колоссальный ущерб организациям и пользователям, этот тип атак привлекает большое внимание администраторов безопасности.

Отказ в доступе к информации. При применении атаки типа DoS против информации, эта информация утрачивает пригодность к использованию – ее уничтожают, искажают или делают недоступной.

Отказ в доступе к приложениям. Другой тип атак типа DoS применяется к приложениям, которые обрабатывают или отображают информацию, а также к компьютерным системам, в которых обрабатываются данные приложения. Если данная атака имеет успех, то становится невозможным выполнение решения задач таким приложением.

Отказ в доступе к системе. Главной целью общего типа атак DoS является вывод системы из строя, так как после этого все приложения и информация этой системы становятся недоступными.

К комбинированным атакам относятся:

1. подмена доверенного субъекта;
2. посредничество;
3. посредничество в обмене незашифрованными ключами;
4. атака эксплойта;
5. парольные атаки;
6. угадывание ключа;
7. фишинг;
8. применение ботнетов;
9. кража конфиденциальных данных.

Чаще всего операционные системы и сети используют IP-адрес компьютера, для проверки нужный это адрес или нет. Атака типа подмена доверенного субъекта, происходит при подмене верного IP-адреса на другой, неверный адрес называется фальсификация адреса или IP-спуфинг (IP-spoofing).

Злоумышленник, применяющий IP-спуфинг, выдает себя за законного пользователя, находясь внутри или вне организации. Он может использовать IP-адрес, который находится в диапазоне санкционированных IP-адресов, или неким внешним адресом, имеющим доступ к ресурсам сети. Для того, чтобы IP-пакеты выглядели так, как будто они исходят с внутренних разрешенных адресов сети злоумышленник, использует особые программы.

Данный тип атак зачастую являются начальной точкой других атак. Наиболее распространенным примером является DoS-атака, которая начинается с постороннего адреса, которая скрывает личность злоумышленника. Чаще всего IP-спуфинг ограничивается ложной информацией или вредоносными командами, встроенными в поток данных.

Не стоит также забывать, что IP-спуфинг можно осуществить только если аутентификация производится на основе IP-адресов, следовательно, при введении других способов аутентификации (например, одноразовых паролей или иных методов криптографии) можно предотвратить IP-спуфинг.

Атака типа посредничество включает в себя подслушивание, перехват и управление передаваемой информации с помощью невидимого промежуточного узла. Не всегда представляется возможным определить с кем происходит обмен данными, если компьютеры взаимодействуют на низких уровнях сети.

Эксплойт (exploit – вредоносный код) – программа, фрагмент кода или ряд команд, использующие уязвимости программного обеспечения, применяются для того, чтобы атаковать компьютерную систему. Целью данной атаки является получение контроля над системой или нарушение функционирования данной системы.

Исходя из того, какой способ выбран, чтобы получить доступ к программному обеспечению, которое имеет некоторые уязвимости, эксплойты подразделяют на удаленные и локальные:

1. удаленный эксплойт использует сеть и уязвимости в защите без предварительного доступа к системе;
2. локальный эксплойт применяется в уязвимой системе и требует предварительный доступ к ней.

Данный тип атаки нацелен на разные компоненты системы – приложения, исполняющиеся на стороне клиента или сервера, а также модули операционной системы.

Парольные атаки – это тип атак, основной целью которых, получение логина и пароля законного пользователя. Злоумышленник может осуществлять парольные атаки, с помощью следующих методов:

1. подмена IP-адреса (IP-спуфинг);
2. подслушивание (сниффинг);
3. простой перебор (брутфорс).

Методы IP-спуфинга и сниффинга данных уже были проанализированы. Данные способы используются, чтобы получить доступ к логину и паролю пользователя, если они передаются в текстовом формате через незащищенный канал.

Зачастую злоумышленники подбирают логин или пароль с использованием многочисленных попыток доступа. Данный подход называется «атака полного перебора» (Brute Force Attack). Для этого применяется особая программа, с помощью которой производятся попытки получить доступ к серверу или другому ресурсу общего пользования. Если у злоумышленника получается подобрать пароль, то ему также удается заполучить доступ к ресурсам как обычному пользователю. Если у пользователя есть особые привилегии доступа, то злоумышленник способен создать так называемый «проход» для возможного доступа. В таком случае, даже если пользователь сменит логин и пароль, у злоумышленника останется работающий «проход» в систему.

Атака типа угадывание ключа, заключается в том, чтобы что узнать значение ключа доступа. Криптографическим ключом называется код или число, которое необходимо, чтобы расшифровать защищенную информацию. Злоумышленник воспользуется данным ключом для того, чтобы получить доступ к защищенным передаваемым данным без ведома получателя и отправителя. При помощи данного ключа он может расшифровать и изменять данные.

Фишинг – это новый вид интернет-мошенничества, который появился относительно недавно. Его целью является получение идентификационных данных пользователей. К атакам типа фишинг относят кражи паролей, PIN-кодов и другой конфиденциальной информации, которая дает доступ к средствам пользователя. Данный тип атак использует легковерность и невнимательность пользователей Интернета, а не уязвимости программного обеспечения. Термин phishing, который созвучен со словом fishing (рыбная ловля), можно расшифровать как password harvesting fishing, что значит

выуживание пароля. Это сравнение не случайно – злоумышленник оставляет приманку в Интернете и «вылавливает» пользователей Интернета, которые клюют на приманку.

Злоумышленник создает очень похожую копию выбранного сайта, а затем по почте рассылается спам-письмо, составленное так, чтобы оно было максимально похоже на настоящее письмо от банка. В письме он использует атрибуты банка, имена и фамилии реальных руководителей банка. В этом письме сообщают о том, что пользователь должен изменить или подтвердить свои данные, так как в системе интернет-банкинга произошли изменения программного обеспечения. В качестве причины указывается выход из строя программного обеспечения или же атака злоумышленников. Цель данных писем – создать условия, при которых пользователь нажмет на определенную ссылку, а затем ввести данные (номер счета, пароль, PIN-код). Зайдя на ложный сайт, пользователь вводит конфиденциальные данные, и злоумышленники получают доступ к его почтовому ящику, а в худшем случае к электронному счету.

Мошенники зачастую применяют троянские программы. В этом случае задача злоумышленника упрощается – когда пользователь заходит на ложный сайт, он «цепляет» программу, которая самостоятельно разыщет на компьютере жертвы все необходимые данные. Помимо троянских программ злоумышленники также используют кейлоггеры. С ложных сайтов загружаются шпионские программы, которые отслеживают нажатие клавиш. При таком подходе не нужно связываться с клиентами банка или компании. Именно поэтому злоумышленники подделывают сайты общего назначения (новостная лента, поисковая система и т.д.).

Ботнет – компьютерная сеть, которая заражена вредоносной программой Backdoor (т.е. зомби-сеть). Данный тип атаки позволяет злоумышленникам при помощи удаленного доступа управлять зараженными частями компьютеров, которые входят в сеть, или даже всей сетью, без ведома пользователя. Подобные программы называют ботами.

Ботнеты обладают высоко вычислительной способностью и являются сильным кибероружием. Хозяин зараженной сети может из любой точки мира управлять зараженными машинами сети, а благодаря тому, как организован Интернет, это можно делать анонимно.

В качестве стратегий администратора были рассмотрены две категории защиты информации:

1. программные продукты, включающие в себя особые программы, комплексы программ и встроенные механизмы операционной системы, которые способны защитить информацию;
2. криптографические средства – то есть особые алгоритмы и математические способы защиты информации, которые используют разнообразные методы шифровки данных.

Среди программных средств можно выделить следующие:

1. антивирусное программное обеспечение;
2. межсетевые экраны (например, брандмауэр, файрвол, и прочие);
3. специализированные программные продукты, препятствующие несанкционированному доступу к информации.

Антивирусное программное обеспечение – это программы, которые предназначены для того, чтобы обнаружить компьютерный вирус, лечить или удалить инфицированные файлы, а также проводить профилактику в целях предотвращения заражения файлов или частей операционной системы вредоносными программами. Примером антивирусного программного обеспечения могут служить AIDSTEST, AVAST, AVP, DrWeb и прочие.

Межсетевые экраны также называются брандмауэрами или файрволами. Это своего рода промежуточный сервер между локальной сетью и глобальной. Он инспектирует весь проходящий трафик сетевого и транспортного уровней. Использование межсетевых экранов позволяет сократить вероятность неразрешенного доступа вне корпоративной сети, однако такая вероятность не может быть устранена полностью. Более надежной версией данного метода является способ маскарада. При

использовании данного метода весь трафик локальной сети через firewall-сервер пересылается, что практически делает локальную сеть невидимой;

Специализированные программные продукты, препятствующие несанкционированному доступу к информации. У этой стратегия защиты фактически есть лучшие возможности и характеристики, чем у встроенных средств. В современной жизни на рынке программных средств представлен большой выбор особых программ, предназначенных для того, чтобы защищать папки и файлы на компьютере, контролировать выполнение пользователями специальных правил безопасности, а также выявлять и пресекать попытки неразрешенного доступа к тем конфиденциальным данным, которые хранятся на персональном компьютере пользователя; наблюдать за теми действиями, которые происходят на работающем автономном режиме или в локальной сети компьютере.

Что касается криптографической защиты информации, то она предназначена для преобразования исходных данных с целью сделать ее недоступной к использованию и ознакомлению посторонними лицами, которые не имеют на эти права и полномочия.

Существуют разные подходы к классификации методов данного способа защиты информации. К примеру, по характеру воздействия на исходную информацию можно выделить четыре группы преобразования данных:

1. шифрование;
2. стеганография;
3. кодирование;
4. сжатие.

Шифрование. Данный процесс заключается в том, чтобы произвести обратимые математические, логические, комбинаторные и другие преобразования исходной информации. В результате этих преобразований зашифрованная информация выглядит как хаотический набор символов, букв, цифр и прочих. Для того, чтобы зашифровать информацию используют

алгоритм преобразования и определенный ключ. Чаще всего, для особого метода шифрования есть неизменный алгоритм. Исходной информацией для алгоритма служат данные, которые нужно зашифровать и непосредственно ключа шифрования. В ключе содержится управляющая информация, определяющая преобразование на определенном этапе алгоритма и величины операндов, которые используются для осуществления алгоритмов шифрования.

Стенография. Данный тип преобразования информации отличается от остальных тем, что он позволяет скрыть как смысл информации, так и сам факт ее передачи и хранения. У данного типа преобразования информации есть большой потенциал, несмотря на то, что в современных компьютерных системах ее практическое использование только начинается. В качестве базы методов стенографии находится маскировка среди открытых файлов закрытой информации. В основе всех методов стеганографии лежит маскировка скрытой информации между открытыми файлами. После начала обработки файлов мультимедиа в компьютерной системе перед стеганографией открылись очень широкие возможности.

Скрыто передать информацию можно несколькими способами. Например, в основе одного из самых простых методов лежит скрытие файлов во время работы в операционной системе MS DOS. За каждым текстовым файлом закрепляется скрытый файл, записанный в двоичном коде, чей объем намного меньше, чем у текстового файла. Эти файлы помечаются буквами EOF (сочетание клавиш Control + Z). Стандартные средства операционной системы прекращают считывание при достижении метки EOF, и, следовательно, скрытый файл остается для нее недоступным. Что касается двоичных файлов, то для них никаких меток не существует. Конец двоичного файла определяет обработка атрибутов, в том числе и длина файла, выраженная в байтах. При открытии файла как двоичного, то можно поучить доступ к закрытому файлу. Более того, скрытый файл можно зашифровать. В

таком случае при случайном открытии этого файла информация будет выглядеть как сбой в работе системы.

Графическую, а также звуковую информацию можно представить в числовом виде. К примеру, наименьший элемент графических файлов может быть закодирован одним байтом. Исходя из алгоритма криптографических преобразований, к младшим разрядам специальных байтов изображения могут быть помещены биты из скрытого изображения. Человеческий глаз не сможет уловить различия между исходным и полученным изображением, если будет правильно подобран алгоритм преобразования и изображение, на фоне которого будет размещен скрытый файл. Более того, даже специальные программы выявляют скрытую информацию крайне сложно. Для внедрения скрытого файла наиболее подходящими являются изображение местности, такие как фотографии, сделанные со спутника или самолета. Благодаря данному способу можно маскировать текст, зашифрованные сообщения, изображения и прочие. При комплексном применении стеганографии и шифрования можно много во много раз повысить сложность обнаружения и скрытых файлов и раскрытия информации, содержащихся в них.

Кодирование. Под кодирование понимается процесс замены конструкций, имеющих смысл (слова, предложения и т.д.) на код. В качестве кодов могут использоваться буквы, цифры или буквы и цифры. Для кодирования и обратного преобразования применяются особые словари или таблицы.

Данный тип скрытой передачи информации лучше всего применять в системах, в которых набор смысловых конструкций ограничен. Данный вид преобразования можно применить, таких как командная линия АСУ. Главными недостатками кодирования считается то, что кодировочные таблицы необходимо хранить, распространять, но в то же время часто менять во избежание вероятного раскрытия кода при обработке перехваченных сообщений.

Сжатие. Стоит сказать, что сжатие информации можно назвать методом криптографического преобразования, только если оговорить некоторые особенности. Основная цель сжатия информации – это уменьшение объема информации. Однако, сжатую информацию невозможно будет прочитать или использовать, не преобразовав ее обратно. Учитывая то, насколько доступны средства сжатия информации и средства ее обратного преобразования, то этот метод нельзя считать надежным средством криптографического преобразования. Даже статистические методы обработки способны раскрыть алгоритмы сжатия. По этой причине конфиденциальные данные шифруются после сжатия.

2.2 Построение математической модели «злоумышленник – администратор»

Исходя из предметной области сформулируем 4 стратегии злоумышленника и 4 стратегии администратора.

Стратегии злоумышленника $З \in \{З_1, З_2, З_3, З_4\}$:

$З_1$– Заражение системы вирусами

$З_2$– Отказ в обслуживании системы

$З_3$– Использование фишинговых сайтов

$З_4$– Использование шпинских программ

Стратегии администратора $А \in \{А_1, А_2, А_3, А_4\}$:

$А_1$– стандартный межсетевой экран операционной системы Windows 2007;

$А_2$– антивирусная программа DrWeb;

$А_3$– антивирусная программа AVAST.

После определения стратегий математическая модель матричной игры принимает вид:

Таблица 2.1 – Платежная матрица

З/А	$А_1$	$А_2$	$А_3$
$З_1$	a_{11}	a_{12}	a_{13}
$З_2$	a_{21}	a_{22}	a_{23}
$З_3$	a_{31}	a_{32}	a_{33}
$З_4$	a_{41}	a_{42}	a_{43}

В соответствии каждой паре стратегий ($З_i$,$А_1$) ставится значении платежа матрицы a_{ij}, то есть выигрыш игрока З и проигрыш $(-a_{ij})$ игрока А.

Для того чтобы найти решение матричной игры, игрокам необходимо выбрать такую стратегию, которая удовлетворяет условию оптимальности. Это означает, что один из игроков должен получать максимальный выигрыш, при условии, что второй придерживается своей стратегии. В этот момент второй игрок должен получать минимальный проигрыш, если первый придерживается своей стратегии.

В игре злоумышленника и администратора естественно предполагать, что оба игрока ведут себя разумно с точки зрения своих интересов.

Данная игра является конченой, так как наборы $З_i$и $А_j$ $(i = 1,2,3,4,$ $j = 1,2,3)$ стратегий у игроков ограничены и матрица имеет размерность 4×3.

Предположим, что значения платежной матрицы a_{ij} известны или были заданы изначально, для любой пары стратегий $(З_i, А_j)$.

Определим в такой матрице наилучшую среди стратегий $З_1, З_2, З_3, З_4$. Выбирая стратегию $З_i$ игрок З должен рассчитывать, что игрок А ответит на нее стратегией $А_j$, которая принесет наименьший выигрыш игроку З.

Введем обозначение наименьшего выигрыша α_i игрока З при выборе им стратегии $З_i$ для всех возможны стратегий игрока А (наименьшее число в i-й строке платежной матрицы), то есть

$$\min_{j=1} a_{ij} = \alpha_i.$$

Среди всех чисел α_i $(i = 1,2,3,4)$ выберем наибольшее: $\alpha = \max_{i=1} \alpha_i$. Назовем α нижней ценой игры или максимальным выигрышем (максимином). Это гарантированный выигрыш игрока З при любой стратегии игрока А. Следовательно,

$$\alpha = \max_{i=1} \min_{j=1} a_{ij}.$$

Стратегия, соответствующая максимину, называется максиминной стратегией. Игрок А заинтересован в том, чтобы уменьшить выигрыш игрока З; выбирая стратегию A_j он учитывает максимально возможный при этом выигрыш для З. Обозначим

$$\beta_j = \max_{i=1} a_{ij}.$$

Среди всех чисел β_j $(j = 1,2,3)$ выберем наименьшее: $\beta = \min_{j=1} \beta_i$. Назовем β верхней ценой игры или минимаксным выигрышем (минимакс). Это гарантированный проигрыш игрока А при любой стратегии игрока З. Следовательно,

$$\beta = \min_{j=1} \max_{i=1} a_{ij}.$$

Стратегия, соответствующая минимаксу, называется минимаксной стратегией.

Такой принцип, диктует игрокам вести осторожную игру, предполагая, что соперник придерживается минимаксной и максиминной стратегий (минимаксный принцип). Этот принцип следует из разумного предположения, что каждый игрок стремится достичь цели, противоположной цели противника.

Если верхняя и нижняя цены игры совпадают $\alpha = \beta = \nu$, то имеется решение в чистых стратегиях – чистая цена игры.

Минимаксные стратегии, соответствующие цене игры, являются оптимальными стратегиями, а их совокупность – оптимальным решением

игры. В этом случае игрок З получает максимальный гарантированный (не зависящий от выбора стратегий игрока А) выигрыш ν.

Пара чистых стратегий $З_i$ и $А_j$ дает оптимальное решение игры тогда и только тогда, когда соответствующий ей элемент a_{ij} является одновременно наибольшим в своем столбце и наименьшим в своей строке. Если существует такая ситуация, то она называется седловой точкой.

Если игра не имеет седловой точки, то приминение чистых стратегий не дает оптимального решения игры. Так, как $\alpha \neq \beta$, седловая точка отсутствует. В таком случае можно получить оптимальное решение, случайным образом чередуя чистые стратегии.

Смешанной стратегией $S_З$ игрока З называется применение чистых стратегий $З_1, З_2, З_3, З_4$ с вероятностями $p_1, p_2, \dots, p_4$ причем таких что сумма вероятностей равна 1:

$$\sum_{i=1}^{4} p_i = 1.$$

Смешанные стратегии игрока З записываются в виде матрицы $S_З$:

Таблица 2.2 – Матрица смешанных стратегий игрока З

$З_1$	$З_2$	$З_3$	$З_4$
p_1	p_2	p_3	p_4

или в виде строки $S_З = (p_1, p_2, p_3, p_4)$.

Аналогично смешанные стратеги игрока А обозначаются S_A:

Таблица 2.3 – Матрица смешанных стратегий игрока А

$А_1$	$А_2$	$А_3$
q_1	q_2	q_3

или в виде строки $S_A = (q_1, q_2, q_3)$, где сумма вероятностей появления стратегий равна 1:

$$\sum_{j=1}^{3} q_j = 1.$$

Чистые стратегии можно считать частным случаем смешанных и задавать строкой, в которой 1 соответствует чистой стратегии.

На основании принципа минимакса определяется оптимальное решение игры: это пара оптимальных стратегий S_3^*, S_A^* в общем случае смешанных, обладающих следующим свойством: если один из игроков придерживается своей оптимальной стратегии, то другому не может быть выгодно отступать от своей цели. Выигрыш, соответствующий оптимальному решению, называется ценой игры ν. Цена игры удовлетворяет неравенству:

$$\alpha \leq \nu \leq \beta,$$

где α и β – нижняя и верхняя цены игры.

По основной теореме теории игр (теорема фон Неймана) каждая конечная игра имеет по крайней мере одно оптимальное решение, возможно, среди смешанных стратегий.

Пусть $S_3^* = (p_1^*, p_2^*, p_3^*, p_4^*)$ и $S_A^* = (q_1^*, q_2^*, q_3^*)$ – пара оптимальных стратегий. Если чистая стратегия входит в оптимальную смешанную стратегию с отличной от нуля вероятностью, то она называется активной.

Теорема об активных стратегиях: если один из игроков придерживается своей оптимальной смешанной стратегии, то выигрыш остается неизменным и равным цене игры ν, если второй игрок не выходит за пределы своих активных стратегий.

Если в данной игре отсутствует седловая точка, тогда в соответствии с основной теоремой теории игр оптимальное решение существует и определяется парой смешанных стратегий $S_3^* = (p_1^*, p_2^*, p_3^*, p_4^*)$ и $S_A^* = (q_1^*, q_2^*, q_3^*)$.

Для нахождения цены игры смешанных стратегий данная задача сводится к задаче линейного программирования.

2.3 Построение математической модели выбора оптимального набора средств защиты

Для поиска наиболее оптимальных стратегий защиты информационных ресурсов можно провести матричную игру злоумышленник – администратор. Нанесение злоумышленником ущерба обычно является скорее следствием его действий, чем самой целью.

В действительности при атаке он может преследовать какие-то свои цели, порой известные лишь ему. Поскольку целью данной работы являлось определение администратором оптимальной стратегии защиты, а цели атакующих злоумышленников были неважны, то можно предположить, что злоумышленник ведет себя разумно и пытается нанести как можно больший ущерб атакуемой компьютерной системе.

Предполагаем, что главной задачей администратора является выбрать такие средства защиты, чтобы при любых предполагаемых стратегиях злоумышленника, защита являлась оптимальной. Под оптимальной защитой понимается такая защита, которая максимально снизит ущерб от атаки.

Для того чтобы принять решение администратору в пользу определенной защиты информационной системы, можно использовать различные критерии принятия решения в условиях неопределенности.

В данной работе были рассмотрены три критерия принятия решения:

1. критерий Вальда;
2. критерий Сэдвиджа;
3. критерий Гурвица.

Критерий Вальда является критерием крайнего пессимизма и помогает определить оптимальную стратегию, которая в наихудших условиях гарантирует максимальный выигрыш. Следовательно, с помощью данного критерия администратор сможет выбрать такую стратегию, при которой наименьший выигрыш является наибольшим среди наименьших выигрышей всех стратегий. Формула критерия Вальда имеет следующий вид:

$$W_{j0} = \min_j \max_i a_{ij}.$$

Такой критерий уместен тогда, когда администратор не столько хочет выиграть (минимальный ущерб информационной системе), сколько не проиграть (ущерб информационной системе не был самым максимальным из возможных).

Критерий Сэдвиджа, как и критерий Вальда, является критерием крайнего пессимизма, а также предполагает самые неблагоприятные для игрока условия. С помощью этого критерия администратор сможет выбрать такую стратегию, при которой наибольший выигрыш является наименьшим среди наибольших выигрышей всех стратегий. Формула Сэдвиджа представляется в виде:

$$W_{j0} = \max_j \min_i a_{ij}.$$

Такой критерий способен предположить наихудшее развитие событий и выбрать оптимальную стратегию, чтобы максимально снизить риски. Администратор, применив данную стратегию, сможет спрогнозировать максимальный ущерб от действий злоумышленника и выбрать такую стратегию, которая будет самой успешной для отражения данной атаки (ущерб системе будет минимальным).

Критерий Гурвица является критерием «оптимизма-пессимизма». Позволяет выбрать некоторый средний результат, находящимся в поле между значениями по критериям «минимакса» и «максимина». Согласно критерию Гурвица, формула имеет вид:

$$W_{j0} = c \max_j a_{ij} + (1 - c) \min_j a_{ij},$$

где $c \in [0,1]$ – коэффициент пессимизма. Крайнему пессимизму $(c = 1)$ можно противопоставить крайний оптимизм $(c = 0)$, когда ставка делается на самый большой возможный выигрыш (наименьший ущерб системе). Данный коэффициент выбирается из субъективных соображений администратора.

Из рассмотренных критериев был выбран критерий Сэдвиджа, который больше подходит, для моделирования ситуации, когда администратору важно предположить худшую ситуацию. Этот вариант наиболее благоприятен для создания азартного злоумышленника, предполагая, что он выберет лучшую атаку на информационную систему.

При таком предположении можно построить матрицу игры двух лиц с нулевой суммой, где коэффициенты матрицы, это предполагаемый ущерб системе от атаки злоумышленника.

В качестве стратегий злоумышленника будем понимать строки $З_i \in \{З_1, \dots, З_n\}$ некоторой матрицы, а в качестве стратегий администратора безопасности – ее столбцы $A_j \in \{A_1, \dots, A_m\}$ К стратегиям злоумышленника можно отнести различные виды информационных атак, а к стратегиям администратора – различные средства защиты информационной системы.

Для проведения на компьютере игры надо знать результаты игры при каждой паре стратегий $(З_i, A_j)$ и вероятности применения атак злоумышленником $p(З_i)$ при выбранной стратегии A_j.

Вероятности $p(З_i)$ могут быть определены эмпирическими методами, например, с помощью систем обнаружения атак (IDS).

Предполагаем, что вероятности применения атаки уже известны или можно предположить, что они все равновероятностные $p(З_i) = 1/n$.

Таблица 2.4 – Матричная игра

		A_1	A_2	…	A_m
$З_1$	$p(З_1)$	a_{11}	a_{12}	…	a_{1m}
$З_2$	$p(З_2)$	a_{21}	a_{22}	…	a_{2m}
…	…	…	…	…	…
$З_n$	$p(З_n)$	a_{n1}	a_{n2}	…	a_{nm}

Целью игрока З (злоумышленника) в матричной игре является, естественно, получение, по возможности, большего выигрыша. Цель же игрока А (администратора) состоит в том, чтобы дать злоумышленнику возможный меньший выигрыш. Поэтому для получения администратором гарантированного выигрыша применим критерий Сэдвиджа.

Пусть игрок З выбирает некоторую свою стратегию $З_i$. Тогда в наихудшем случае (а в теории игр игроки предполагаются весьма осторожными и рассчитывают на наименее благоприятный для себя поворот событий; такое наименее благоприятное для игрока З положение дел может наступить, например, в том случае, когда стратегия $З_i$ станет известной игроку А) он получит выигрыш

$$\min_j a_{ij}.$$

Предвидя такую возможность, игрок З должен выбрать свою стратегию $З_{i_0}$ так, чтобы максимизировать свой минимальный выигрыш:

$$\min_j a_{i_0 j} = \max_i \min_j a_{ij}. \quad (3)$$

Значит, стоящий в правой части написанного равенства «максимин» является гарантированным выигрышем игрока З. Симметричные рассуждения, проводимые за игрока A, показывают, что игрок A должен выбирать такую свою стратегию y_{j_0}, что

$$\min_i a_{i j_0} = \min_j \max_i a_{ij}. \quad (4)$$

Здесь стоящий справа «минимакс» является тем выигрышем игрока З, больше которого он при правильных действиях противника получить не может. Поэтому фактический выигрыш игрока З должен при разумных действиях партнеров лежать между правыми частями формул (3) и (4). Выигрыш игрока З называется значением игры и равен элементу матрицы $a_{i_0 j_0}$.

ГЛАВА 3 РЕАЛИЗАЦИЯ ПРОГРАММЫ ОПТИМИЗАЦИИ ВЫБОРА

В данной работе был рассмотрен процесс создания программного продукта, который позволит администратору выбрать оптимальную стратегию для защиты информации. Данная программа поможет службе безопасности выбрать оптимальные средства защиты с помощью игрового подхода. Продукт должен соответствовать всем основным требованиям, которые будут рассмотрены ниже на стадии анализа.

В качестве подхода к проектированию программного обеспечения был выбран классический каскадный подход – подход, основанный на модели водопада (рисунок 3.1).

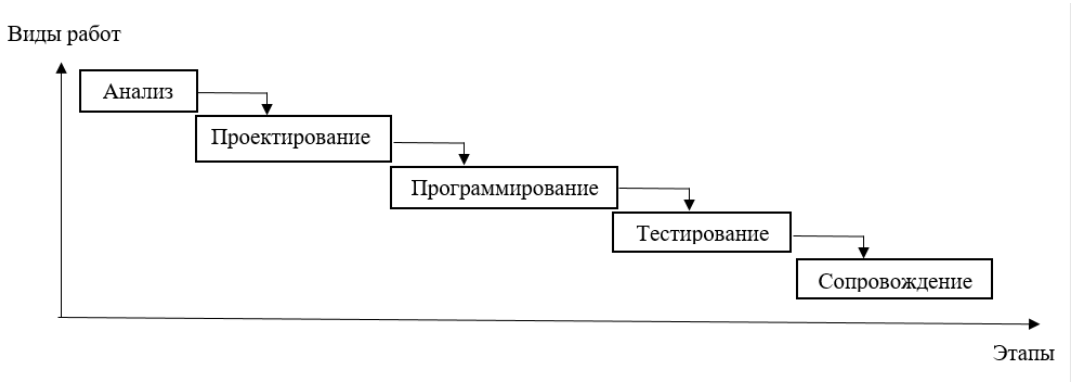


Рисунок 3.1 – Классический каскадный подход

Такой подход к ведению жизненного цикла проекта встречается довольно редко. Его можно рассматривать как отправную точку для создания программы. Принцип чистого каскадного подхода таков, что переход к следующей стадии разработки осуществляется только после того, когда завершена работа с текущей стадией. Возвраты к стадиям, которые считаются пройденными – не предусмотрены.

3.1 Анализ требований

Процесс выработки требований – это создание спецификаций к программному обеспечению, которая проводится в начале проекта, на стадии анализа.

Основными функциональными требованиями программного продукта являются: программа должна на основе входных данных выбрать самую оптимальную (подходящую) стратегию для защиты.

К нефункциональным требованиям относится: разработка программы с помощью объектно-ориентированного подхода, с использованием языка программирования Java.

После этого определим приоритеты требований с помощью набора критериев MoSCoW:

1. выбор оптимальной стратегии для защиты информации;
2. расчет максимального ущерба системе;
3. расчет оптимальной стратегии с помощью совместимости программ защиты;
4. вывод результата в виде графика лестничной функции.

Для построения модели прецедентов использовался Visual Paradigm for UML, Version 8.0 – это набор средств для моделирования информационных систем и бизнес-процессов, генерации кода на базе построенных моделей, проектирования БД и решения многих других задач.

Определим основные лица или актеров, взаимодействующих с программным продуктом и построим модель прецедентов с помощью Use Case диаграммы, которая изображена на рисунке 3.2.

Рисунок 3.2 – Диаграмма прецедентов

На диаграмме представлены: граница системы (прямоугольник), основные актеры (администратор) и прецеденты с которыми взаимодействуют актеры. Спецификации прецедентов изображены в виде таблицы**.**

Таблица 3.1 – Спецификация прецедентов

Прецедент: выбрать оптимальную стратегию защиты
ID: 2
Краткое описание: выбрать оптимальный набор средств защиты, в зависимости от введенных данных
Главный актер: Администратор
Второстепенные актеры: нет
Предусловия: нет
Основной поток: 1. прецедент запускается, когда актер ввел данные, выбрал способ расчета (оптимальная стратегия) и нажал кнопку рассчитать. 2. программа преобразует, данные и выполняет операции с ними для выбора средств оптимальной защиты.

Продолжение таблицы 3.1

3. программа выводит результат в виде названия средства защиты
Постусловия: нет
Альтернативные потоки: нет

3.2 Проектирование программы

Проанализировав требования к программе, необходимо спроектировать ее первую версию. В данном разделе аналитическая модель перетекает в проектную.

Необходимо определить в полном объеме, как будут реализовываться функциональные артефакты (прецеденты) программы. Определенные в общих чертах артефакты являются входными данными для более детального проектирования, в процессе которого происходит их полное описание.

Входные данными программы являются предполагаемые стратегии злоумышленника $x_i \in \{x_1, \dots, x_n\}$ и стратегии администратора $y_j \in \{y_1, \dots, y_m\}$:

1. названия стратегий x_i и y_j;
2. предполагаемая вероятность успеха атаки $p(x_i)$;
3. предполагаемый ущерб от атаки в виде денежных единиц;
4. предполагаемый ущерб от защиты в виде стоимости, выраженной в виде денежных единиц.

Выходными данными является результат в виде оптимального средства защиты. Ниже изображена блок-схема программы.

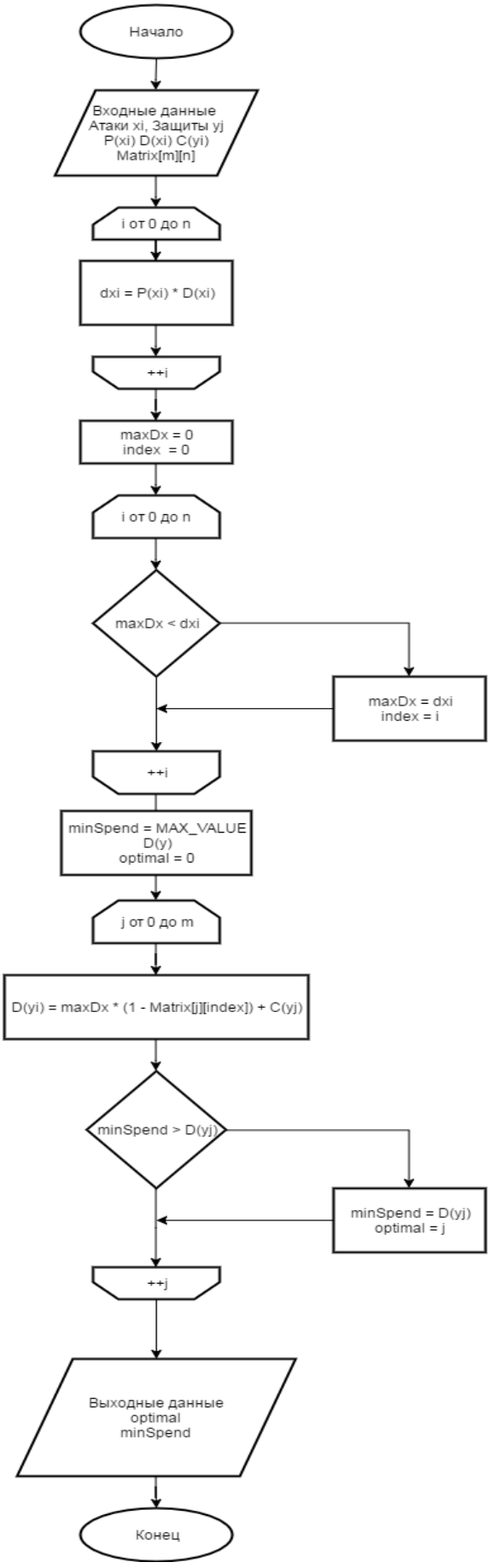


Рисунок 3.3 – Блок-схема программы

Для более четкого представления конечного результата, был от руки нарисован (рисунок 3.4) первый макет программы.

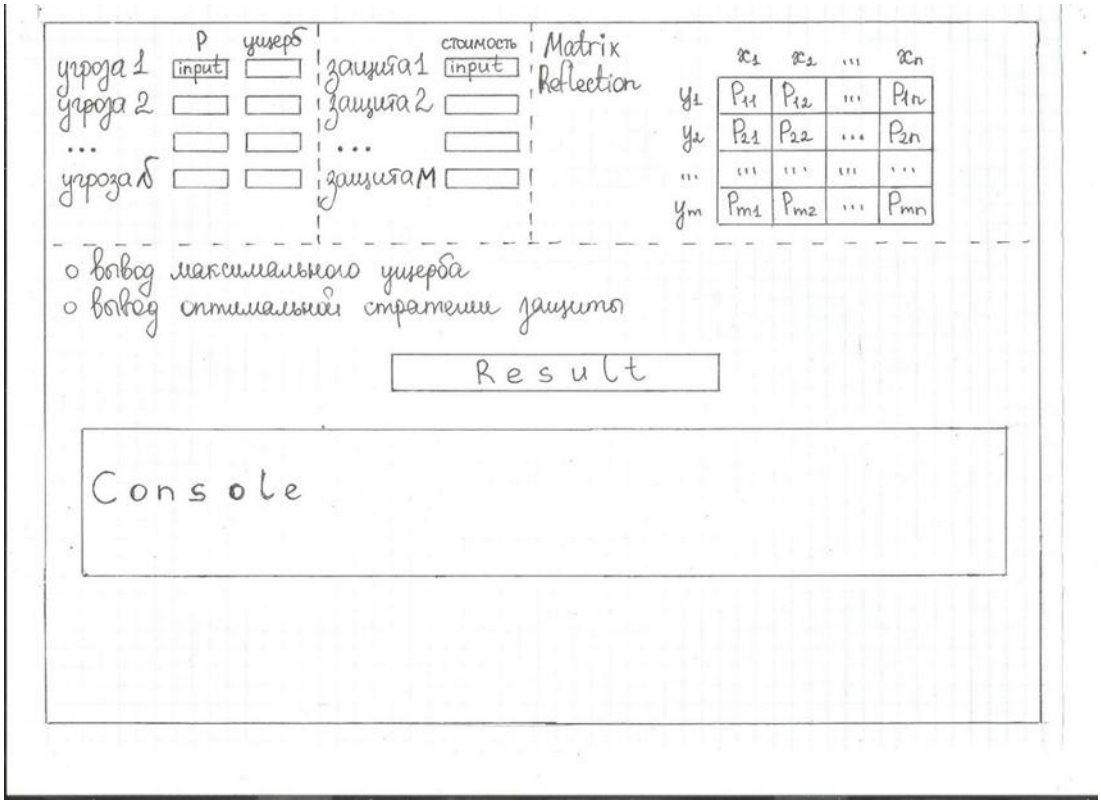


Рисунок 3.4 – Макет разрабатываемой программы

Макет может помочь продумать начальную архитектуру программы и на его основе спроектировать «уточняющую» диаграмму классов.

Создание диаграммы классов, также реализовывалось в Visual Paradigm с помощью средств проектирования диаграмм классов. Конечный результат представлен в «Приложении А» данной работы.

Работа основных классов программы будет рассмотрена в следующей подглаве.

3.3 Разработка кода программы

Разработка кода программы осуществлялась с помощью объектно-ориентированного языка Java. Достоинством языка, можно отметить механизм компиляции в промежуточные байт-коды, которые исполняются на виртуальной машине JVM. Это означает, что разрабатываемая программа может исполняться на любой машине, где есть JVM.

Разработка кода программы велась в Eclipse IDE for Java Developers Version: Neon.3. С помощью средств Visual Paradigm можно сгенерировать программный код Java. Код программы для удобства можно сразу экспортировать в файлы с нужным расширением и открыть их с помощью Eclipse (рисунок 3.5).

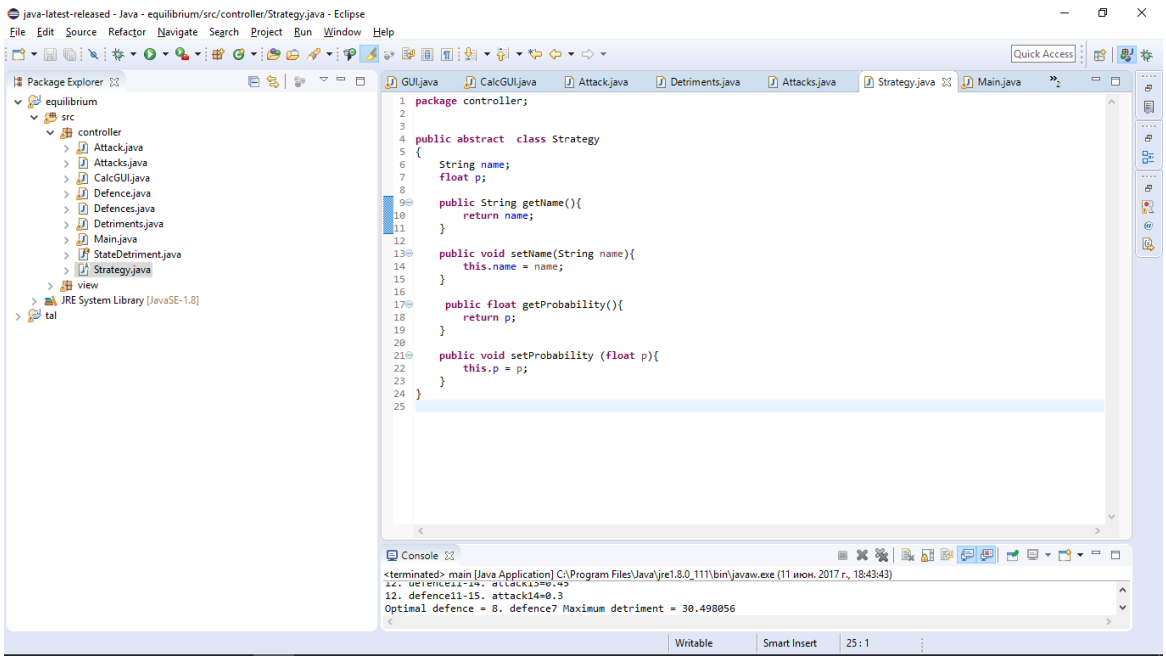


Рисунок 3.5 – Программа открыта в Eclipse

Для различных операций с входными данными были спроектированы классы с некоторыми параметрами и методами.

Для удобства была разделена логика программы и ее представление в виде отдельных пакетов. Диаграмма пакетов показа ниже.

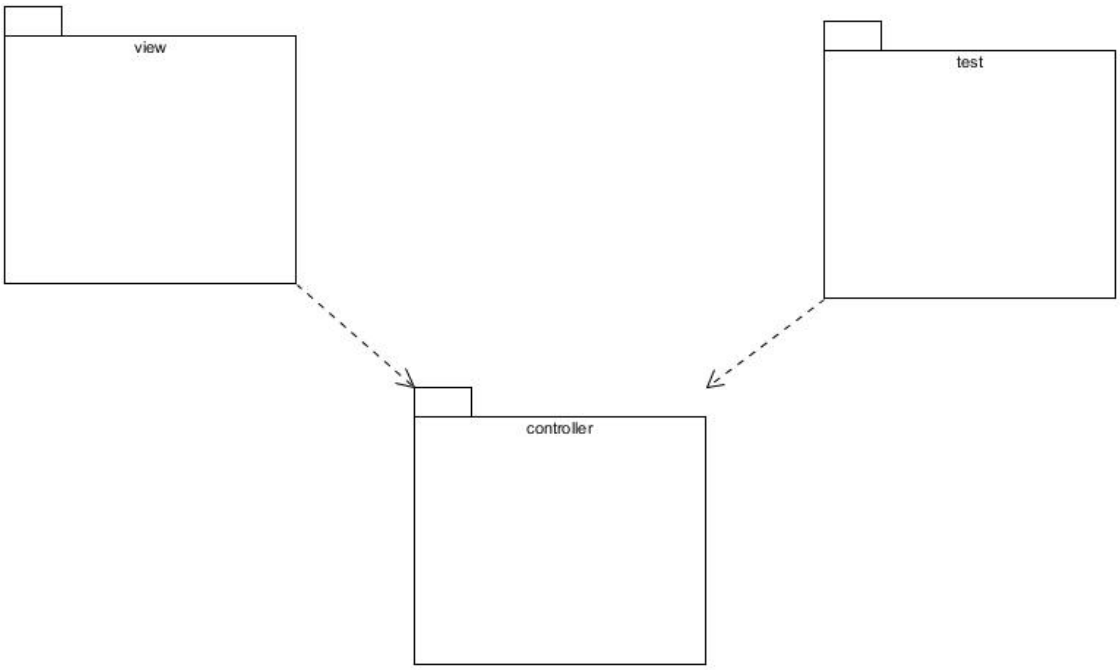


Рисунок 3.6 – Диаграмма пакетов

Основные классы программы:

1. Attacks и Defences – классы представляют собой набор стратегий x_i злоумышленника и администратора y_j;

2. Strategy – это абстрактный класс, в котором выносится общее для классов Attack и Defence;

3. Attack и Defence – классы, описывающие одну единицу из возможных стратегий злоумышленника $x_i \in \{x_1, \dots, x_n\}$ и администратора $y_j \in \{y_1, \dots, y_m\}$. Расширяют класс Strategy;

4. Detriments – класс, представляющий основную логику задачи линейного программирования. С помощью данного класса рассчитывается результат в виде оптимальной защиты;

5. GUI – класс, который описывает графический интерфейс программы. С помощью данного класса пользователь программы может задавать значения входных данных, а также видеть результат программы;

6. CalcGUI – класс хранит и обрабатывает (преобразует) входные данные, полученные из GUI. С помощью данного класса реализована

основная последовательность действий программы. Также данный класс является слушателем событий и реализует интерфейс ActionListener;

7. JUnit – класс для модульного тестирования программы. Использует библиотеку тестирования JUnit;

8. Main – исполняющий класс программы с методом main().

Полный листинг кода программы представлен в «Приложении Б».

Разработанная программа представлена на рисунке 3.7.

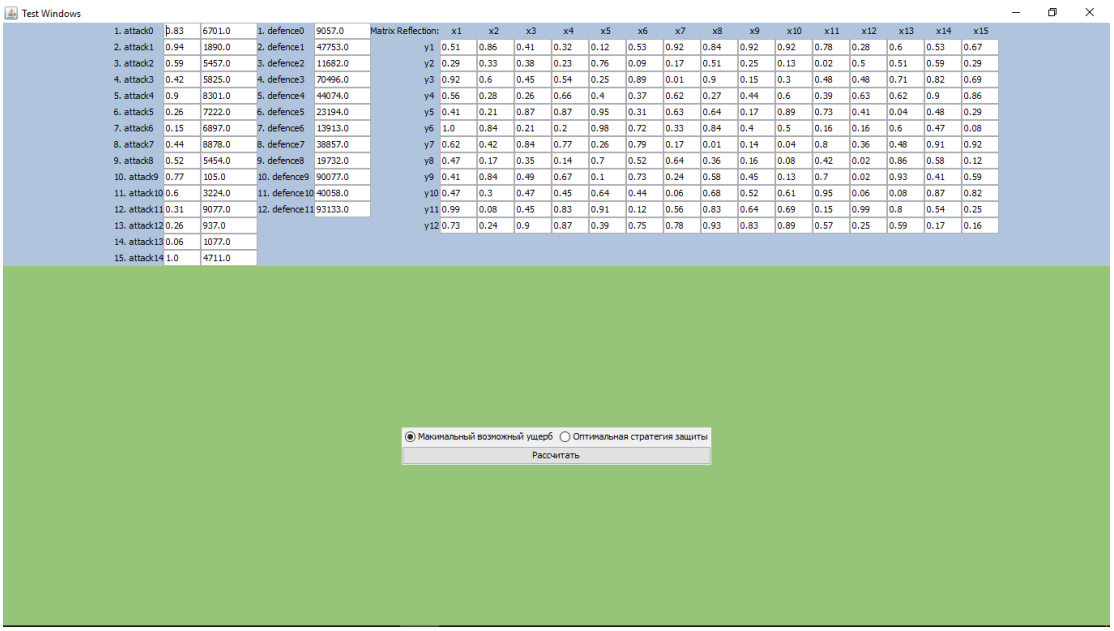


Рисунок 3.7 – Окно программы

3.4 Тестирование и отладка программы

Тестирование программы должно происходить параллельно с созданием программного кода. Тестирование программы делится на различные уровни:

1. модульное тестирование – тестирование отдельных участков кода;

2. интеграционной тестирование – совместное тестирование компонентов программы;

3. системное тестирование – тестирование программы в целом;

4. итоговое тестирование первой рабочей версии программы.

В данной работе был рассмотрен только первый уровень тестирования программы.

Для модульного тестирования программы был выбран JUnit фреймворк, который позволяет производить тестирование отдельных методов или классов. Полученный опыт при работе с модульным тестированием важен в разработке концепций тестирования программного обеспечения.

Программная среда Eclipse имеет различные средства для отладки кода и уже поставляется вместе с фреймворком JUnit. Реализация простого теста представлена на рисунке 3.8.

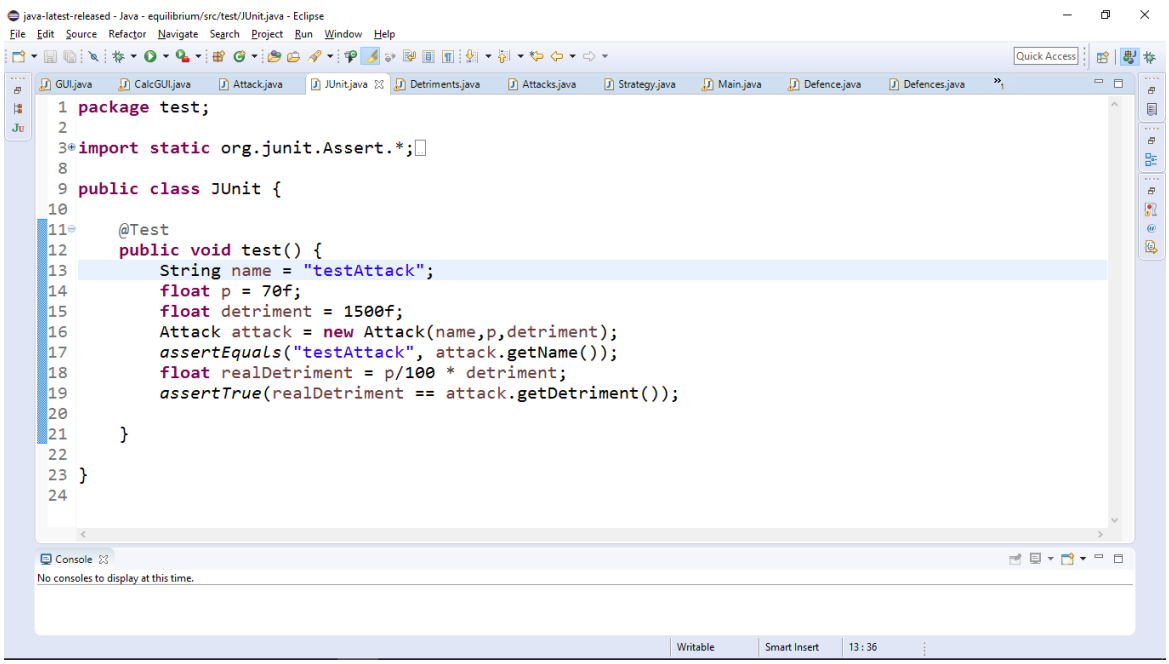


Рисунок 3.8 – Пример JUnit теста

ЗАКЛЮЧЕНИЕ

В ходе проделанной работы были изучены теоретические основы игрового подхода. На основе которого, были построены две математические модели конфликта «злоумышленник – администратор». Для оптимизации выбора набора средств защиты информации применялся критерий Сэдвиджа. Также с помощью данного критерия можно рассчитать максимально возможный ущерб системе от применения атак злоумышленником. Это позволяет администратору, опираясь на программу, не строить субъективные решения, а использовать лучший оптимальный вариант защиты информации.

Основной задачей выпускной квалификационной работы была разработка программы, которая позволяет на основании введенных пользователем данных, решить задачу выбора средств защиты. Эта задача была успешно достигнута, и в ходе данной работы был подробно описан процесс создания программы. Результатом процесса является программа, которая дает администратору или службе безопасности возможность эффективно подобрать программное обеспечения для защиты информации.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

Научная и методическая литература

1. Внуков, А. А. Защита информации: Учебное пособие / А. А. Внуков. – М.: Юрайт, 2017 г. – 242 с.

2. Гуц, А. К. Теория игр и защита компьютерных систем: Учебное пособие / А. К. Гуц, Т. В. Вахний. – Омск: Изд-во ОмГУ, 2013 г. –160 с.

3. Денисов, Д. В. Безопасность в интернете: защита от внешних угроз: часть сборника / Д. В. Денисов. – СПб.: Синергия, 2016 г. – 8 с.

4. Колобашкина, Л. В. Основы теории игр: Учебное пособие / Л. В. Колобашкина. – М.: БИНОМ. Лаборатория знания, 2014 г. – 200 с.: ил.

5. Кумарина, Г. Ф. Теория и методика игры: Учебник и практику для прикладного бакалавриата / Г. Ф. Кумарина, О. А. Степанова. – М.: Юрайт, 2016 г. – 276 с.

6. Лабскер, Л. Г. Теория игр в экономике (практикум с решениями задач): Учебное пособие / Л. Г. Лабскер, Н. А. Ященко. – М.: КНОРУС, 2014 г. – 264 с.

7. Лемешко, Б. Ю. Теория игр и исследование операций: Конспект лекций / Б. Ю. Лемешко. – Новосибирск: Изд-во НГТУ, 2013 г. – 167 с.

8. Новиков, Ф. А. Дискретная математика / Ф. А. Новикков. – СПб.: Питер, 2017 г. – 496 с.

9. Шаньгин, В. Ф. Информационная безопасность и защита информации: электронное издание / В. Ф. Шаньгин. – Саратов: Профобразование, 2017 г. – 702 с.: ил.

10. Шапкин, А. С. Математические методы и модели исследования операций / А. С. Шапкин, В. А. Шапкин. – М.: Дашков и Ко, 2017 г. – 398 с.

11. Ширяв, В. И. Исследование операций и численные методы оптимизации: Учебное пособие / Ширяв В. И. – М.: Ленанд, 2017 г. – 224 с.

12. Салмина, Н. Ю. Теория игр: Учебное пособие / Н. Ю. Салмина. – Томск: Эль Контент, 2012 г. – 92 с.

Электронные ресурсы

13. Сайт «Олбест» [Электронный ресурс]: статья., Режим доступа: http://otherreferats.allbest.ru/mathematics/00029136_0.html, свободный (дата обращения 15.05.2017).

14. Сайт научная электронная библиотека «киберленинка» [Электронный ресурс]: статья., Режим доступа: https://cyberleninka.ru/article/n/programmnoe-prilozhenie-dlya-vybora-optimalnogo-nabora-sredstv-zaschity-kompyuternoy-informatsii-na-osnove-teorii-igr, свободный (дата обращения 23.05.2017).

Литература на иностранном языке

15. Aumann R. J., Maschler M., Stearns R. E. Repeated games of incomplete information: an approach to the non-zero sum case // Report of the U.S. Arms Control and Disarmament Agency ST-143. – Washington, D.C., 1966. – Chap. IV. Pp. 117–216.

16. Marino A., De Meyer B. Continuous versus Discrete Market Games // Cowles Foundation Discussion Paper No. 1535. – 2005.

17. Mertens J. F., Zamir S. The value of two-person zero-sum repeated games with lack of information on both sides // International Journal of Game Theory. – 1971. – Vol. 1. – Pp. 39–64.

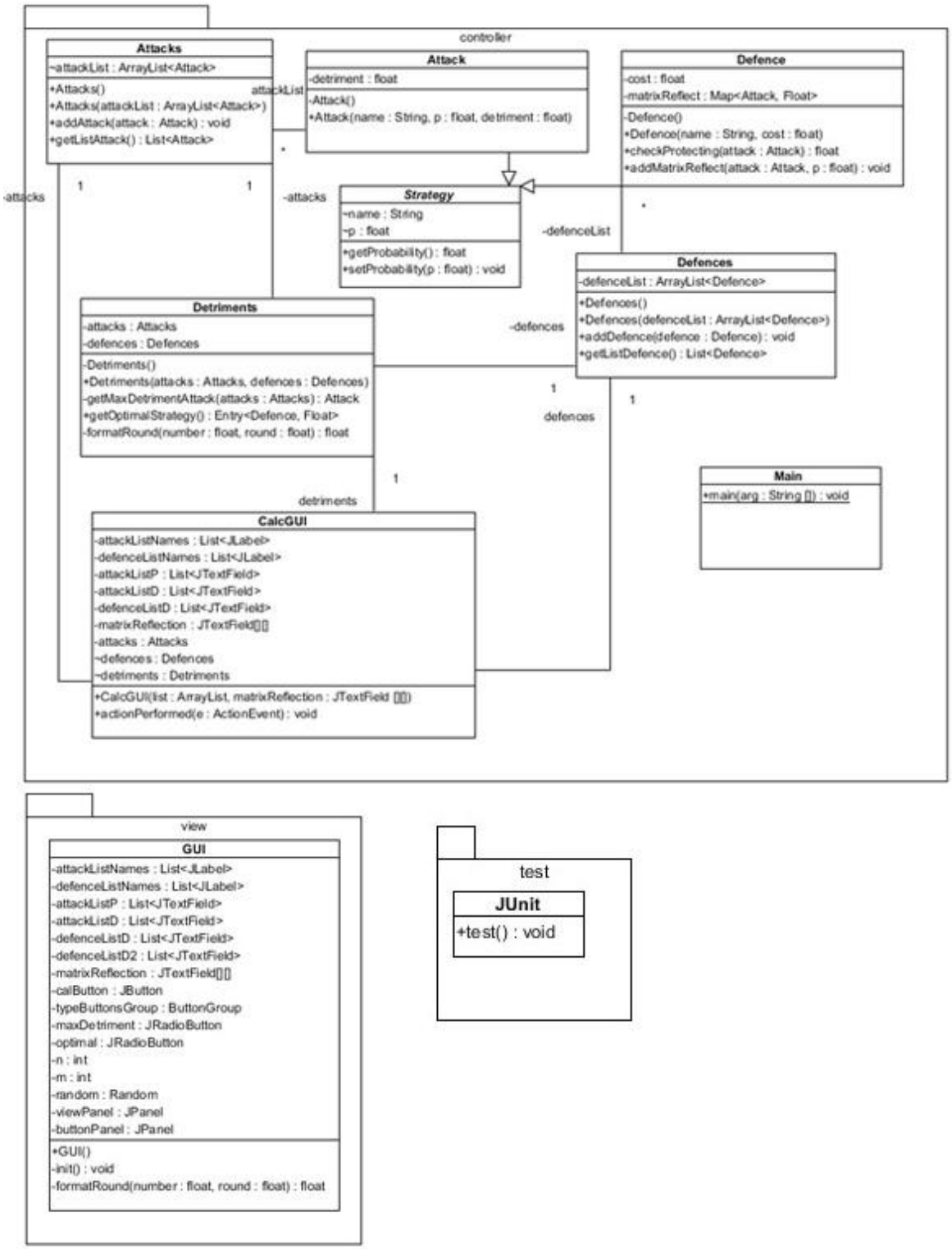
18. Myerson R. B. Game Theory: analysis of conflict. – Harvard University: Harvard University Press, 1997. – 600 pages.

19. Maschler M., Solan E., Zamir S. Game Theory. – Cambridge: Cambridge University Press, 2015.

20. Osborne M.J. An Introduction to Game Theory. – Oxford University: Oxford University Press – 2003.

ПРИЛОЖЕНИЕ А

Диаграмма классов



ПРИЛОЖЕНИЕ Б

Листинг кода программы

файл Main.java

```
package controller;

import javax.swing.*;

import java.util.Map.*;
import view.*;

public class Main
   {
    public static void main(String arg[])
    {

        try{

      UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel")
;
      }catch(Exception e){
      e.printStackTrace();
      }
      JFrame.setDefaultLookAndFeelDecorated(true);
        GUI gui = new GUI();
             gui.setLocationRelativeTo(null);
        gui.setVisible(true);
        gui.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

}
```

файл Attack.java

```
package controller;

import java.util.*;

public class Attack extends Strategy
{
   private StateDetriment stateDetriment;
   private float detriment;

   private Attack()
   {

   }

   public Attack(String name, float p, float detriment)
   {
      this.name = name;
      this.p = p /100;
      this.detriment = detriment;
   }


   public StateDetriment getStateDetriment()
   {
       return stateDetriment;
```

```
   }

   public float getDetriment()
   {
      return detriment * p;
   }

}
```

файл Attacks.java

```
package controller;

import java.util.*;

public class Attacks
{
   ArrayList<Attack> attackList;

    public Attacks()
    {
        attackList = new ArrayList<Attack>();
    }

    public Attacks(ArrayList<Attack> attackList)
    {
         this.attackList = new ArrayList<Attack>(attackList);
    }

    public void addAttack(Attack attack){
        attackList.add(attack);
    }

     public List<Attack> getListAttack(){
        return attackList;
    }


}
```

файл CalcGUI.java

```
package controller;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.Collections;
import java.util.List;
import java.util.Map.Entry;

import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class CalcGUI implements ActionListener {

      private List<JLabel> attackListNames;  //Список названий угроз
```

```
private List<JLabel> defenceListNames; //Список названий защит
private List<JTextField> attackListP;  //Поля для ввода вероятности успеха
private List<JTextField> attackListD;  //Поля для ввода предполагаемого ущерба
private List<JTextField> defenceListD; //Поля для ввода стоимости защиты
private JTextField[][] matrixReflection; // Матрица отражения атак
private Attacks attacks;
Defences defences;

Detriments detriments;
   public CalcGUI(ArrayList<Object> list, JTextField[][] matrixReflection){
         if(!list.isEmpty()){
               try{
                     this.attackListNames = (ArrayList<JLabel>) list.get(0);
                     this.attackListP = (ArrayList<JTextField>) list.get(1);
                     this.attackListD = (ArrayList<JTextField>) list.get(2);
                     this.defenceListNames = (ArrayList<JLabel>) list.get(3);
                     this.defenceListD = (ArrayList<JTextField>) list.get(4);
                     this.matrixReflection = matrixReflection;
               } catch(Exception e) {
                     e.printStackTrace();
               }
         }
          attacks = new Attacks();
        defences = new Defences();
   }

   @Override
   public void actionPerformed(ActionEvent e) {
          //Заполняем угрозы(атаки)
        for(int i = 0; i<attackListNames.size(); ++i){
          String name = null;
          float p = 0;
          float detriment = 0;
          try{
                name = attackListNames.get(i).getText();
                p = Float.parseFloat(attackListP.get(i).getText());
                detriment = Float.parseFloat(attackListD.get(i).getText());
          }catch (Exception e1){
                e1.printStackTrace();
          }
          attacks.addAttack(new Attack(name,p/100,detriment));
        }

      //Заполняем защиты
        for(int i = 0; i<defenceListNames.size(); ++i){
          String name = null;
          float detriment = 0;
          try{
                name = defenceListNames.get(i).getText();
                detriment = Float.parseFloat(defenceListD.get(i).getText());
          }catch (Exception er){
                er.printStackTrace();
          }
          defences.addDefence(new Defence(name,detriment));
        }
        //Заполнение матрицы защита отражает атаку с вероятностью
        System.out.println("Matrix reflection:"); //
         // заполнение  вероятностями матрицу отражения
         for(int i = 0; i < matrixReflection.length; ++i){
               Defence defence = defences.getListDefence().get(i);
               for(int j= 0; j < matrixReflection[i].length; ++j){
```

```
                        JTextField field = matrixReflection[i][j];
                        Attack attack = attacks.getListAttack().get(j);
                        float p = Float.parseFloat(field.getText());
                        defence.addMatrixReflect(attack, p);
                        System.out.println(defence.getName() + "-" +
attack.getName() + "=" + p);
                    }
            }

           detriments = new Detriments(attacks, defences);



          //Показываем результат
             Entry<Defence, Float> optimal = detriments.getOptimalStrategy();
             String result = "Optimal defence = " + optimal.getKey().getName() + "
Maximum detriment = " + optimal.getValue();
             JOptionPane.showMessageDialog(null, result, "Результат",
JOptionPane.DEFAULT_OPTION);
          System.out.print("Optimal defence = " + optimal.getKey().getName() + "
Maximum detriment = " + optimal.getValue());
      }

      }
```

файл Defence.java

```
package controller;

import java.util.HashMap;
import java.util.Map;

public class Defence extends Strategy
{
    private float cost;
    private Map<Attack, Float> matrixReflect; // Матрица: с какой вероятностью защита
                                                // успешно
отразить атаку
    private Defence()
    {

    }

   public Defence(String name, float cost)
   {
      this.name = name;
      this.cost = cost;
      matrixReflect = new HashMap<>();
   }

   public float checkProtecting(Attack attack)
   {
         //расчет прогнозируемого ущерба если хакер применил attack к данной защите:
         // (вероятностный) ущерб системе от атаки * вероятность успеха защиты)) +
стоимость защиты
        float spedings = attack.getDetriment() * (1 - matrixReflect.get(attack)) +
cost;
        return spedings;
   }

   public float getCost()
```

```
    {
        return cost;
    }

    public void addMatrixReflect(Attack attack, float p){
                //Заполнение матрицы вероятности отражения атаки
                      matrixReflect.put(attack, p);
         }
}
```

файл Defences.java

```
package controller;

import java.util.*;

public class Defences
{

    private ArrayList<Defence> defenceList;

     public Defences()
     {
         defenceList = new ArrayList<Defence>();
     }

     public Defences(ArrayList<Defence> defenceList)
     {
          this.defenceList = new ArrayList<Defence>(defenceList);
     }

     public void addDefence(Defence defence){
         defenceList.add(defence);
     }

     public List<Defence> getListDefence()
     {
         return defenceList;
     }


       }
```

файл Detriments.java

```
package controller;

import java.text.DecimalFormat;
import java.util.Map.*;

import javax.swing.JTextField;

import java.util.AbstractMap.SimpleEntry;
import java.util.ArrayList;
import java.util.List;

public class Detriments
{
       private Attacks attacks;
       private Defences defences;
```

```
    private Detriments()
    {

    }

    public Detriments(Attacks attacks, Defences defences)
    {
       this.attacks = attacks;
       this.defences = defences;
    }

    private Attack getMaxDetrimentAttack(Attacks attacks){
       Attack maxAttack = null;
       float maxDetriment = 0;
       for(Attack attack: attacks.getListAttack()){
             if(maxDetriment < attack.getDetriment()){
                   maxDetriment = attack.getDetriment();
                   maxAttack = attack;
             }
       }
       return maxAttack;
    }

    public Entry<Defence, Float> getOptimalStrategy()
    {
        float minDetriment = Float.MAX_VALUE;
        Defence optimalDefence = null;
        Attack maxAttack = getMaxDetrimentAttack(attacks);
        for(Defence defence : defences.getListDefence())
        {
             if(minDetriment > defence.checkProtecting(maxAttack)){
                   minDetriment = defence.checkProtecting(maxAttack);
                   optimalDefence = defence;
             }
        }

        return new SimpleEntry<Defence, Float>(optimalDefence, minDetriment);
    }

    private float formatRound(float number, float round)
    {
       float temp=(float) Math.pow(10,round);
        return Math.round(number*temp)/temp;
    }

     }
```

файл Strategy.java

```
package controller;


public abstract  class Strategy
{
    String name;
    float p;

    public String getName(){
        return name;
    }
```

```
    public void setName(String name){
        this.name = name;
    }

     public float getProbability(){
        return p;
    }

    public void setProbability (float p){
        this.p = p;
    }
      }
```

файл GUI.java

```
package view;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Container;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JTextField;

import controller.CalcGUI;

public class GUI extends JFrame {
    private List<JLabel> attackListNames;  //Список названий угроз
    private List<JLabel> defenceListNames; //Список названий защит
    private List<JTextField> attackListP;  //Поля для ввода вероятности успеха
    private List<JTextField> attackListD;  //Поля для ввода предполагаемого ущерба
    private List<JTextField> defenceListD; //Поля для ввода стоимости защиты
    private JTextField[][]  matrixReflection; // Поля матрицы: защита отражает атаку
с верояностью
    private JButton calButton;
    private ButtonGroup typeButtonsGroup;
    private JRadioButton maxDetriment;
    private JRadioButton optimal;
    private int n, m; //Размерность матрицы Угрозы - Защиты
    private Random random;
    private JPanel viewPanel;
    private JPanel buttonPanel;

    public GUI(){
         super("Test Windows");
         init(); //Инициализация объктов панели
         this.setBounds(200, 200, 1200, 600);

         Container container = this.getContentPane();
         viewPanel.setLayout(new GridBagLayout());
         viewPanel.setBackground(new Color(176, 196, 222));
```

```
        buttonPanel.setLayout(new GridBagLayout());
        buttonPanel.setBackground(new Color(150, 196, 120));

        GridBagConstraints c = new GridBagConstraints();
        c.fill = GridBagConstraints.HORIZONTAL;
        //Заполнение угрозы
        for(int i = 0; i<n; ++i){
               attackListNames.add(new JLabel(i+1 + ". attack" + i));
            c.gridx = 0;
            c.gridy = i;

            viewPanel.add(attackListNames.get(i), c);

            float var =  formatRound((float)Math.random(), 2);
            attackListP.add(new JTextField(String.valueOf(var),5));
            c.gridx = 2;
            viewPanel.add(attackListP.get(i), c);

            var = (float)random.nextInt(10000);
            attackListD.add(new JTextField(String.valueOf(var), 8));
            c.gridx = 4;
            viewPanel.add(attackListD.get(i), c);
        }
        //Заполнение защиты
        for(int i = 0; i<m; ++i){
               defenceListNames.add(new JLabel(i+1 + ". defence" + i));
            c.gridx = 10;
            c.gridy = i;
            viewPanel.add(defenceListNames.get(i), c);

            float var = (float)random.nextInt(100000);
            defenceListD.add(new JTextField(String.valueOf(var),8));
            c.gridx = 12;
            viewPanel.add(defenceListD.get(i), c);

        }
        c.gridy=1;
        int startPositionX;
        int startPositionY = c.gridy;
        matrixReflection = new JTextField[m][n];
        for(int i = 0; i<m; ++i){
                startPositionX = c.gridx = 17;
                //List<JTextField> Listfield = new ArrayList<JTextField>(); //
список строк
               for(int j = 0; j<n; ++j){
                      float p = formatRound((float)Math.random(), 2);
                      matrixReflection[i][j] = new JTextField(String.valueOf(p),5);
                      viewPanel.add( matrixReflection[i][j], c);
                      ++startPositionX;
                      c.gridx = startPositionX;
               }
               ++startPositionY;
               c.gridy = startPositionY;
        }
        c.gridy = c.gridy - m - 1;
        c.gridx = 15;
        viewPanel.add(new JLabel("Matrix Reflection:"), c);

        boolean f = false;
        startPositionX = c.gridx = 17;
        startPositionY = c.gridy + 1;
```

```
      for(int i = 0; i <m; ++i){
            if(!f){
                  for(int j = 0; j<n; ++j){
                        viewPanel.add(new JLabel("     x" + (j + 1)), c);
                        ++startPositionX;
                        c.gridx = startPositionX;
                  }
                  f = true;
            }
            c.gridx = 15;
            c.gridy = startPositionY;
            viewPanel.add(new JLabel("                     y" + (i + 1)), c);
            ++startPositionY;
      }

      typeButtonsGroup.add(maxDetriment);
   typeButtonsGroup.add(optimal);
   maxDetriment.setSelected(true);

      c.gridx = 0;
      c.gridy = 2;
      buttonPanel.add(maxDetriment, c);

      c.gridx = 2;
      c.gridy = 2;
      buttonPanel.add(optimal, c);

      c.gridwidth = 3;
      c.gridx = 0;
      c.gridy = 6;
      buttonPanel.add(calButton, c);

      BorderLayout mainLayout = new BorderLayout();
      container.setLayout(mainLayout);
      container.add(viewPanel, BorderLayout.PAGE_START);
      container.add(buttonPanel,BorderLayout.CENTER);


      ArrayList<Object> list = new ArrayList<Object>();
      list.add(attackListNames);
      list.add(attackListP);
      list.add(attackListD);
      list.add(defenceListNames);
      list.add(defenceListD);
     // list.addAll(matrixReflection);

      CalcGUI calcGUI = new CalcGUI(list, matrixReflection);
      //Листенер для подсчета введенных данных
      calButton.addActionListener(calcGUI);
}


private void init(){
   viewPanel        = new JPanel();
   buttonPanel      = new JPanel();
      attackListNames  = new ArrayList<JLabel>();
      defenceListNames = new ArrayList<JLabel>();
      attackListP      = new ArrayList<JTextField>();
      attackListD           = new ArrayList<JTextField>();
      defenceListD          = new ArrayList<JTextField>();
      typeButtonsGroup = new ButtonGroup();
```

```
        maxDetriment     = new JRadioButton("Макимальный возможный ущерб");
        optimal          = new JRadioButton("Оптимальная стратегия защиты");
        calButton              = new JButton("Рассчитать");
        n                = 15; // Угрозы
        m                      = 12; //Защиты
        random           = new Random();
    }

    private float formatRound(float number, float round)
    {
       float temp=(float) Math.pow(10,round);
        return Math.round(number*temp)/temp;
    }

      }
```

файл Junit.java

```
package test;

import static org.junit.Assert.*;

import org.junit.Test;

import controller.Attack;

public class JUnit {

      @Test
      public void test() {
            String name = "testAttack";
            float p = 70f;
            float detriment = 1500f;
            Attack attack = new Attack(name,p,detriment);
            assertEquals("testAttack", attack.getName());
            float realDetriment = p/100 * detriment;
            assertTrue(realDetriment == attack.getDetriment());

      }

      }
```