Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет геосистем и технологий»
(СГУГиТ)

А. А. Басаргин

МЕТОДЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Утверждено редакционно-издательским советом университета
в качестве учебного пособия для обучающихся по направлению подготовки
09.03.02 Информационные системы и технологии (уровень бакалавриата)

Новосибирск
СГУГиТ
2022

УДК 004.414.23
Б27

Рецензенты: кандидат технических наук, доцент НГТУ *В. С. Карманов*
кандидат технических наук, доцент, СГУГиТ *А. А. Колесников*

Басаргин, А. А.

Б27 Методы искусственного интеллекта : учебное пособие / А. А. Басаргин. – Новосибирск : СГУГиТ, 2022. – 164 с. – Текст : непосредственный.
ISBN 978-5-907513-45-7

Учебное пособие подготовлено кандидатом технических наук, доцентом А. А. Басаргиным на кафедре прикладной информатики и информационных систем СГУГиТ.

В настоящем издании содержится краткое изложение теоретического материала и примеры решения практических задач, направленных на реализацию создания нейронных сетей для исследования генетических алгоритмов, а также для решения задач управления и наблюдения методами нечеткой логики.

Учебное пособие по дисциплине «Методы искусственного интеллекта» предназначено для обучающихся по направлению подготовки 09.03.02 Информационные системы и технологии (уровень бакалавриата) по темам в объеме, определенным рабочим планом учебной дисциплины «Методы искусственного интеллекта».

Данное пособие рекомендовано к изданию кафедрой прикладной информатики и информационных систем, Ученым советом Института геодезии и менеджмента СГУГиТ.

Печатается по решению редакционно-издательского совета СГУГиТ

УДК 004.414.23

ISBN 978-5-907513-45-7

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Данное издание является частью учебно-методического комплекса по дисциплине «Методы искусственного интеллекта» и предназначено для проведения практических занятий у обучающихся по программе подготовки 09.03.02 Информационные системы и технологии (уровень бакалавриата).

Главная цель учебного пособия – помощь обучающимся в организации самостоятельной работы в процессе лабораторных занятий и подготовки к экзамену по дисциплине «Методы искусственного интеллекта».

Искусственный интеллект (ИИ) является одним из разделов информатики. Основной задачей ИИ будет являться разработка аппаратно-программных средств. Они позволяют пользователю ставить и решать свои интеллектуальные задачи. При этом взаимодействие с интеллектуальной информационной системой происходит на естественном языке.

Методы искусственного интеллекта разнообразны. Для решения конкретной задачи методы будут заимствоваться и адаптироваться из других наук. Во время проектирования и реализации интеллектуальной информационной системы требуется привлекать специалистов из прикладной области знаний.

Исходя из этого, для создания ИИ привлекаются специалисты из экономики, программирования лингвистики, психологии, нейрофизиологии, информатики и т. д. Одним из главных направлений информатики является исследование, разработка и реализация методов для решения задач распознавания образов. Данная задача является важной частью автоматизированных систем, систем управлений и обработки информации и систем принятия решений. Классификация явлений, сигналов и предметов, которые определяются набором признаков и свойств, является задачей, находящей свое применение во многих отраслях, например, диагностической медицине, робототехнике, исследовании искусственного интеллекта, информационном поиске и многом другом. Обработка изображений и классифика-

ция визуальных образов находит применение в системах безопасности и видеонаблюдения, в управлении и контроле доступа, в системах виртуальной реальности и поиска по изображению. Системы по распознаванию автомобильных знаков, маркировок изделий, человеческих лиц и отпечатков пальцев, рукописных текстов на данный момент широко используются в различных сферах современной жизни.

Важность задачи классификации и ее актуальность можно отследить по работам Т. Кохонена «Self-organization and associative memory», А. Крыжевского «Imagenet classification with deep convolutional neural networks», Т. Визеля и Д. Хьюбела «Brain and visual perception», Я. Лейкуна «Comparison of learning algorithms for handwritten digit recognition» и многих других. Ощутимый прогресс в решении задач распознавания образов был достигнут благодаря появлению новых методов: сверхточных нейронных сетей (НС), констелляционных моделей и методов снижения размерности. Несмотря на значительный прогресс, достигнутый в последнее время в сфере классификации образов, современные исследования подчеркивают тот факт, что существующие методы и алгоритмы по распознаванию образов до сих пор отстают от модели биологической зрительной системы, потому что они не способны функционировать на неограниченном множестве классов распознавания, не имеют устойчивости к вариативности объектов в пределах категорий и их инвариантным преобразованиям.

Поэтому актуальной проблемой на данном этапе развития систем по распознаванию образов остается классификация объектов на изображении под действием преобразований, при которых значительно изменяется форма объекта на изображении, но при этом принадлежность объекта к классу остается неизменной. В теории классификации образов эта проблема имеет название «проблема инверсии». Многослойные сверточные сети, методы ORB и SIFT предоставляют частичное решение проблемы инверсии, обеспечивая стойкость только к ограниченному подмножеству преобразований. В тех сферах жизнедеятельности, где для классификации используются объекты, взятые из естественной среды, например, зрительная система в робототехнике, видеонаблюдении, анализе данных с фотографий и тому подобное, данная проблема является наиболее актуальной.

Учебное пособие призвано обеспечить:

1) самостоятельную работу обучающихся по данной дисциплине;

2) контроль знаний обучающихся во время выполнения лабораторных работ;

3) представление о задачах и содержании практической составляющей учебной дисциплины «Методы искусственного интеллекта».

В конце учебного пособия приведен список рекомендуемой литературы, который позволяет обучающимся самостоятельно изучить дополнительный материал по темам дисциплины «Методы искусственного интеллекта».

1. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ПОДГОТОВКЕ К ЛАБОРАТОРНЫМ РАБОТАМ

Каждый обучающийся должен ознакомиться с теоретическими основами и описанием, разбираемых в пособие, примеров, выполнить задание на ПК, придерживаясь описанной технологии, а также подготовить печатный отчет.

Для выполнения лабораторных работ необходимо использовать ПК, на которых установлен офисный пакет программ и программные приложения *Neuroph Studio* (создание искусственных нейронных сетей), Darwin (исследование генетических алгоритмов), CLIPS (разработка программ на составление планов действий технических систем), FisPRO (решение задач управления и наблюдения методами нечеткой логики).

После выполнения лабораторной работы обучающийся проходит процедуру ее защиты, отвечая на вопросы преподавателя. Работа считается полностью выполненной при наличии у обучающегося всех файлов, подтверждающих практическое выполнение работы, аккуратно оформленного печатного отчета и положительной оценки за ответы на контрольные вопросы.

Содержание и оформление отчета

1. Цель работы.
2. Формулировка задания.
3. Подробное описание выполненных действий.
4. Ответы на контрольные вопросы.
5. Общий вывод о проделанной работе.

К отчету должны быть приложены файлы с именем, заданным по шаблону «Номер группы. Фамилия обучающегося», созданные в результате выполнения задания.

2. ОСНОВНЫЕ ЗАДАЧИ ТЕОРИИ КЛАССИФИКАЦИИ И РАСПОЗНАВАНИЯ ОБРАЗОВ

Распознавание, или классификация образов, – это наука, которая занимается изучением методов и алгоритмов классификации различных объектов в мире. Каждый день мы сталкиваемся с задачей распознавания, встречая знакомых на улице, читая книги, решая примеры, слушая музыку, воспринимая речь и многое другое. В процессе своего существования живые организмы вынуждены решать задачу классификации объектов, и от того, как хорошо и быстро они могут это делать, зависит их существование. В последнее время задачу классификации стараются решать с помощью вычислительных средств.

Классификация – это систематическое распределение рассматриваемых предметов, процессов и явлений по родам, типам и видам по каким-либо значимым признакам [1].

Можно выделить следующие основные задачи, в решении которых используются методы распознавания образов [3]:

– медицинская диагностика;
– распознавание речи;
– анализ сцен;
– распознавание изображений;
– распознавание символов;
– кластеризация, классификация, поиск в базах знаний и данных.

Основные задачи из теории распознавания образов рассмотрим на примере работы технической системы, которая показана на рис. 2.1 [1].

Известно, что на вход технической системы поступает изображение некого символа χ, который требуется распознать. Этот объект называется образом. Далее, система по некоторому набору правил преобразует данный символ χ в вектор признаков $\overline{\chi}$. Входной объект χ будет принадлежать некоторому классу ϖ_k из множества всех классов $\Omega = \{\omega_1 .. \omega_m\}$. Классы не должны пересекаться. Для решения задачи по распознаванию образа

должен быть сопоставлен образ χ одному из классов ϖ_k. Это делается с помощью классификатора.

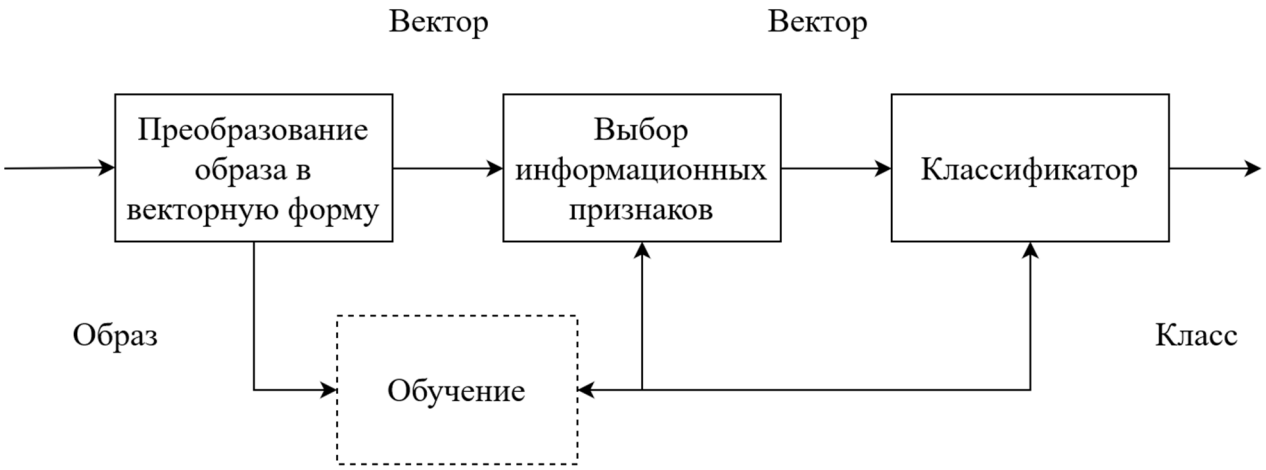

Рис. 2.1. Принципиальная схема системы распознавания образов

Классификатором называется набор правил, по которым образ соотносится одному из классов и реализуется в блоке классификации. На выход классификатор должен выдать класс, к которому относится объект, а также указать степень соответствия данному классу или выдать информацию, что входной объект не принадлежит множеству классов [3].

Для уменьшения размерности вектора-признака и повышения быстродействия системы может использоваться блок выбора информативных признаков. Данный блок располагается между блоками преобразования образа в векторную форму и классификатором [1].

В общей системе распознавания образов может быть обучающий модуль, который позволяет на основе образца обучающих изображений, с которым связан соответствующий класс, формировать правила классификации. Также по обучающей выборке можно выделить правило наиболее информативных признаков [1].

На основе принципиальной схемы системы распознавания образов можно выделить следующие основные задачи теории распознавания [10]:

– математическое описание образов. Для этого в математике используется векторное описание образов. Векторное описание образов заключа-

ется в том, что каждому образу будет сопоставлен вектор признаков $\chi = (\chi_1, ..., \chi_n)^T$, где χ_i – признак этого образа. Пространством признаков будут элементы векторного пространства *X*. Такое пространство должно быть метрическим и конечномерным. Также для задачи распознавания изображений можно применить матричное описание;

– подборка наиболее информативных признаков, описывающих данное изображение. Одна из основных задач распознавания образов – найти минимальное количество признаков, по которым будет достигнута требуемая точность распознавания, то есть поиск знаков, наиболее информативно описывающих изображение в данной системе распознавания. Алфавит в этом случае будет полным набором признаков, используемых для распознавания, а словарь признаков – минимальным набором признаков, достаточным для решения проблемы распознавания. Эффективность системы распознавания во многом зависит от выбора алфавита знаков и определения словаря;

– описание классов распознаваемых образов. Такая задача заключается в нахождении границ классов. Система в процессе своей работы может определить границы классов или границы классов можно задать на этапе разработки;

– поиск оптимальных методов классификации, т. е. определение методов, необходимых для сопоставления вектора признаков некоторому классу;

– оценка достоверности классификации изображений. Эта оценка позволяет оценить размер потерь, связанных с неправильной классификацией.

3. МАТЕМАТИЧЕСКАЯ ПОСТАНОВКА ЗАДАЧИ КЛАССИФИКАЦИИ И СИСТЕМ РАСПОЗНАВАНИЯ ОБРАЗОВ

Характеристики образов (признаки) в основном можно разделить на следующие типы [7]:

– физические характеристики (детерминированные и вероятностные), например, показания различных датчиков. Данный тип характеристик лучше всего представлять в векторном описании;

– качественные характеристики, например, понятия «низкий», «высокий», «толстый» и т. п. Данный вид характеристик можно описать с помощью теории нечетких множеств;

– структурные характеристики, которые описывают изображение сложных объектов и сцен. На основе формального языка можно описать структурные характеристики;

– логические характеристики: высказывания, которые соответствуют значениям «истина» или «ложь».

По характеру информативных признаков системы распознавания можно разделить на следующие типы [2]: логические, детерминистские, структурные, вероятностные и комбинированные.

Другим критерием для классификации систем распознавания является количество априорной информации для распознавания. По данному критерию системы распознавания можно разделить на три типа [2]:

– системы без обучения, т. е. в данной системе имеется достаточное количество априорной информации для составления алфавита и словаря признаков, а также выделения границ классов. В данной системе отсутствует блок обучения;

– системы, основанные на обучении с учителем. В данных системах количество априорной информации позволяет выделить только алфавит и словарь признаков. Необходимо условие минимальности ошибки неправильной классификации, чтобы система самостоятельно выделяла правила классификации. Для выделения таких правил системе предоставляется

набор размеченных данных (т. е. данных, в которых каждому объекту сопоставлен его класс), называемых обучающей выборкой;

– системы, основанные на самообучении (на объяснении). В этом случае задача распознавания образов сводится к вводу списка правил, который основан на априорной информации и не позволяет сформировать словарь признаков. Для определения словаря признаков и определения границ классов система распознавания основывается на данном наборе признаков.

Пусть U – множество объектов в задачи распознавания, а u – отдельный объект распознавания. Каждый образ $u \in U$ может характеризоваться бесконечным числом признаков. Мы должны сформировать подмножество признаков на этапе формирования алфавита. Данное подмножество будет называться пространством признаков X. В большинстве случаев множество X состоит из линейных или метрических структур. Обычно X – линейное или конечное метрическое $(X = R^n)$ пространство. Пусть x – элемент пространства X, соответствующий образу $u \in U$, а $P : U \to X$ – оператор, отображающий u в x. Отметим, что $X = P(U)$, а $P^2 = P$, т. е. P является оператором проектирования. Допустим, что в задаче классификации во множестве образов U нас интересуют некоторые подмножества, называемые классами. Множество классов $\Omega = \{\omega_1 .. \omega_m\}$ в базовой задаче классификации является конечным. Классы в таком случае образуют полную группу подмножеств из U, т. е. [1]:

$$\bigcup_{i=1}^{m} \overline{\omega} = U, u\varpi_i \bigcap \varpi_j . \qquad (3.1)$$

В данном случае задача классификации называется обобщенной.

Классифицировать образ $u \in U$ по классам $\omega_1, ..., \omega_m$ – это значит найти функцию $g : U \to Y$, $Y = \{y_i, ..., y_m\}$, которая называется индикаторной. Данная функции сопоставляет образ $u \in U$ и метку класса $y_i \in Y$, которому принадлежит рассматриваемый образ, т. е. $g(x) = c$ при условии, что $x \in \overline{\omega}_i$ [1].

На самом деле мы не имеем дело со всем множеством образов U, а только с проекцией $X = P(U)$, которая называется пространством призна-

ков. Тогда требуется найти решающую функцию $\tilde{g}: X \to Y$, которая сопоставляет каждому вектору признаков $x = P(U) \in X$ метку $y_i \in Y$, соответствующую классу ϖ_i, т. е. $\tilde{g}(x) = y_i$ при условии, что $x = P(u), u \in \varpi_i$ [1].

Следует заметить, что множество $P^{-1}(x), x \in X$ может быть не одноэлементным, т. е. данное множество будет иметь непустые пересечения с другими классами ϖ_i. Поэтому задача классификации является некорректной, т. е. значение функции $\tilde{g}(x)$ может быть неоднозначным [8]. В соответствии с общим подходом решения некорректных задач из множества решений функции $\tilde{g}(x)$ можно выделить однозначную ветвь, если она будет удовлетворять определенным условиям оптимальности, например, минимальная ошибка неправильной классификации.

Множеству классов $\Omega = \{\omega_1, ..., \omega_m\}$ в пространстве признаков соответствует некоторое покрытие $\tilde{X}_1, ..., \tilde{X}_m$ пространства $X: \tilde{X}_i = \{x = P(U): x \in \varpi_i\}$, $i = 1, ..., m$. Так как покрытия могут пересекаться, то будет использоваться разбиение X на $X_1, ..., X_m$, удовлетворяющее условию $X_i \subseteq \tilde{X}_i$. Данное разбиение неоднозначно, поэтому наиболее правильный выбор информативных признаков приведет к снижению степени неоднозначности разбиения $X_1, ..., X_m$. Сама область X_i называется областями предпочтения классов ϖ_i.

Система распознавания на этапе обучения обладает информацией о классах в виде множества пар (x_j, y_j), $j = 1, ..., N$, $x_j = P(u_j) \in X$, $y_j = g(x_j) \in Y$. Данная пара (x_j, y_j) называется прецедентом, а множество $\Xi = \{x_1, ..., x_N\}$ – обучающей выборкой. Задача классификации заключается в нахождении решающего правила $\tilde{g}(x)$ по множеству прецедентов $\{\Xi, Y\}$, которое осуществляло бы классификацию обучающей выборки с наименьшим числом ошибок [2].

В обычном случае пространство признаков считается евклидовым, т. е. $X = R^i$. Частота появления правильных решений определяет качество решающего правила.

Для решения задачи классификации существует множество методов. К ним относятся: байесовский классификатор, линейный разделитель, индукция правил, алгоритмическая композиция, а также нейронные сети, которые в последнее время зарекомендовали себя в данной сфере.

4. ОБУЧЕНИЕ МАШИННОЕ И ПО ПРЕЦЕДЕНТАМ

Машинным обучением называется систематическое обучение алгоритмов и систем, в результате которого их знания или качество работы возрастают по мере накопления опыта [9].

Отсюда следует, что машинное обучение изучает алгоритмы, которые способны обучаться. Существует два метода обучения. Первый – обучение по прецедентам, также называется индуктивным обучением. Данный метод заключается в поиске общих закономерностей по частным эмпирическим данным. Второй метод заключается в формализации знаний, полученных экспертами, и переносе их в базы знаний. Так как принято считать, что дедуктивный метод относится к экспертным системам, то можно считать синонимами машинное обучение и обучение по прецедентам [2].

Пусть X – множество объектов, а Y – множество допустимых ответов, тогда существует функция $y^*: X \to Y$, называемая целевой функцией. Значение данной функции $y_i = y^*(x_i)$ установлено на конечном подмножестве $\{x_1, ..., x_i\} \in X$. Тогда прецедентом называется пара (x_j, y_j) «объект – значение», а обучающей выборкой называется набор пар $X^i = (x_i, y_i)_{i=1}^i$ [4].

Задача восстановления зависимости y^*, или построения решающей функции $\alpha: X \to Y$, приближающей функцию $y^*(x)$, на множестве X как объектов обучающей выборки, так и всех объектов данного множества, называется обучением по прецедентам [4].

Так как решающая функция α должна иметь эффективную реализацию на компьютере, то она называется алгоритмом.

Перейдем к описанию понятия объекта и признака.

Исход измерения определенной характеристики объекта называется признаком. Формально признак описывается отображением $f: X \to D_f$, где D_f – множество возможных значений признака. Следовательно, можно расценивать как признак алгоритм $\alpha: X \to Y$ [4].

Признаки D_f можно разделить на несколько типов в зависимости от природы [5]:

– бинарные признаки, если $D_f = \{0,1\}$;

– номинальные признаки, если D_f – конечное множество;

– порядковые признаки, если D_f – конечное упорядоченное множество;

– количественные признаки, если $D_f = R$.

Однородными признаки называются в том случае, если весь их набор имеет одинаковый тип, иначе признаки называются разнородными.

Пусть $f_1, ..., f_n$, тогда признаковым описанием объекта $x \in X$ называется вектор $f(x)_1, ..., f(x)_n$. В последующем будем полагать, что $X = D_{f1}, ..., D_{fn}$, т. е. признаковое описание объекта из X – это есть и сам объект из множества X. Матрицей признаков называется набор признаковых описаний для всех объектов выборки; он имеет форму записи в виде таблицы $l \times n$:

$$F = \left\| f_j(x_i) \right\|_{i*n} = \begin{pmatrix} f_1(x_1) & ... & f_n(x_1) \\ ... & ... & ... \\ f_1(x_i) & ... & f_n(x_i) \end{pmatrix}. \qquad (4.1)$$

В прикладных задачах наиболее часто используется матрица объектов признаков (4.1). Множество возможных ответов Y задачи обучения по прецедентам в зависимости от природы можно разделить на следующие типы [5]:

– задача классификации на M непересекающихся классов, если $Y = \{1, ..., M\}$. Для данной классификации алгоритм должен отвечать на вопрос «какому классу принадлежит x?», т. е. все объекты множества X разбиваются на классы $K_y = \left\{ x \in X : y^*(x) = y \right\}$;

– классификация на M пересекающихся классов, если $Y = \{0,1\}^M$.

В простой формулировке это поиск решения для M неизвестных задач классификации, имеющих два непересекающихся класса:

– задача восстановления регрессии, если $Y = R$;

– задачи прогнозирования, которые являются частными случаями восстановления регрессии и классификации, где прошлое поведение объекта описывается $x \in X$, а его будущее поведение описывается $y \in Y$.

Параметрическое семейство отображений $A = \{g(x), \theta \,||\, \theta \in \Theta\}$ называется моделью алгоритмов, где $g : X \times \Theta \to Y$ является некоторой фиксированной функцией, а Θ – пространство поиска (пространство параметров) – множеством допустимых значений параметра θ [1].

Обучением или настройкой алгоритма $a \in A$ будет называться процесс поиска оптимальных параметров по обучающей выборке X^i для модели θ [1].

Отображение $\mu : (X \times Y)^i \to A$, ставящее для произвольной конечной выборки $X^i = (x_i, y_i)_{i=1}^i$ некоторый алгоритм $a \in A$, называется методом обучения. Метод обучения должен допускать эффективную программную реализацию.

В обучении по прецедентам различают два этапа [3]:

– метод μ по выборке X^i строит алгоритм $\alpha = \mu(X^i)$ на этапе обучения;

– сопоставление ответа $y = \alpha(x)$ для нового объекта *x* происходит на этапе применения алгоритма α.

Также обучение делится на два основных метода – это обучение с учителем, где обучение происходит по размеченным данным (т. е. каждому значению обучающей выборки соответствует какой-то ответ), и обучение без учителя, где обучение осуществляется не по размеченным данным [2].

Самым сложным этапом является обучение, так как он заключается в поиске оптимальных параметров модели, при которых работа данного алгоритма будет соответствовать функционалу качества.

Неотрицательная функция $Z(\alpha, x)$, которая дает характеристику величине ошибки для алгоритма α на объекте *x*, называется функцией потерь. Корректным ответ $\alpha(x)$ называется в том случае, когда функция потерь равна нулю, т. е. $Z(\alpha, x) = 0$ [1].

Сумма всех функций потерь алгоритма α на выборке X^i, деленной на размерность выборки *X*, называется функционалом:

$$Q(\alpha, X^i) = \frac{1}{l} \sum_{i=1}^{i} Z(\alpha, x_i). \qquad (4.2)$$

Так как функция Q вычисляется по эмпирическим данным $(x_i, y_i)_{i=1}^{i}$, то ее называют эмпирическим риском, или функционалом средних потерь [1].

Бинарным функционал потерь называется в случае, когда принимает значение 0 или 1. Тогда $Z(\alpha, x) = 1$ показывает, что алгоритм α допускает ошибку на объекте x, а частотой ошибок называется функционал Q [10].

Часто используются функции потерь при $Y \in R$ [1]:

– индикатор ошибки – $Z(\alpha, x) = [\alpha(x) \neq y^*(x)]$. Обычно используется для задач классификации;

– отклонение от правильного ответа – $Z(\alpha, x) = [\alpha(x) - y^*(x)]$. В таком случае функционал Q называют средней ошибкой для алгоритма α на выборке X^i;

– квадратичная функция потерь – $Z(\alpha, x) = [\alpha(x) - y^*(x)]^2$. Тогда функционал Q будет называться средней квадратичной ошибкой для алгоритма a на выборке X^i. Часто применяется в задачах регрессии.

Минимизация эмпирического риска – это классический метод обучения, который состоит в том, чтобы найти в модели A алгоритм α, который для функционала качества Q будет доставлять минимальное значение на заданной обучающей выборке X^i [3]:

$$\mu(X^i) = \arg\min Q(\alpha, X^i). \qquad (4.3)$$

Элементы множества X в задачах обучения по прецедентам не являются реальными объектами, а представляют лишь доступные данные о них. Поскольку значения признаков $f_i(x)$ и целевой зависимости $y^*(x)$ обычно измеряются с погрешностями, то данные об объектах могут быть неточные. Также данные могут оказаться неполными в связи с тем, что измеряются не все признаки, а лишь доступные для измерения. Поэтому при одних и тех же описаниях могут соответствовать различные объекты и ответы. Тогда $y^*(x)$ не является функцией. Для этого вводится понятие вероятностной постановки задачи [6].

Предположим наличие неизвестного вероятностного распределения вместо использования неизвестной целевой функции $y^*(x)$ на множестве

$X * Y$ с плотностью $p(x, y)$. Из данного множества случайно и независимо выбирается l наблюдений $X^i = (x_i, y_i)_{i=1}^i$. Данные выборки называются случайно распределенными, или простыми [2].

Так как можно представить в виде вероятностного распределения $p(x, y) = p(x)p(y/x)$ функциональную зависимость $y^*(x)$, положив $p(y \| x) = \delta(y - y^*(x))$, где $\delta(z)$ – дельта-функция, то такая вероятностная постановка задачи является наиболее общей [3].

При вероятностной постановке задачи используется модель совместной плотности распределения ответов и объектов $\varphi(x, y, \theta)$, которая апроксимирует неизвестную плотность $p(x, y)$, взамен модели алгоритма $g(x, \theta)$, которая производит аппроксимацию неизвестной зависимости $y^*(x)$. Далее находится параметр θ, дающий максимально правдоподобную выборку данных X^l, т. е. данная выборка лучше всего согласуется с моделью плотности [2].

Совместная плотность распределения всех наблюдений в случае, если наблюдения в выборке X^i являются независимыми, равна произведению плотностей $p(x, y)$ в каждом наблюдении. Если вместо $p(x, y)$ подставить модель плотности $\varphi(x, y)$, то получится функция правдоподобия [2]:

$$L(\Theta, X^i) = \prod_{i=1}^{i} \varphi(x_i, y_i, \Theta). \qquad (4.4)$$

Необходимо искать значение, при котором функция (4.5) имеет максимальное значения, так как в этом случае выборка лучше согласуется с моделью. В математической статистике это получило название принцип максимума. После нахождения параметра θ, построить алгоритм $\alpha_\theta(x)$ по плотности $\varphi(x, y, \theta)$ не составит труда [2].

Минимизировать функционал $-\ln(L)$ вместо максимизации L проще, так как он аддитивен по объектам выборки, т. е. имеет вид суммы [2]:

$$\ln L(\theta, X^i) = -\sum_{i=1}^{i} \ln \varphi(x_i, y_i, \theta) \to \min. \qquad (4.5)$$

Если вероятностную функцию потерь определить как $Z(\alpha_\theta, x) = l \ln \varphi(x, y, \theta)$, то функционал – $\ln(L)$ совпадает с функционалом эмпирического риска, т. е. чем хуже пара (x_i, y_i) согласуется с моделью φ, тем выше величина потери $Z(\alpha_\theta, x)$ и меньше значение плотности распределения $\varphi(x, y, \theta)$ [2].

Для формализации задачи обучения существует два родственных подхода: один основан на введении функции потерь, другой – на введении вероятностной модели порождения данных. Такие подходы приводят к схожим оптимизационным задачам, т. е. обучение – это оптимизация [4].

Важно понимать, что в машинном обучении нет одного правильного ответа. Обычно данные зашумлены, и в этом случае было бы неверно стараться найти модель, которая на сто процентов классифицировала бы обучающие данные, так как эта модель оказалась бы переобученной, что в итоге привело бы к ошибкам распознавания новых данных. Переобучением (переподгонкой) называется эффект, когда алгоритм, работающий на объектах, не использующихся в обучающей выборке, показывает качество существенно хуже, чем на использующихся. Если минимизировать функционал $Q(\alpha, X^i)$ на алгоритме a, то это не значит, что данный алгоритм на целевой произвольной контрольной выборке $X^k = (x_i, y_i)_{i=1}^k$ будет хорошо приближать целевую зависимость [4].

Если метод минимизирует эмпирический риск, но при этом не способен обучаться, то в итоге он запомнит обучающую выборку X^i и построит алгоритм, который будет проводить сравнение объектов x с объектом x_i из обучающей выборки, т. е. если $x = x_i$, то алгоритм выдаст ответ y_i, иначе алгоритм выдаст произвольный ответ. Эмпирический риск в данном случае примет значение, равное 0, т. е. наименьшее возможное; при этом алгоритм не обладает способностью восстанавливать зависимость вне обучающих материалов. Из этого можно сделать вывод: чтобы получить хорошее качество обучения, необходимо обобщать данные, а не только запоминать.

Следовательно, обучающая способность метода μ при условии, что выборки X^i и X^k являются представительными, характеризуется величиной $Q(\mu(X^i), X^k)$. Представительными называются такие выборки X^i и X^k, которые являются простыми и получены из одного и того же вероятностного распределения на множестве X [6].

Если при малых значениях η справедливо неравенство $P_x l_x k\{Q(\mu(X^i), X^k)\} > \eta$, то метод обучения μ называется состоятельным.

Параметр называется точностью, а параметр $(1-\eta)$ – надежностью. Допустима также эквивалентная формулировка: для любых простых выборок X^i и X^k оценки $Q(\mu(X^i), X^k)$ справедлива вероятность $1-\eta$ [1].

В самых разных областях человеческой деятельности встречается множество задач по классификации, регрессии и прогнозированию. Приведем несколько примеров таких задач [4]:

– задачи классификации: распознавание лиц и рукописного текста, антиспам для электронной почты, кредитный скоринг;

– задачи восстановления регрессии: оценка максимальной суммы кредита, оценка объемов продаж;

– задачи ранжирования: ранжирование поисковых запросов, ранжирование в системах коллаборативной фильтрации;

– задачи прогнозирования: сейсмопредсказания, изменение стоимости, нагрузка на call-центр;

– задачи кластеризации: группировка текста, разделение людей на психотипы;

– задачи поиска ассоциаций: анализ рыночных корзин, анализ поведения пользователей.

Так как не существует универсального метода обучения по прецедентам, который будет способен решать практически все задачи одинаково хорошо, то каждый существующий метод обладает своими преимуществами и недостатками, а также границами применения. На практике приходится проводить численные эксперименты с целью подбора оптимального метода для решаемой задачи.

Поэтому существует два типа экспериментальных исследований [6]:

– эксперименты на модели данных. Цель данных экспериментов – выявление границ метода обучения, получения понимания, на что влияют параметры обучения и построения примеров его удачной и неудачной работы;

– эксперименты на реальных данных. Подобные эксперименты решают конкретные прикладные задачи и осуществляют поиск слабых мест в используемом методе, а также границ его применения.

5. ОСНОВНЫЕ МЕТОДЫ МАШИННОГО ОБУЧЕНИЯ ДЛЯ РЕШЕНИЯ ЗАДАЧИ КЛАССИФИКАЦИИ

Подход к задачам обучения – это парадигма, концепция, точка зрения на процесс обучения, которая приводит к поиску комбинаций базовых предположений, эвристик и гипотез, на основе которых строится модель, функционал качества и методы его оптимизации [2].

Разные подходы могут приводить к одной и той же модели, но ее обучение будет проводиться разными способами. Существуют следующие основные подходы [4]:

– классификация с помощью деревьев решений. Алгоритм заключается в делении исходных данных на группы до получения их однородных множеств. Совокупность правил, дающих такое разнообразие, позволяет определять для новых данных наиболее вероятные номера классов, т. е. делать прогноз. Принцип рекурсивного деления используется в методе дерева решений. В узлах, начиная с корневого, выбирается признак, значение которого используется для разбиения всех данных на 2 класса. Процесс продолжается до тех пор, пока не выполнится критерий остановки. Основные недостатки данного метода заключаются в следующем: проблема получения дерева решений является полной задачей; при создании сложной конструкции происходит переобучение; сложное описание концептов; чувствительность к обучающей выборке [11];

– байесовская классификация. Байесовский подход к классификации основан на теореме, утверждающей, что если плотности распределения каждого из классов известны, то искомый алгоритм можно выписать в явном аналитическом виде. Более того, этот алгоритм оптимален, то есть обладает минимальной вероятностью ошибок. На практике плотности распределения классов, как правило, не известны. Их приходится оценивать (восстанавливать) по обучающей выборке. В результате байесовский алгоритм перестает быть оптимальным, так как восстановить плотность по выборке можно только с некоторой погрешностью. Чем короче выборка, тем выше

шансы подогнать распределение под конкретные данные и столкнуться с эффектом переобучения. Байесовский подход к классификации является одним из старейших, но до сих пор сохраняет прочные позиции в теории распознавания. Он лежит в основе многих достаточно удачных алгоритмов классификации. К числу байесовских методов классификации относятся: логистическая регрессия, квадратичный дискриминант, линейный дискриминант Фишера, наивный байесовский классификатор, метод радиальных базисных функций [2];

– классификация с помощью искусственных нейронных сетей (данный метод подробно будет рассмотрен в данной работе);

– классификация с помощью метода опорных векторов. В основе метода опорных векторов лежит идея разделения облаков данных гиперплоскостями, находящихся на максимальном расстоянии от облаков. К достоинствам метода можно отнести эффективную работу при малом размере обучающей выборки, к недостаткам – необходимость ядер для конкретного случая и большое время обучения [12];

– классификация с помощью метода ближайших соседей. Метод ближайших соседей – простейший метрический классификатор, основанный на оценивании сходства объектов. Классифицируемый объект относится к тому классу, которому принадлежат ближайшие к нему объекты обучающей выборки.

Метод ближайшего соседа является, пожалуй, самым простым алгоритмом классификации. Классифицируемый объект x относится к тому классу y_i, которому принадлежит ближайший объект обучающей выборки x_i. В методе ближайших соседей для повышения надежности классификации объект относится к тому классу, которому принадлежит большинство из его соседей – k ближайших к нему объектов обучающей выборки x_i. В задачах с двумя классами число соседей берут нечетным, чтобы не возникало ситуаций неоднозначности, когда одинаковое число соседей принадлежит разным классам.

Метод ближайших соседей в задачах с числом классов 3 и более уже не работает, и ситуации неоднозначности все равно могут возникать. Тогда k-му соседу приписывается вес ω_i, как правило, убывающий с ростом ранга

соседа *i*. Объект относится к тому классу, который набирает больший суммарный вес среди *k* ближайших соседей [4];

– классификация с помощью генетических алгоритмов.

Для решения задач классификации в последнее время зарекомендовали себя нейронные сети. Например, проект Clarifail является одним из лучших в решении задачи классификации изображений и видео и основывается на модели искусственных нейронных сетей. Другим примером по использованию нейронных сетей можно привести российский сервис Findface от компании N-tech.Lab, который по фотографии находит профиль в социальной сети «ВКонтакте». А один из интересных стартапов Ava по фотографии блюда может сообщить количество калорий.

Подобных примеров можно привести много. Нейросетевые технологии для решения задач классификации применяют многие известные компании, такие как Google, Nvidia, Tesla Motors и другие; это показывает, насколько эффективно проявляют себя нейронные сети [18].

Из достоинств нейросетевых алгоритмов можно выделить следующее: эффективная работа с зашумленными данными; адаптация к изменениям окружающей среды; отказоустойчивость, т. е. при повреждении некоторого количества нейронов сеть продолжает функционировать; быстродействие.

6. НЕЙРОСЕТЕВАЯ ИНТЕРПРЕТАЦИЯ ЗАДАЧИ КЛАССИФИКАЦИИ ИЗОБРАЖЕНИЙ

Человеческий мозг образуется сетью нейронов, которая представляет собой параллельную высокоэффективную комплексную систему обработки информации. Данная система распознает образы во много раз быстрее, чем современные вычислительные устройства. Мозг решает задачу распознавания образов эффективнее и принципиально другим образом, в отличие от того, как это делает любая вычислительная система. Именно этот факт побудил ученых к исследованию и воссозданию нейронных сетей.

Искусственная нейронная сеть – это упрощенная модель биологической НС. Реальная нейронная сеть – это скопление множества ячеек, которые называются нейронами, связанных и взаимодействующих между собой различными способами. Искусственная нейронная сеть дублирует лишь основные принципы работы биологических НС [17].

При решении нелинейных задач человеческий мозг, в отличие от современных компьютеров, имеет ряд преимуществ [14]:

– способность к самообучению;
– устойчивость к ошибкам;
– адаптивность;
– массовый параллелизм;
– распределенные методы представления;
– практически полное отсутствие границ допустимой области решений;
– гибкость решения нелинейных проблем.

Определение искусственной нейронной сети как адаптивной машины является наиболее емким [17].

Искусственная нейронная сеть (neural network) – это существенно параллельно распределенный процессор, который обладает способностью к сохранению и репрезентации опытного знания.

Она сходна с мозгом в двух аспектах:

– в процессе обучения знание приобретается сетью;

– силы межнейронных соединений применяются для сохранения знания. Также это называют синаптическими весами.

В 1943 г. нейрофизиолог Мак-Каллок и нейролингвист Уолтер Питтс впервые предложили модель кибернетического нейрона в статье «Исчисление идей, имманентных нервной активности». Формально искусственный нейрон представляет упрощенную модель биологического (рис. 6.1) [16]. Он состоит из тела нейрона (ядро), множества отростков – дендритов, по которым сигнал поступает от других нейронов, и аксона, через который нейрон передает импульс к дендритам другого нейрона. Аксон одного нейрона соединяется с дендритами другого нейрона с помощью синапсов. При превышении некоторого суммарного порогового значения заряда, накопленного в клетке нейрона, происходит возбуждение нейрона. Возбужденный нейрон передает сигнал через аксон и синапсы на другие нейроны [15].

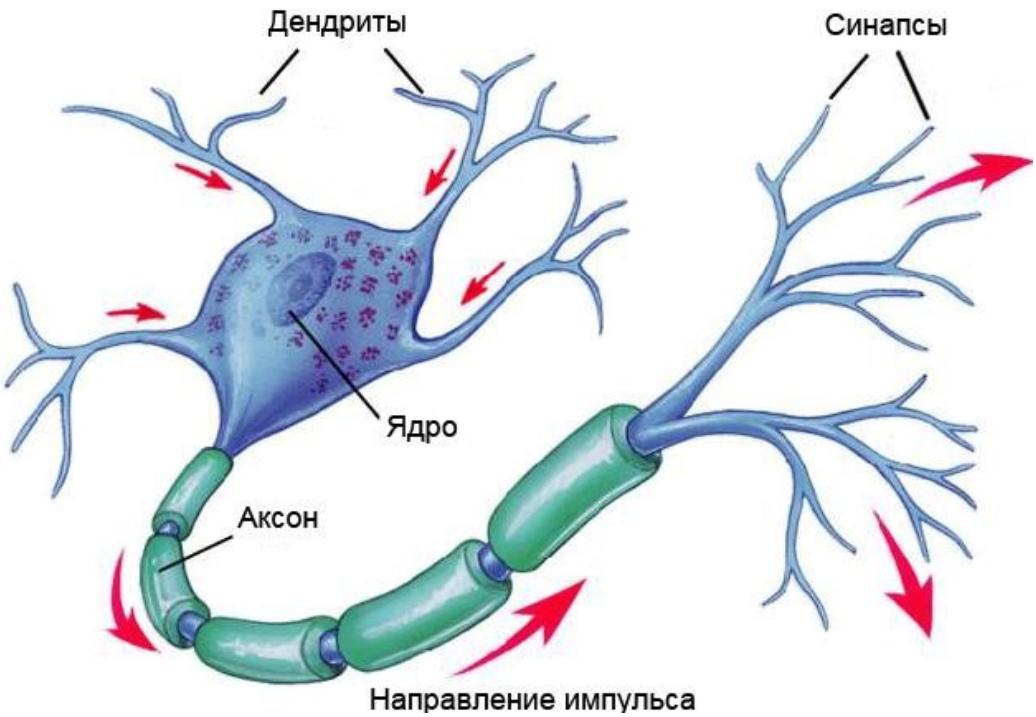

Рис. 6.1. Модель биологического нейрона

Формально искусственный нейрон состоит из трех частей (рис. 6.2): входного сумматора, нелинейного преобразователя и точки ветвления на входе [17].

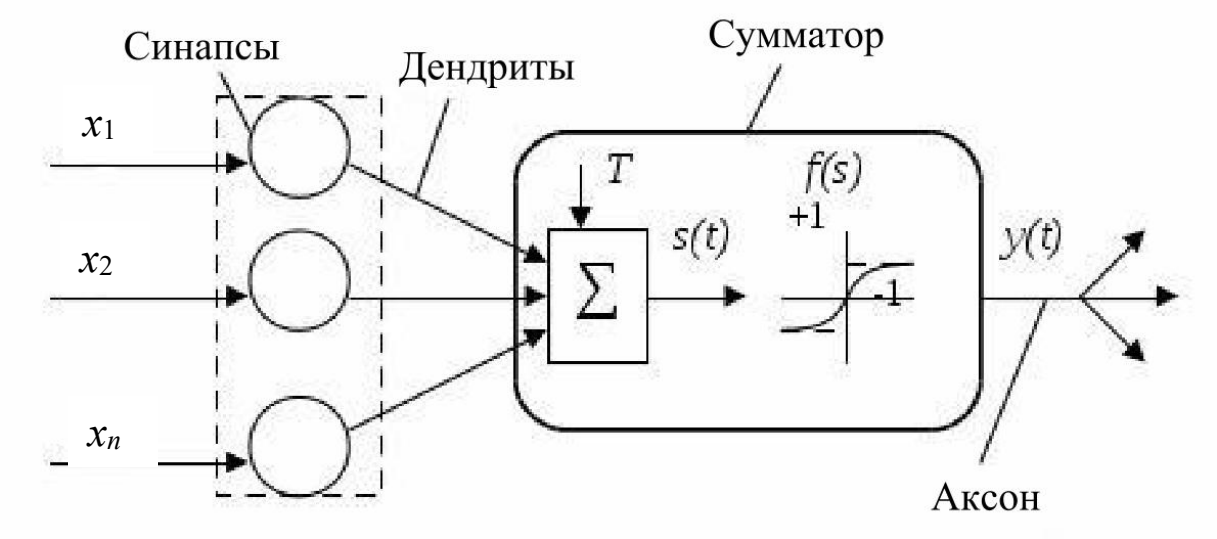

Рис. 6.2. Модель искусственного нейрона

Важной частью нейрона является адаптивный сумматор. Он предназначен для вычисления скалярного произведения вектора входного сигнала x на вектор настраиваемых параметров ω. Далее сигнал проходит через функцию активности. Функция активности, или нелинейный преобразователь, представляет собой скалярный входной сигнал $t = (\omega, x)$.

Сигнал φ передает его на аксон. В качестве функций активаций обычно используется [16]:

– $(1+e^{-t})^{-1}$ – сигмоидная функция;

– $\text{th}(x)$ – гиперболический тангенс;

– $\ln(t+\sqrt{t^2}+1)$ – логарифмическая функция;

– $\text{sgn}(t) = \begin{cases} 1, t > 0 \\ -1, t \le 0 \end{cases}$ – сигнум, или функция знака;

– $\eta(t) = \begin{cases} 1, t > 0 \\ 0, t \le 0 \end{cases}$ – функция Хэвисайда;

– $\varphi(e) = t$ – тождественная функция.

Синапс, или линейная связь, умножает входной сигнал x на вес синапса w.

Для рассылки одного сигнала по другим нейронам используется точка ветвления [14].

Архитектуру искусственной нейронной сети можно представить в виде направленного взвешенного графа, где узлами являются нейроны.

Рассматривая архитектуру нейронных сетей, можно выделить следующие классы [17]:

– сеть прямого распространения, или многослойный персептрон. В этой архитектуре нейронная сеть проставляется последовательными слоями, которые имеют однонаправленную связь между собой. Данные сети являются статическими, т. е. они вырабатывают одну совокупность выходных значений, не зависящих от предыдущего состояния сети;

– сети с обратными связями, или рекуррентные. Такие сети являются динамическими, так как в силу обратных связей в них модифицируются входы нейронов, что приводит к изменению состояния сети.

Фундаментальным свойством мозга является обучение. В контексте искусственных нейронных сетей можно дать следующее понятие обучения: обучение НС – это настройка архитектуры сети и весов связей для эффективного выполнения специальной задачи [16].

Процесс настройки весов или обучение нейронной сети происходит по обучающей выборке. По мере итеративной настройки весовых коэффициентов качество работы нейронной сети улучшается. По сравнению с системами, следующими по определенной совокупности правил, выделенных экспертами, способность нейронных сетей к обучению делают их привлекательными. Для конструирования процесса обучения прежде всего необходимо иметь модель внешней среды, в которой функционирует нейронная сеть, – знать доступную для сети информацию. Эта модель определяет парадигму обучения. Также необходимо понять, каким способом проводить модификацию весовых параметров сети, какие правила обучения управляют процессом настройки. Алгоритм обучения означает процедуру, в которой используются правила обучения для настройки весов.

7. ОБУЧЕНИЕ НЕЙРОННОЙ СЕТИ МЕТОДОМ ОБРАТНОГО РАСПРОСТРАНЕНИЯ ОШИБОК

Алгоритм обратного распространения является одним из самых популярных алгоритмов, который основывается на коррекции ошибок нетренированной сети.

В алгоритме обратного распространения ошибок существует два прохода [14]:

– прямой, при котором данные из обучающей выборки в виде вектора подаются на вход нейронной сети, и распространение сигнала происходит в прямом направлении. После прохождение сигнала через все слои сети генерируется выходной вектор;

– обратный, при котором выходной вектор вычитается из желаемого из учебных данных, в результате чего находится сигнал ошибки, распространяющийся по сети в обратном направлении навстречу синаптическим связям. При обратном проходе вычисляется величина коррекции для весов нейронной сети.

Рассмотрим алгоритм обратного распространения ошибок на примере двухслойного персептрона (рис. 7.1).

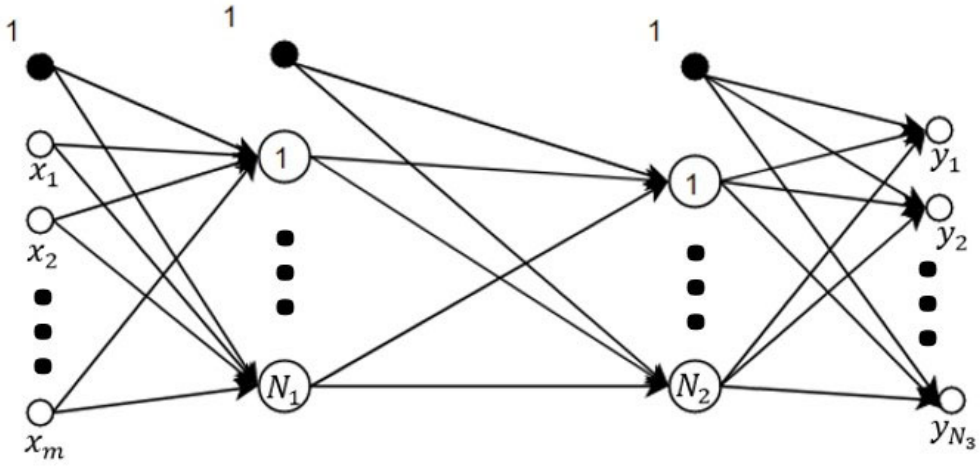

Рис. 7.1. Модель двухслойного персептрона

На начальном этапе обучения веса нейронов инициализированы случайным образом. Для тренировки сети используется обучающая подвыборка из N входных векторов $x_1, x_2, ..., x_N$ и соответствующие желаемые результаты на выходе $d_1, d_2, ..., d_N$. N_1, N_2, N_3 – количество нейронов соответственно в первом, втором и третьем слое. Последовательное обучение подразумевает собой представление по порядку входных векторов один за одним и проведение коррекции весов согласно выбранным правилам. Тогда сигнал ошибки на j-выходном нейроне можно найти по следующей формуле [16]:

$$e_j(n) = d_j(n) - y_i(n). \tag{7.1}$$

Иногда применяется аналогия из термодинамики – энергия ошибки нейрона, определенная как $\frac{1}{2}e_j^2$. Общая энергия ошибки сети будет определена как

$$E(n) = \frac{1}{2}\sum_{j=1}^{N} e_j^2(n). \tag{7.2}$$

Также (7.2) можно рассматривать в качестве функции для минерализации, которая зависит от синаптических связей всей сети. Усредненную энергию по всем образцам учебного набора можно определить как энергию среднеквадратичной ошибки, которую можно рассматривать в качестве критерия эффективности обучения нейронной сети [16]:

$$\bar{E}(n) = \frac{1}{N}\sum_{n=1}^{N} E(n) = \frac{1}{2N}\sum_{n=1}^{N}\sum_{j=1}^{N} e_j^2(n). \tag{7.3}$$

Аргумент функции активации нейрона во втором слое сети на итерации n, т. е. при подаче из обучающей выборки n-го образца определяется как

$$\upsilon_j(n) = \sum_{i=1}^{N} w_{ij}(n) y_i(n), \tag{7.4}$$

где $y_i(n)$ – выходной сигнал предыдущего слоя.

Функциональный сигнал на выходе рассматриваемого нейрона равен

$$y_j(n) = \sigma_j(\upsilon_j(n)). \tag{7.5}$$

Алгоритм обратного распространения заключается в использовании к синаптическому весу $\omega_{ij}(n)$ коррекции $\Delta\omega_{ij}(n)$, которая пропорциональна частной производной $\frac{\partial E(n)}{\partial \omega_{ij}(n)}$.

Для вычисления компонента градиента ошибки можно использовать цепное правило, т. е.

$$\frac{\partial E(n)}{\partial \omega_{ij}(n)} = \frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{y_j(n)}{\partial \upsilon_j(n)} \frac{\upsilon_j(n)}{\partial \omega_{ij}(n)}. \tag{7.6}$$

Стоит учесть, что в (7.12) все обозначения относятся только к окружению *n*-го нейрона. Продифференцировав уравнения (7.3), (7.4), (7.5) и (7.6) по $e_j(n)$, $y_j(n)$, $\upsilon_j(n)$ и $\omega_{ij}(n)$, соответственно получим:

$$\frac{\partial E(n)}{\partial e_j(n)} = e_j(n); \tag{7.7}$$

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1; \tag{7.8}$$

$$\frac{y_j(n)}{\partial \upsilon_j(n)} = \sigma_j^{/}(\upsilon_j(n)); \tag{7.9}$$

$$\frac{\upsilon_j(n)}{\partial \omega_{ij}(n)} = y_i(n). \tag{7.10}$$

Используя все полученные выше выражения, подставим в (7.6)

$$\frac{\partial E(n)}{\partial \omega_{ij}(n)} = -e_j(n)\sigma_j^{/}(\upsilon_j(n))y_i(n). \tag{7.11}$$

Коррекция $\Delta\omega_{ij}(n)$, применяемая к весу $\omega_{ij}(n)$, определяется по дельта-правилу

$$\Delta\omega_{ij}(n) = -\eta \frac{\partial E(n)}{\partial \omega_{ij}(n)}, \tag{7.12}$$

где $-\eta$ является параметром скорости обучения алгоритма обратного распространения ошибок.

Определить направление минимума (7.12) целевой функции можно с помощью знака минус, который указывает направление движения противоположному градиенту.

Подставляя (7.12) в (7.11), получим

$$\Delta\omega_{ij}(n) = \eta\delta_j(n) y_i(n). \tag{7.13}$$

Обычный локальный градиент $\delta_j(n)$ определяется с помощью выражения

$$\delta_j(n) = -\frac{\partial E(n)}{\partial \upsilon_j(n)} = \frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{y_j(n)}{\partial \upsilon_j(n)} = e_j(n)\sigma_j(\upsilon_j(n)). \tag{7.14}$$

Как можно увидеть из этих выражений, сигнал ошибки $e_j(n)$ необходим нам для вычисления величины коррекции. В случае выходного нейрона получить сигнал ошибки несложно, так как желаемый результат известен, и остается только вычислить локальный градиент, используя (7.11) и (7.14), но, чтобы вычислить сигнал ошибки нейрона в скрытом слое, необходимо вычислять рекурсивно ошибки всех связанных с ним нейронов.

Тогда для вычисления локального градиента нейрона в скрытом слое можно использовать выражение [16]

$$\delta_j(n) = -\frac{\partial E(n)}{\partial y_j(n)} \frac{y_j(n)}{\partial \upsilon_j(n)} = -\frac{\partial E(n)}{\partial y_j(n)} \sigma_j(\upsilon_j(n)). \tag{7.15}$$

При вычислении частной производной $\frac{\partial E(n)}{\partial y_j(n)}$ в (7.15), где суммирование производится по нейронам выходного слоя, получим

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial y_j(n)}. \tag{7.16}$$

При использовании цепного правила для производной $\frac{\partial e_k(n)}{\partial y_j(n)}$ данное выражение можно переписать в форму

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial y_j(n)} \frac{\partial \upsilon_k(n)}{\partial y_j(n)}. \tag{7.17}$$

Величина ошибки для *k*-го выходного нейрона составляет

$$e_k(n) = d_k(n) - \sigma_k(\upsilon_k(n)). \tag{7.18}$$

Отсюда

$$\frac{\partial e_k(n)}{\partial \upsilon_j(n)} = -\sigma_k(\upsilon_k(n)). \tag{7.19}$$

Для нейрона в любом слое дальнейшее упрощение выражений не производится несмотря на то, что $\sigma(\upsilon) = \upsilon$ и $\sigma = 1$.

Предполагая, что $y_0(n) = 1$, индуцированное локальное поле нейрона будет равняться [16]

$$\upsilon_k(n) = \sum_{k=0}^{N_2} \omega_{kj}(n) y_j(n). \tag{7.20}$$

Продифференцировав это выражение, получим

$$\frac{\partial \upsilon_k(n)}{\partial y_j(n)} = \omega_{kj}(n). \tag{7.21}$$

Используя (7.18) и (7.19) в (7.20), получим соотношение

$$\frac{\partial E(n)}{\partial y_j(n)} = -\sum_k e_k(n)\sigma_k(\upsilon_k(n))\omega_{kj}(n) = \sum_k \delta_k(n)\omega_{kj}(n). \qquad (7.22)$$

Для локального градиента $\delta_j(n)$ j-го скрытого нейрона получим формулу обратного распространения, подставляя (7.21) в (7.14):

$$\delta_j(n) = \sigma_j(\upsilon_j(n))\sum_k \delta_k(n)\omega_{kj}(n). \qquad (7.23)$$

Собрав все воедино, для реализации алгоритма обратного распространения ошибки можно определить дельта-правило

$$\Delta\omega_{ji}(n) = \eta\delta_j(n)y_j(n), \qquad (7.24)$$

где $\Delta\omega_{ji}(n)$ – коррекция веса; η – параметр скорости обучения; $\delta_j(n)$ – локальный градиент; $y_i(n) = 1$ – входной сигнал j-го нейрона [18].

Стоит учесть, что локальный градиент $\delta_j(n)$ будет зависеть от положения в нейронной сети:

– j-й нейрон является выходным. В данном случае локальный градиент $\delta_j(n)$ согласно (7.22) будет равняться произведению производной $\sigma_j(\upsilon_j(n))$ на ошибку $e_j(n)$;

– j-й нейрон является скрытым. Локальный градиент $\delta_j(n)$ можно найти путем умножения производной $\sigma_j(\upsilon_j(n))$ на взвешенную сумму градиентов, которые вычисляются для нейронов, связанных с данным.

Каждая итерация алгоритма обратного распространения ошибки заключается в двух этапах [18]:

– прямой проход: $y_i(n) = \sigma_j(\upsilon_j(n))$, где $\upsilon_i(n) = \sum\omega_{kij}(n)y_i(n)$. $y_i(n) = x_i(n)$, если нейрон находится в первом скрытом слое, и $y_i(n) = \sigma_j(n)$, где $\sigma_j(n)$ – на выходном терминале сети, если нейрон является выходным. Прямой проход подает на вход очередной вектор x и заканчивается, вычисляя ошибку на выходе $e_n = d_n - \sigma_n$;

– обратный проход, в котором на выходной слой подается сигнал ошибки и при движении справа налево параллельно производится вычисление локального градиента. При движении в обратном направлении осуществляется коррекция весов, вычисляется локальный градиент, который в дальнейшем используется для вычисления градиента последующего (при обратном распространении) слоя нейронов.

К функции активации в нейронных сетях при использовании алгоритма обратного распространения ошибки предъявляется требование дифференцируемости [20]. Так как через значение самой функции можно выразить производную функции активации, это позволяет при прямом проходе не производить непосредственное вычисление производной функции, а использовать значения функции, вычисленные при прямом проходе [22].

Алгоритм обратного распространения строит в пространстве весов некоторую траекторию аппроксимации, которая вычисляется с помощью метода наискорейшего спуска. При меньшем параметре обучения η корректировка синаптических связей производится меньше, а траектория в пространстве весов становится более гладкой. Но чрезмерное уменьшение этого параметра приводит к замедлению процесса обучения, а использование слишком большого параметра приводит к неустойчивости алгоритма, и минимизировать ошибку не получится. Обобщенное дельта-правило позволяет повышать скорость обучения без потерь устойчивости и является модернизированным дельта-правилом (7.13) путем использования «момента инерции»

$$\Delta\omega_{ij}(n) = \alpha\omega_{ij}(n-1) + \eta\delta_j(n)y_i(n), \qquad (7.25)$$

где α – постоянный момент, в большинстве случаев положительный.

С точки зрения теории оптимизации алгоритм обратного распространения может рассматриваться как специфическая реализация градиентного метода минимизации целевой функции. А обобщенное дельта-правило может рассматриваться как метод регуляризации решения задачи минимизации [16].

8. ЗАДАЧИ, РЕШАЕМЫЕ С ПОМОЩЬЮ НЕЙРОННЫХ СЕТЕЙ

На данном этапе развития НС хорошо зарекомендовали себя в решении таких задач машинного обучения, как [17]:

– классификация образов. Задача состоит в указании принадлежности входного образа (например, речевого сигнала или рукописного символа), представленного вектором признаков, одному или нескольким предварительно определенным классам. К известным приложениям относятся распознавание букв, распознавание речи, классификация сигнала электрокардиограммы, классификация клеток крови. В робототехнике одним из основных приложений является распознавание объектов из видеоинформации, полученной от системы технического зрения [16];

– аппроксимация функций. Предположим, что имеется обучающая выборка $((x_i, y_i),..,(x_n, y_n))$ (пары данных «вход – выход»), которая генерируется неизвестной функцией (x), искаженной шумом. Задача аппроксимации состоит в нахождении оценки неизвестной функции (x). Аппроксимация функций необходима при решении многочисленных инженерных и научных задач моделирования;

– кластеризация / категоризация. При решении задачи кластеризации, которая известна также как классификация образов «без учителя», отсутствует обучающая выборка с метками классов. Кластеризация – метод, применяющийся для анализа больших наборов данных, заключающийся в разбиении всего множества на группы близкородственных элементов (кластеры). Кластеризация может быть использована для решения таких задач, как обработка изображений, классификация, тематический анализ коллекций документов, построение репрезентативной выборки. Известны случаи применения кластеризации для извлечения знаний, сжатия данных и исследования свойств данных;

– предсказание / прогноз. Пусть заданы n дискретных отсчетов $\{y(t_1), y(t_2), ..., y(t_n)\}$ в последовательные моменты времени $t_1, t_2, ..., t_n$. Задача состоит в предсказании значения $y(t_n + 1)$ в некоторый будущий мо-

мент времени $t_n + 1$. Предсказание / прогноз имеют значительное влияние на принятие решений в бизнесе, науке и технике. Предсказание цен на фондовой бирже и прогноз погоды являются типичными приложениями техники предсказания / прогноза;

– ассоциативная память. В модели вычислений фон Неймана, которая используется в современных вычислительных машинах, обращение к памяти доступно только посредством адреса, который не зависит от содержания памяти. Более того, если допущена ошибка в вычислении адреса, то может быть найдена совершенно иная информация. Ассоциативная память, или память, адресуемая по содержанию, доступна по указанию заданного содержания. Содержимое памяти может быть вызвано даже по частичному входу или искаженному содержанию. Ассоциативная память чрезвычайно желательна при создании мультимедийных информационных баз данных [18];

– оптимизация. Многочисленные проблемы в математике, статистике, технике, науке, медицине и экономике могут рассматриваться как проблемы оптимизации. Задачей алгоритма оптимизации является нахождение такого решения, которое удовлетворяет системе ограничений и максимизирует или минимизирует целевую функцию. Задача коммивояжера, относящаяся к классу NP-полных, является классическим примером задачи оптимизации.

9. РЕШЕНИЕ ЗАДАЧИ КЛАССИФИКАЦИИ ИЗОБРАЖЕНИЙ С ПОМОЩЬЮ СВЕРТОЧНОЙ НЕЙРОННОЙ СЕТИ

Задача классификации изображения заключается в приеме картинки на вход некоего классификатора, а на выходе получать класс, соответствующий объекту на изображении, или вектор вероятностей, который лучше всего классифицирует изображение. В качестве классификатора может выступать как человек, так и машина. В отличие от человека, компьютер видит изображение в виде массива пикселей, а точнее, в виде массива чисел, характеризующих каждый пиксель. Для человека эти цифры бессмысленны, но для компьютера это единственные данные для распознавания [19].

В машинном обучении одним из самых распространенных способов классификации объекта на изображении является признаковое описание объекта, характеризующее каждый объект набором признаков. Но стоит учесть, что не для всех данных открытые признаки не дают хорошей точности классификации. Такими признаками могут являться цифровой звуковой сигнал или цвет точек. Машине гораздо сложнее производить классификацию изображений котов или машин, чем человеку, так как эти данные имеют скрытые признаки, которые тяжело понять машине, в отличие от человека [19].

Глубокое обучение – это набор алгоритмов машинного обучения. Данный набор методов пытается моделировать высокоуровневые абстракции в данных, т. е. старается выделить из данных скрытые признаки [3].

Автоэнкодер – один из алгоритмов машинного обучения без учителя, в котором размер выходного вектора равен входному. Нейронная сеть прямого распространения без обратных связей является одной из распространенных архитектур автоэнкодера и состоит из входного слоя и выходного, равных по количеству нейронов и скрытых слоев. Данные, полученные на входном слое, сжимаются с помощью скрытого слоя, зачем происходит восстановление на выходном слое, благодаря чему выделяются скрытые признаки [22].

Но автоэнкодер имеет существенный недостаток, а именно: его стоит применять на данных небольшого размера, так как при использовании больших данных процесс обучения существенно замедлится. В качестве примера можно рассмотреть классификацию картинок размером $m \times n = 8 \times 8$. Мы хотим выделить 50 признаков. Тогда размер входного слоя составит $8 \times 8 = 36$ нейронов, соответственно, размер выходного слоя тоже будет равен 36 нейронам (из определения автоэнкодера). Если мы хотим выделять 50 признаков, то и размер скрытого слоя будет составлять 50 нейронов. Соответственно число весов равно $2 \times m \times n \times k = 2 \times 8 \times 8 \times 50 = 6\,400$. Если же размер изображений для классификации будет составлять 100×100, то количество связей увеличится до 100 000 [22].

Для снижения зависимости размера нейронной сети от размера входного изображения используется метод сверточных нейронных сетей. Сверточные нейронные сети были предложены Яном Лекуном на основе исследования зрительной коры кошек, в ходе которого были открыты простые клетки, реагирующие на прямые линии под разным углом, и сложные, реагирующие только в случае комбинации нескольких простых клеток. Отсюда и появилась идея сверточных нейронных сетей, которая заключается в чередовании сверточных и субдискретизирующих слоев [8].

Сверточная нейронная сеть отличается от обычного персептрона тем, что каждый нейрон слоя связан не со всеми нейронами предыдущего слоя, а только с его частью. В данном типе сетей для операции свертки используется небольшая матрица, которая передвигается по обрабатываемому слою с некоторым шагом смещения. Для первого слоя данная матрица двигается по входному изображению.

После каждого сдвига формируется сигнал для активации нейрона следующего слоя и с соответствующей позиции. Такая матрица называется ядром свертки или фильтром, она производит кодирования какого-либо признака. Следующий слой благодаря этой матрице показывает наличие того или иного признака в предыдущем обрабатываемом слое и его координаты; тем самым формируется карта признаков. В сверточных нейронных сетях используется не одна матрица весов, а набор, характеризующий тот или иной признак. Ядра свертки формируются автоматически во время обучения, например, методом обратного распространения ошибки. Каждый

набор весов при проходе формирует свой признак. Таким образом сеть становится многоканальной [12].

Для уменьшения размерности карты признаков используется операция субдискретизации, а слой сети в данном случае называется слоем субдискретизации, или пулинга. Для задачи классификации изображений информация о присутствии признака важнее, чем его положения, поэтому слой пулинга из нескольких соседних нейронов выбирает максимальный или находит их среднее значение весов. Получившийся нейрон подставляется в новую уплотнению карту признаков (рис. 9.1) [19].

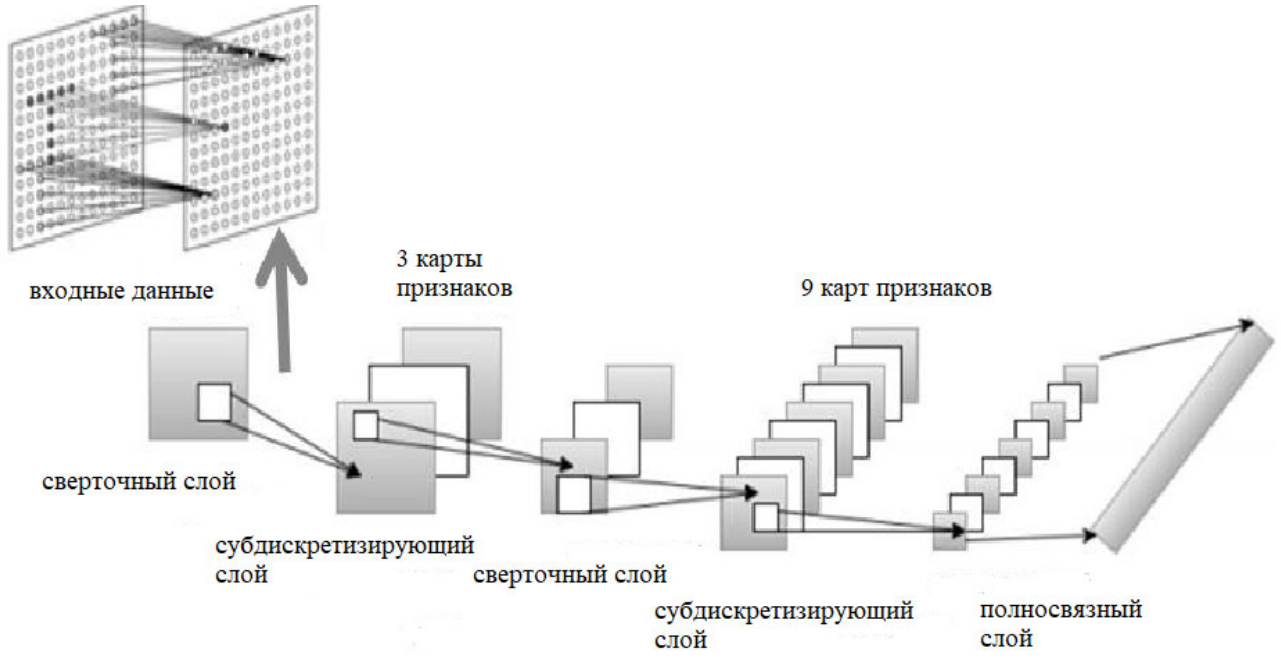

Рис. 9.1. Модель сверточной нейронной сети

Рассмотрим архитектуру сверточной нейронной сети подробнее. Она состоит из множества слоев. Первоначально идет входной слой, который является изображением для классификации и представляет собой матрицу цифр, описывающих каждый пиксель изображения. После входного слоя сигнал поочередно проходит через слои свертки и субдискретизации, которые чередуются. Наличие слоев свертки позволяет составлять новые карты признаков уже по имеющимся, а слой пулинга позволяет уменьшать размер этих слоев.

Данное чередование слоев позволяет распознавать сложные иерархические признаки. После прохождения нескольких слоев свертки и субдискретизации карта признаков обычно превращается в вектор признаков, но таких векторов становятся сотни. Перед выходным слоем в сверточной нейронной сети находится обычный многослойный персептрон, на вход которому подается карта признаков. Этот персептрон и решает задачу классификации по карте признаков [19].

Сверточная нейронная сеть обычно состоит из следующих слоев [20]:

– слой свертки (рис. 9.2), который служит для составления карт признаков;

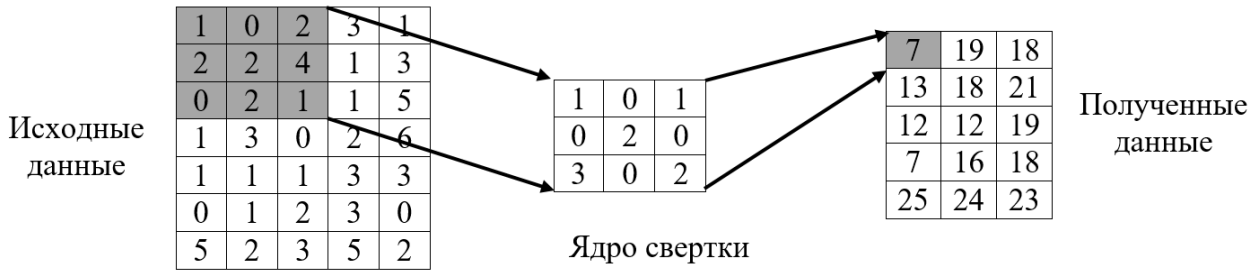

Рис. 9.2. Слой свертки

– ReLU-слой – блок линейной ректификации, используется в качестве функции активации после сверточного слоя и представляет собой функцию вида $f(x) = \max(0, x)$;

– слой субдискретизации, или пулинга (рис. 9.3). Служит для сокращений размерности карт признаков;

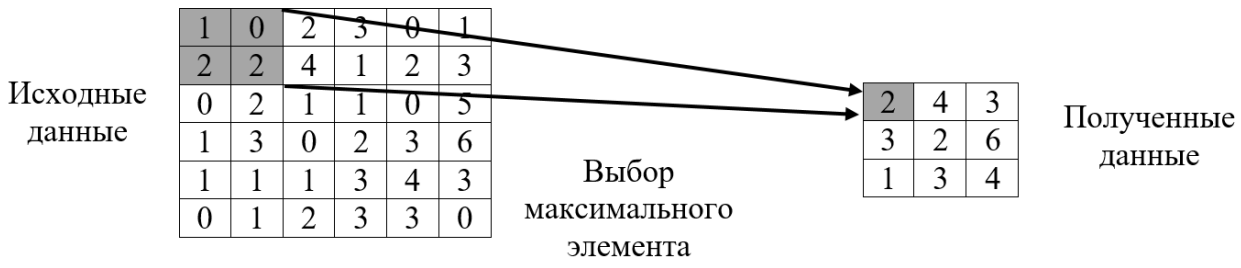

Рис. 9.3. Слой субдискретизации

– полносвязанный слой – слой, который по карте признаков определяет, к какому классу относится наблюдаемый объект, и на выходе выдает вектор вероятностей принадлежности объекта к тому или иному классу;

– dropout-слой. Используется для борьбы с переобучением при обучении стохастическим градиентным спуском. Работа данного слоя заключается в модернизации структуры сети путем выбрасывания нейрона с некоторой вероятностью.

Говоря о преимуществах сверточной нейронной сети, можно выделить следующее [21]:

– уменьшение количества параметров и повышение скорости обучения сети, в отличие от многослойного персептрона;

– устойчивость к помехам, искажениям, сдвигам и поворотам входного изображения;

– возможность распараллеливания вычислений и реализации алгоритмов обучения сети на графических процессорах (GPU), что сокращает время обучения нейронной сети;

– на данный момент является одним из самых лучших и зарекомендовавших себя алгоритмов для решения задачи классификации изображений.

К недостаткам алгоритма можно отнести только большое количество настраиваемых параметров.

При решении задачи классификации символов на изображении на вход сверхточной нейронной сети подается массив, описывающий изображение символа. После прохода сигнала через сеть на выходе формируется вектор; в случае если распознавание проводится только по английскому алфавиту и цифрам, то размер выходного вектора равен 62. В выходном векторе показывается вероятность отношения символа к какому-либо классу. Как уже говорилось, для обучения обычно применяется метод обратного распространения ошибок.

10. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ ЗАДАЧ РАСПОЗНАВАНИЯ ОБРАЗОВ, ИССЛЕДОВАНИЯ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ, УПРАВЛЕНИЯ И НАБЛЮДЕНИЯ МЕТОДАМИ НЕЧЕТКОЙ ЛОГИКИ

10.1. Лабораторная работа № 1. Создание нейронной сети с помощью Neuroph Studio

Время выполнения – 2 часа.

Цель работы: научиться создавать искусственную нейронную сеть с помощью Neuroph Studio, задавать ее параметры и проводить обучение.

Задачи работы

1. Изучить основное меню приложения *Neuroph Studio*.
2. Создавать проекты в *Neuroph Studio*.
3. Создавать сети персептронов.
4. Создавать тренировочные наборы данных.
5. Осуществлять тренировку сети.

Перечень обеспечивающих средств

Для выполнения работы необходимы:

– конспект лекций;

– персональный компьютер с установленными на нем приложениями *Microsoft Office Word* и *Neuroph Studio*.

Методика выполнения работы

Создание проекта в Neuroph Studio

1. Для начала работы необходимо запустить приложение *Neuroph Studio*. Окно приложения показано на рис. 10.1.

2. Для создания проекта в приложении *Neuroph Studio* необходимо:

– выбрать *File* → *New Project*. В результате появится диалоговое окно (рис. 10.2);

– выбрать *Neuroph Project* и нажать на *Далее*.

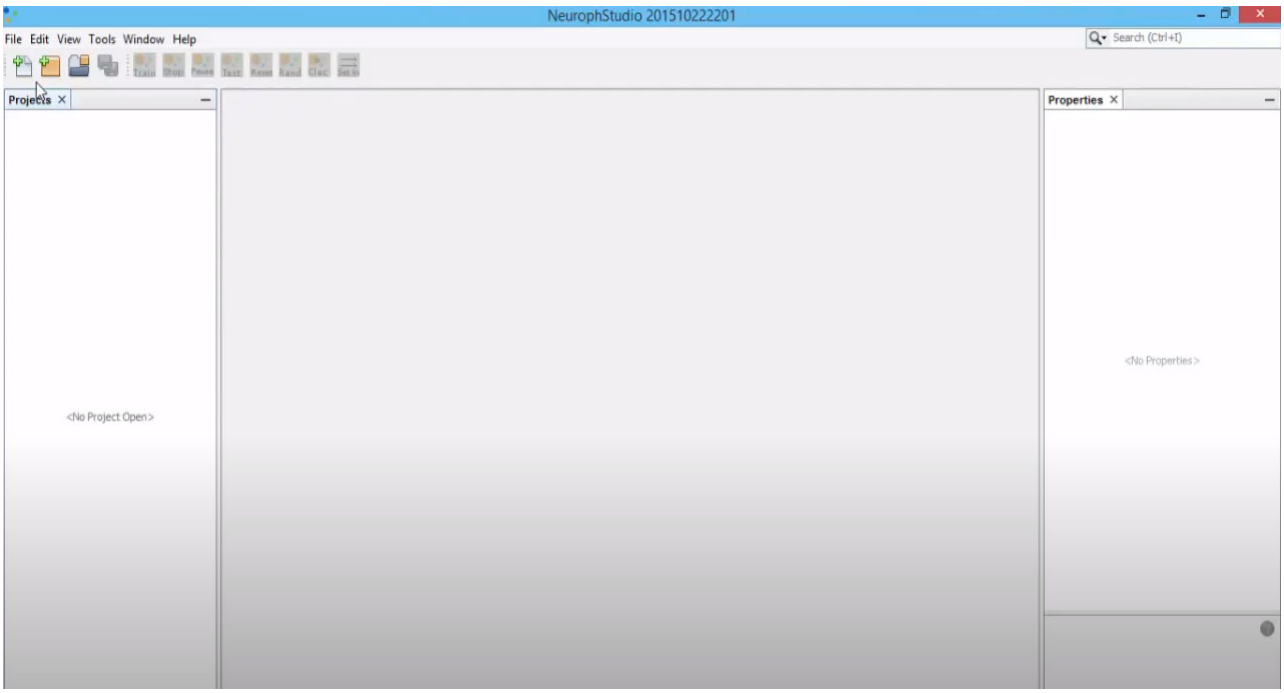

Рис. 10.1. Окно приложения *Neuroph Studio*

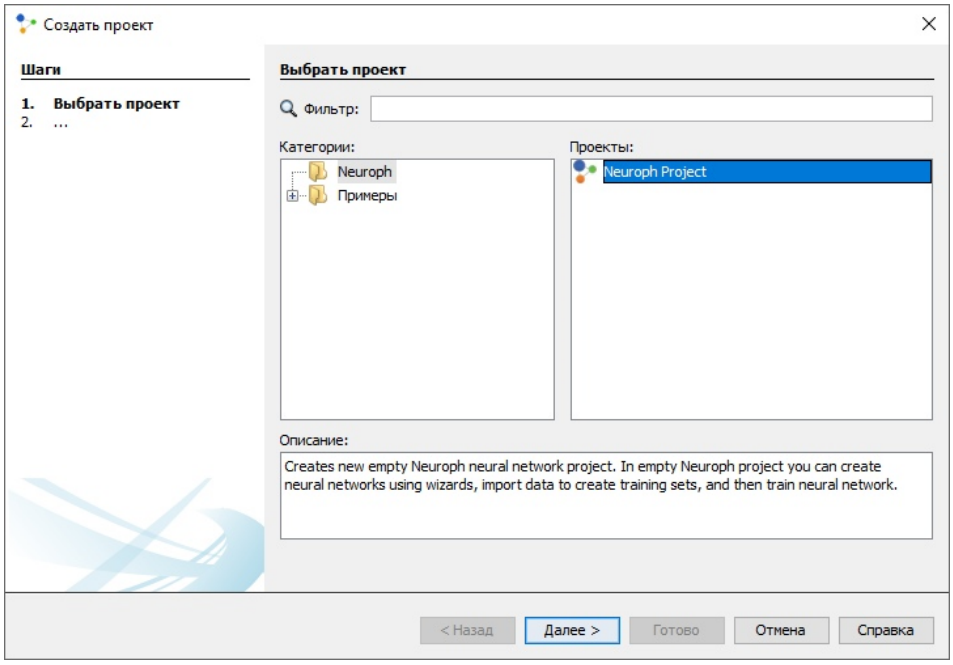

Рис. 10.2. Создание проекта в приложении *Neuroph Studio*

3. Затем требуется ввести имя проекта, указать его расположение и нажать *Готово* (рис. 10.3). В результате будет создан проект, в который можно добавить нейронную сеть.

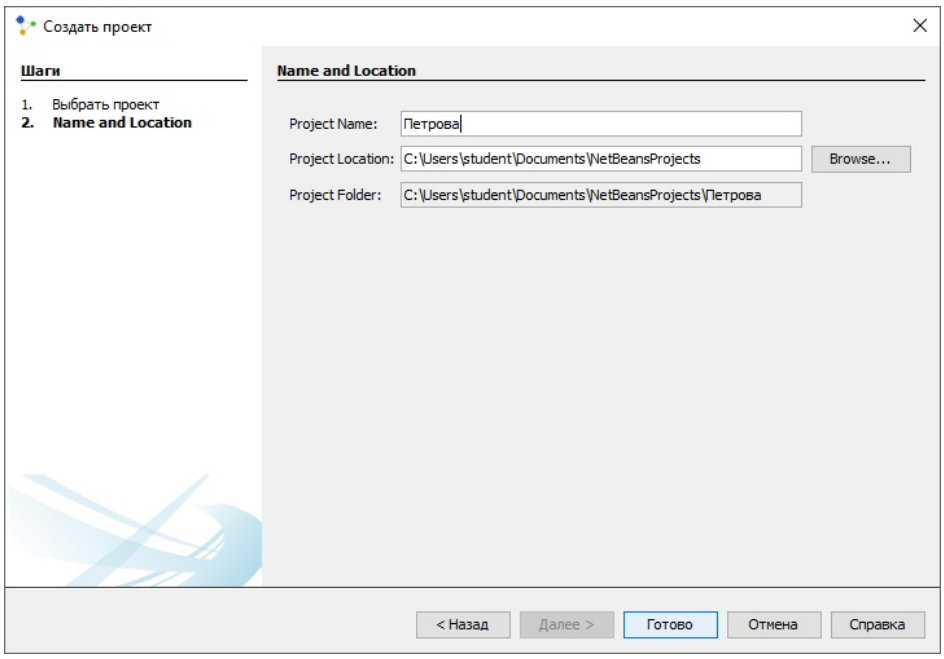

Рис. 10.3. Пункт основного меню *File*

Создание сети персептронов

1. Чтобы создать сеть персептронов, нужно выбрать новый файл. Это делается с помощью команды *File* → *New File*.

2. В окне создания требуется выбрать нужный проект, категорию *Neuroph* и тип файла *Neural Network* (рис. 10.4).

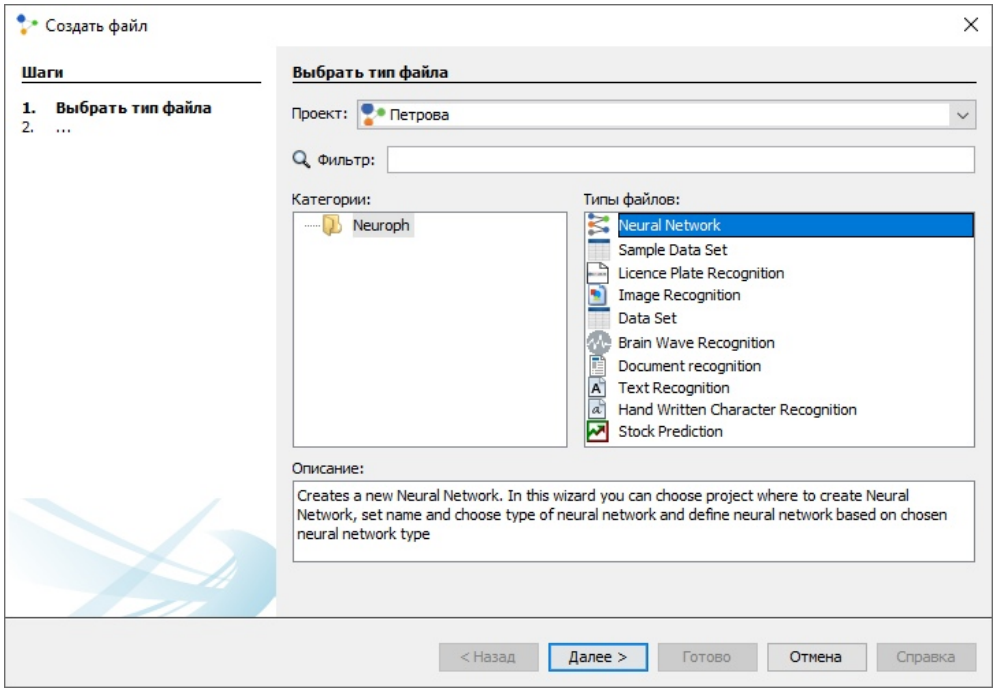

Рис. 10.4. Выбор типа файла *Neural Network*

3. Затем необходимо ввести имя сети и выбрать тип *Perceptron Network* (рис. 10.5).

4. В новом диалоге настройки персептронов вводим число входных нейронов *Inputs Num* – 1 и выходных *Outputs Num* – 2, выбираем обучение персептронов *Perceptron Learning* и нажимаем *Готово* (рис. 10.6).

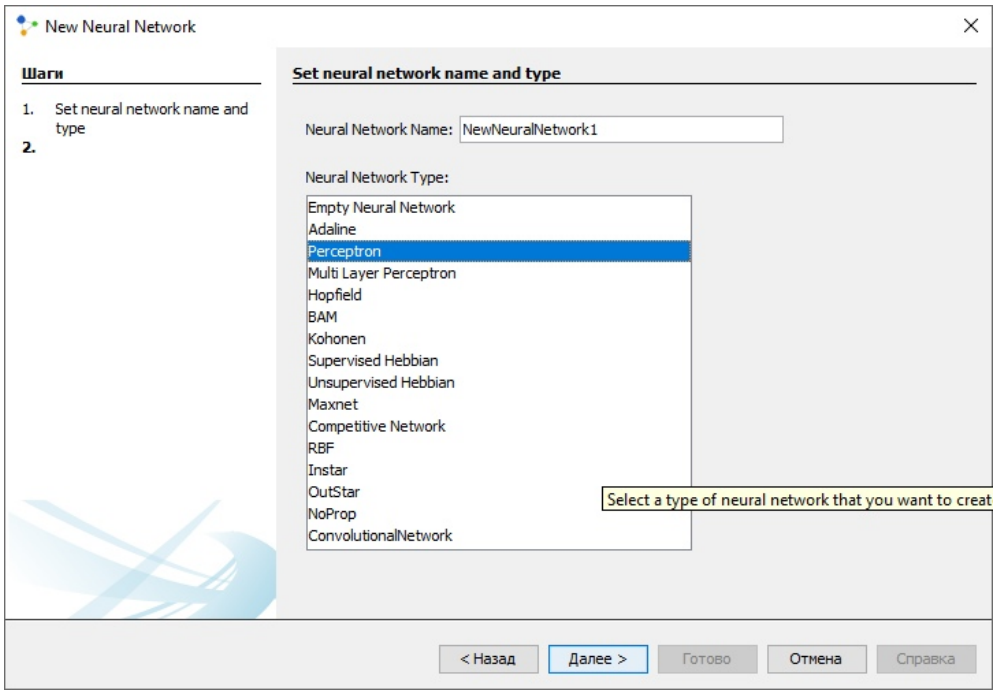

Рис. 10.5. Название сети и выбор типа сети

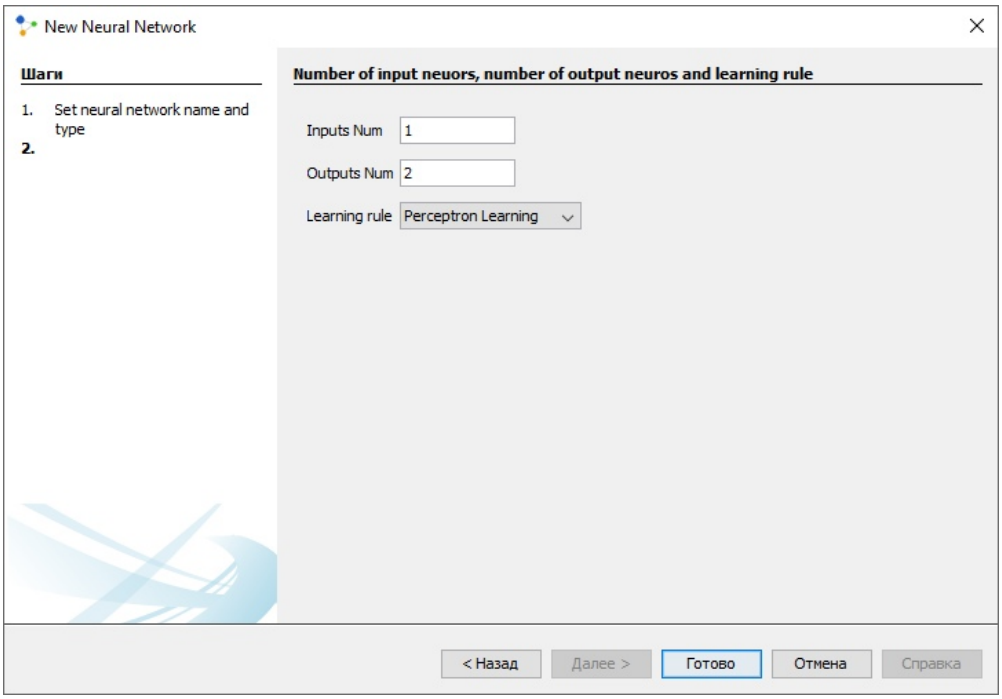

Рис. 10.6. Настройка персептронов

5. Нейронная сеть персептронов с двумя нейронами на входе и одним на выходном слое создана (рис. 10.7).

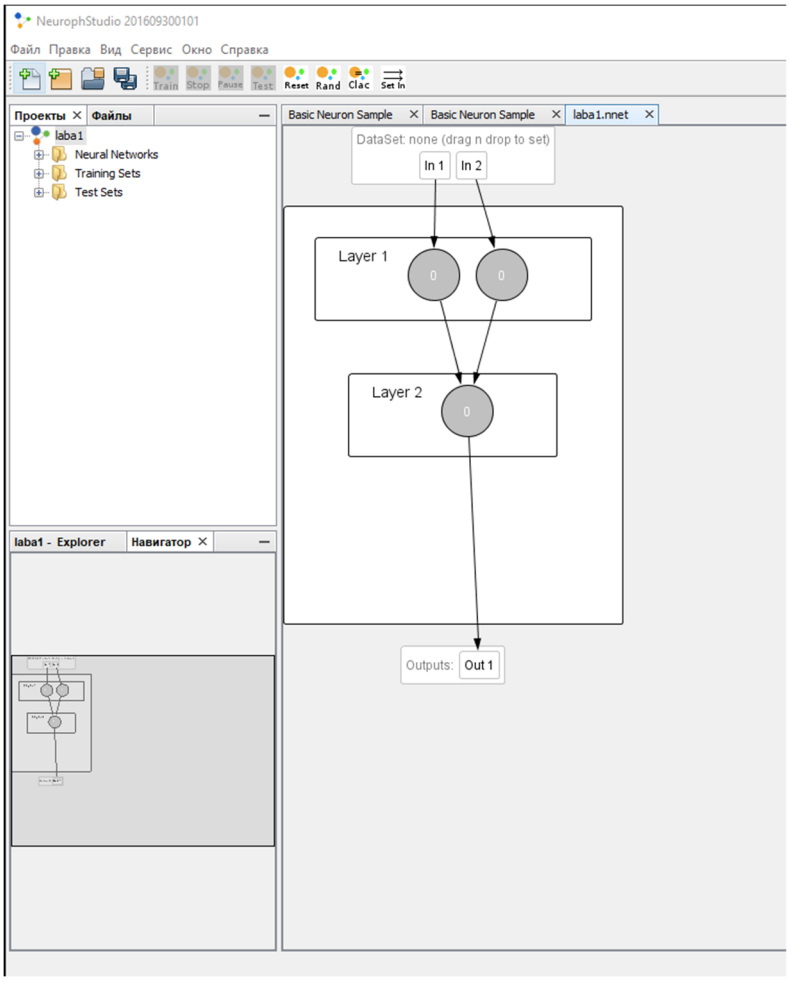

Рис. 10.7. Сеть персептронов

Создание тренировочного набора данных

1. Тренировочный набор создается с помощью мастера тренировки. Мастер тренировки запускается с помощью команды *File* → *New File*. Выбираем тип набора тренировочных данных *Data set file type*, затем нажимаем *Далее* (рис. 10.8).

2. Далее необходимо ввести имя тренировочного набора данных, число входов и число выходов. После создания тренировочного набора нужно ввести значения на входах и выходах нейронов (рис. 10.9).

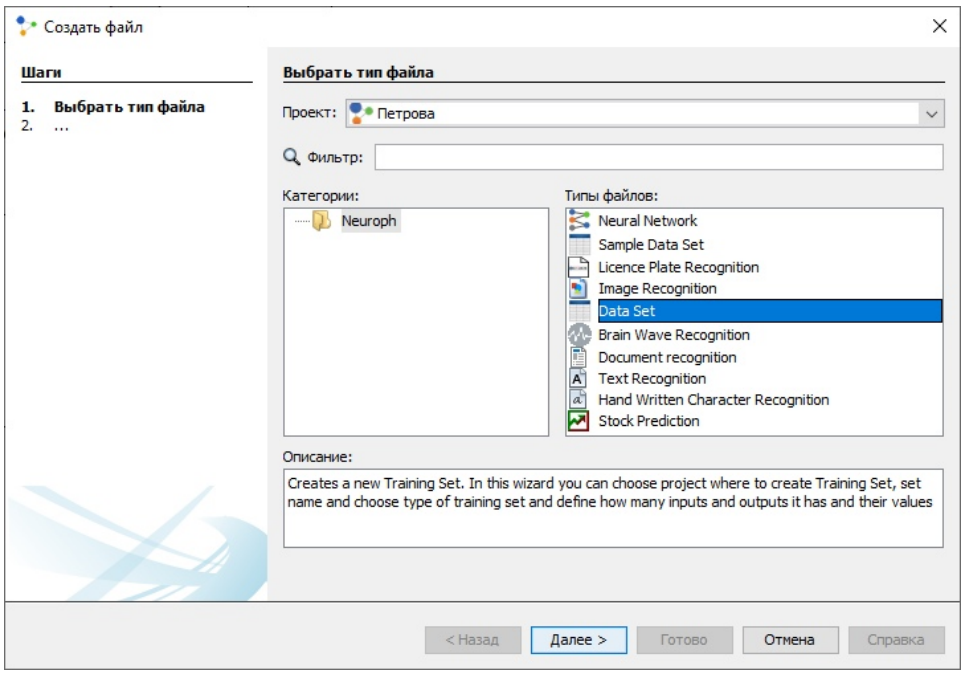

Рис. 10.8. Выбор типа набора тренировочных данных

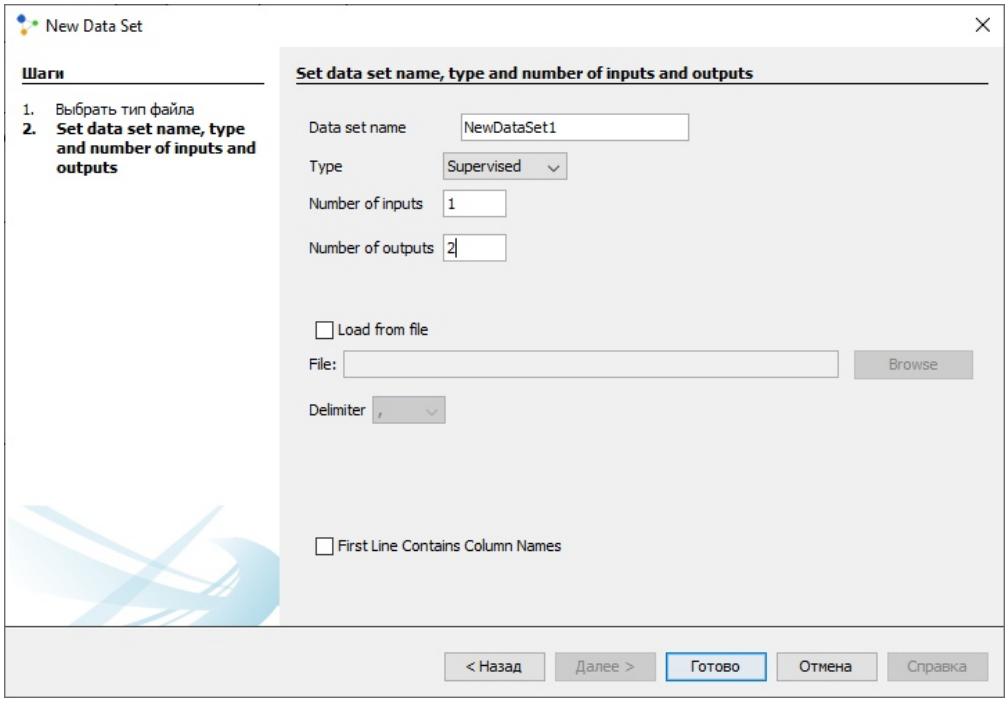

Рис. 10.9. Название тренировочного набора и число входов и выходов

Для добавления новых элементов нужно использовать кнопку *Add row*, а после ввода всех элементов нажать кнопку *Готово* (рис. 10.10).

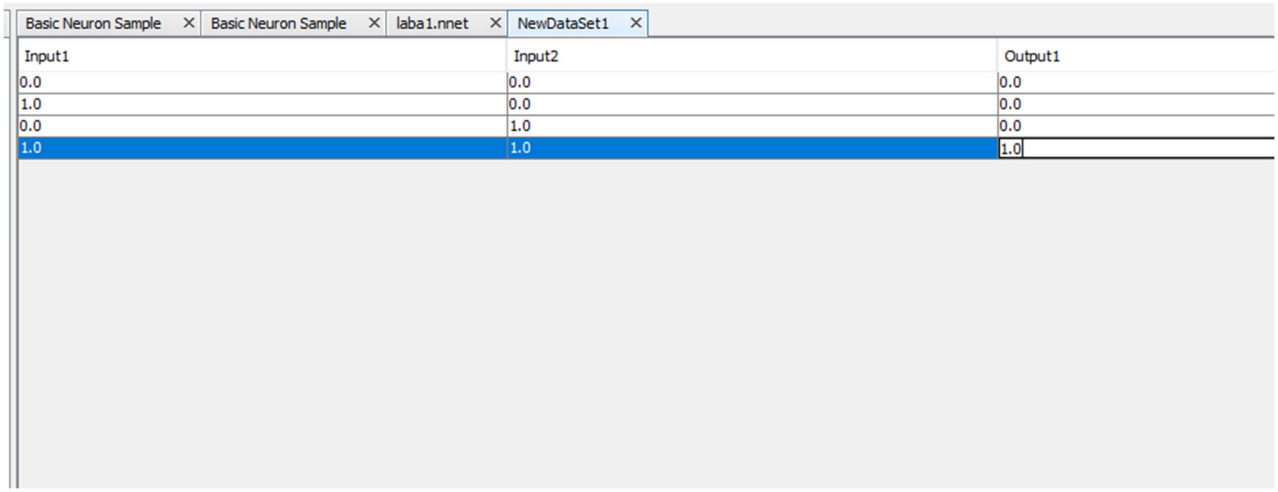

Рис. 10.10. Создание тренировочного набора

Тренировка сети

1. Для тренировки сети был выбран набор, создание которого описано в предыдущем пункте. После выбора тренировочного набора необходимо нажать кнопку на *Train*. Тренировка будет завершена, когда число ошибок сети станет равно нулю. Для этого в диалоге установки параметров обучения *Set Learning parameters* используйте значения по умолчанию и снова нажмите на кнопку *Train*. Когда число ошибок сети *Total Net Error* станет равным нулю, тренировка будет завершена (рис. 10.11).

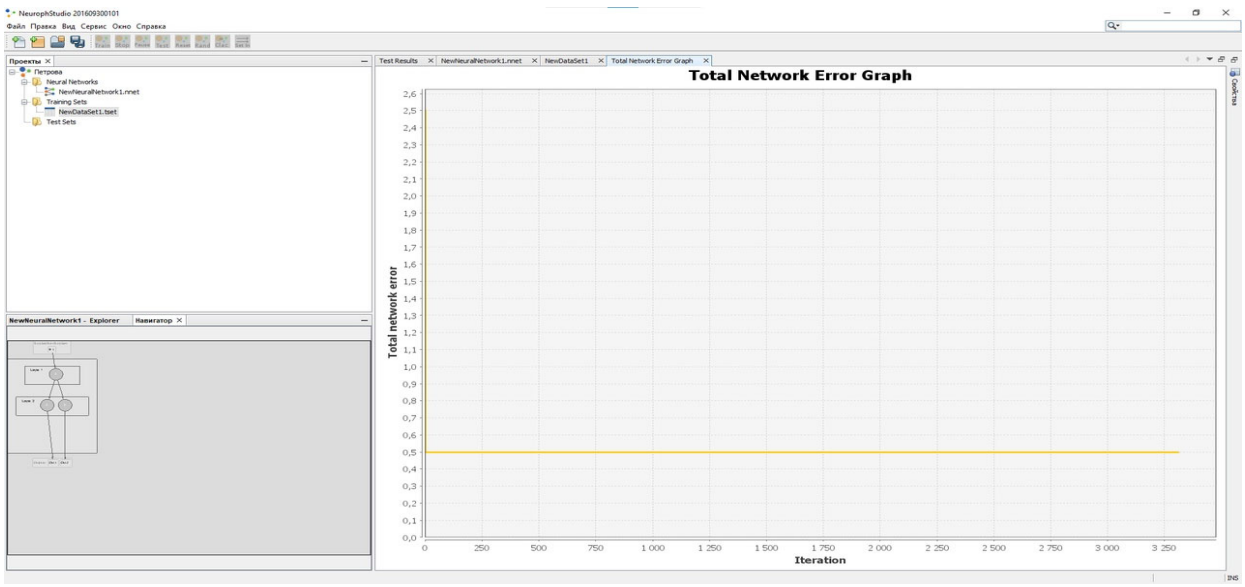

Рис. 10.11. Результат тренировки сети

2. После того как тренировка будет завершена, можно протестировать нейронную сеть тем же набором тренировочных данных. Для этого требуется выбрать его и нажать на кнопку *Test*. Результаты теста будут открыты в новом окне.

Для тестирования единичным набором значений используем кнопку *Set Input button*. Это откроет диалог ввода значений *Set Network Input*, в который могут быть введены значения, разделенные пробелом (рис. 10.12).

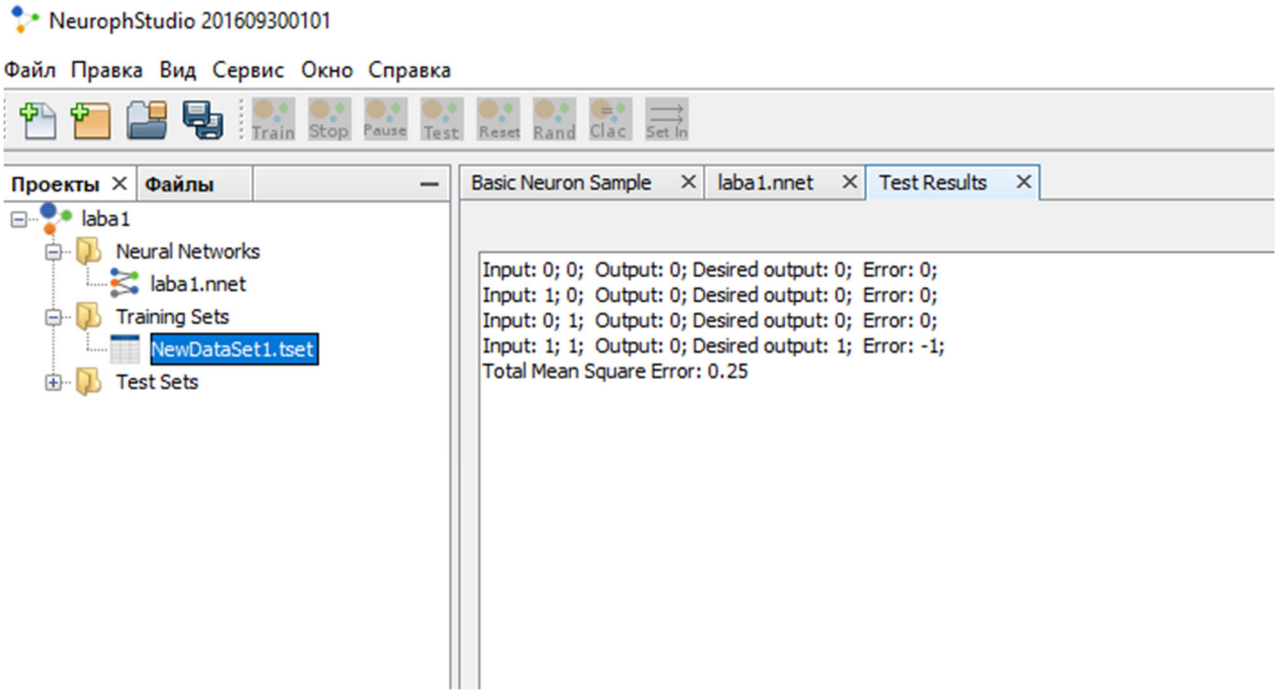

Рис. 10.12. Результат теста нейронной сети

Контрольные вопросы

1. Как осуществляется создание проекта в *Neuroph Studio*?
2. Опишите процесс создания сети персептронов в *Neuroph Studio*.
3. Какие настройки есть в параметрах при создании персептронов?
4. Как осуществляется создание тренировочного набора данных?
5. Как осуществляется тренировка сети?

10.2. Лабораторная работа № 2.
Создание многослойной нейронной сети

Время выполнения – 2 часа.

Цель работы: научиться создавать многослойные нейронные сети с помощью программного комплекса *Neuroph Studio*.

Задачи работы

1. Последовательно выполнить инструкции задания.
2. Найти ответы на контрольные вопросы.

Перечень обеспечивающих средств

Для выполнения работы необходимы:

– конспект лекций;

– персональный компьютер с установленными на нем приложениями *Microsoft Office Word* и *Neuroph Studio*.

Методика выполнения работы

1. Запустите приложение *Neuroph Studio*.
2. Создайте проект, как в прошлой лабораторной работе (рис. 10.13).

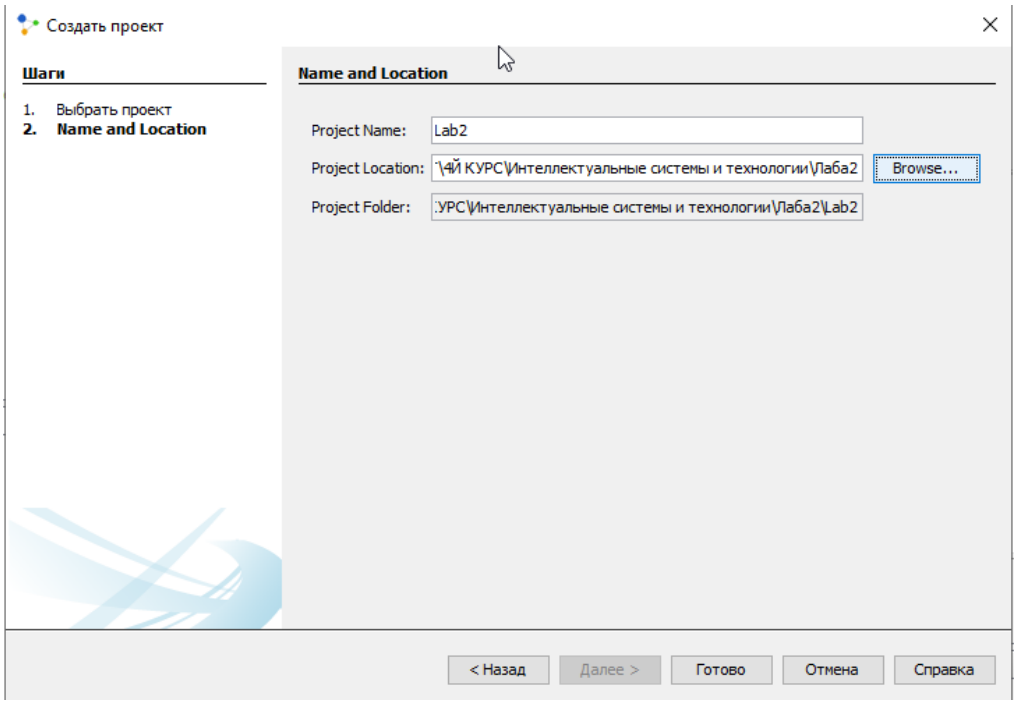

Рис. 10.13. Создание проекта

3. Создайте сеть персептронов. Для этого выполните команду *File* → *New File*.

4. Из выпадающего окна меню выберите тип файла *Neural Network*.

5. Введите имя сети *Lab2* и выберите тип *Multi Layer Perceptron network*.

6. В настройках введите число входных нейронов *Input* – 2, число скрытых нейронов *Hidden* – 3 и выходных *Output* – 1 (рис. 10.14).

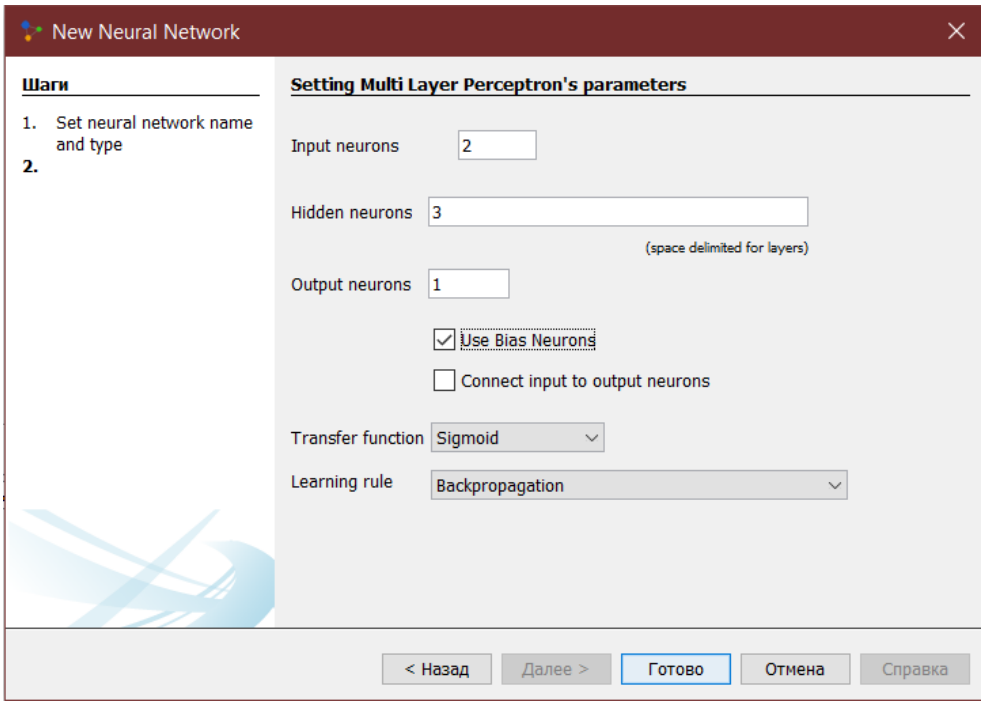

Рис. 10.14. Создание сети

7. Оставьте включенным переключатель *Use Bias Neurons* и выберите функцию передачи *Sigmoid*, а правило обучения персептронов – *Backpropagation* и нажмите на *Готово*.

В результате получится многослойная нейронная сеть, в которой два нейрона на входе, три скрытых и один на выходе (рис. 10.15).

8. Создание тренировочного набора данных происходит с помощью команды *File* → *New File*. Выберите тип набора тренировочных данных *Data set file type* и затем введите имя тренировочного набора данных *NewDataSet1*. В итоге создан тренировочный набор данных.

Используя кнопку *Add row*, добавьте новые элементы и нажмите *OK* по завершении (рис. 10.16).

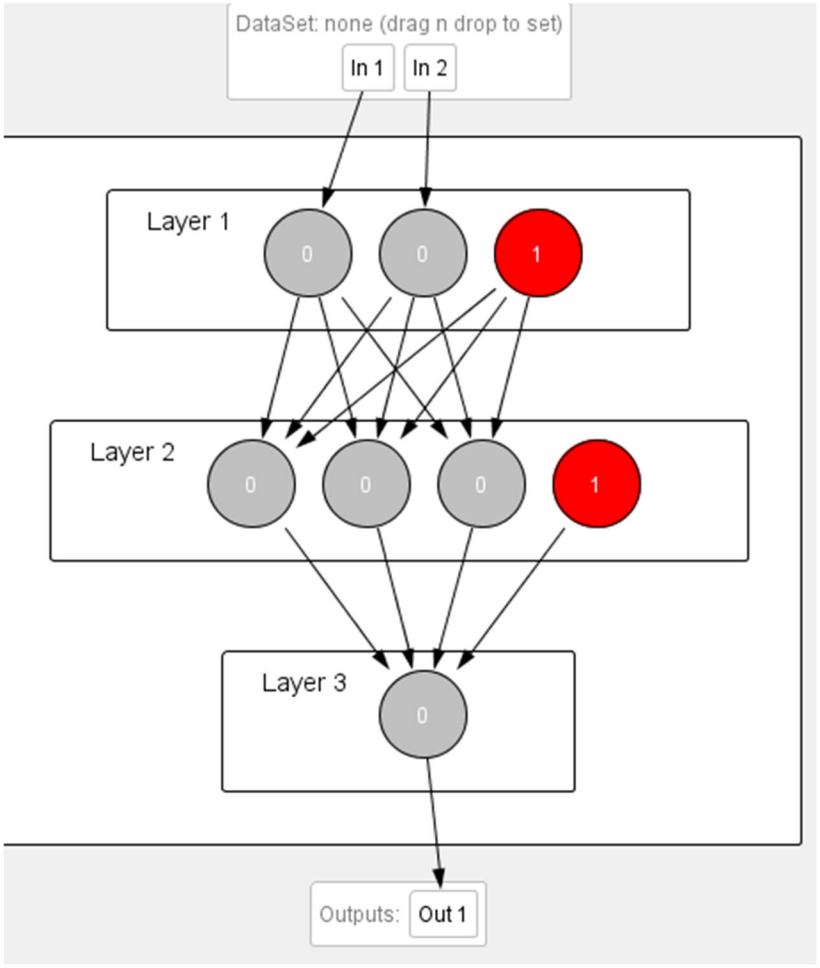

Рис. 10.15. Созданная многослойная нейронная сеть персептронов

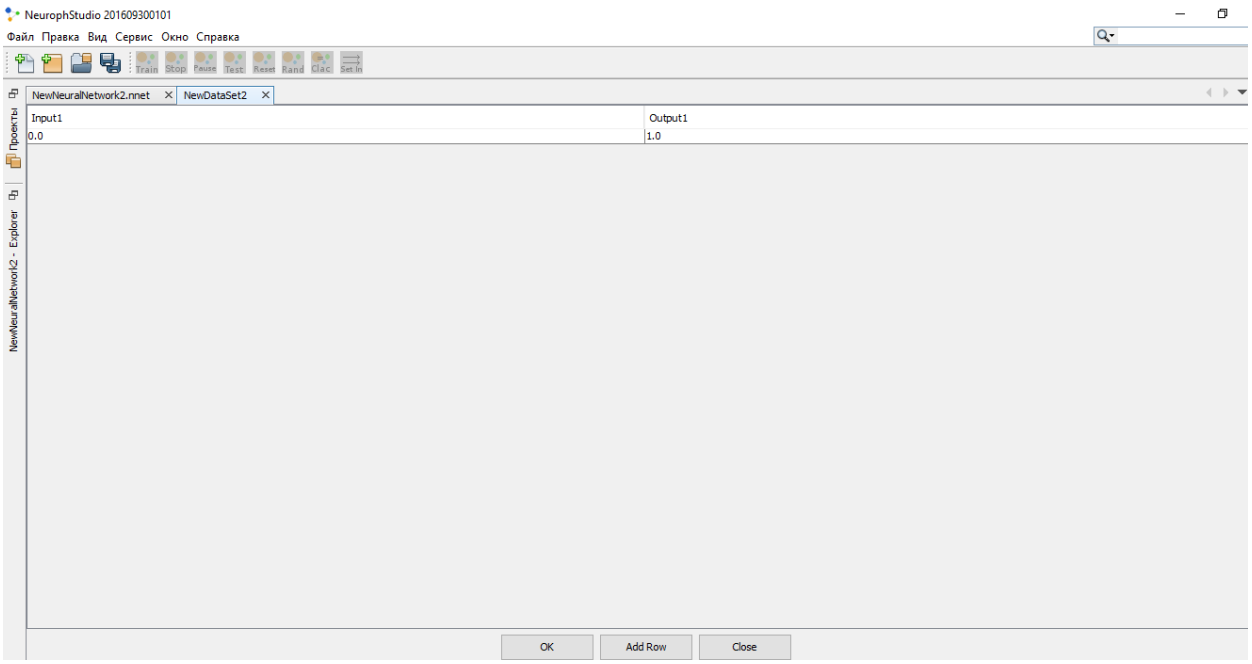

Рис. 10.16. Тренировочный набор данных

9. Тренировка сети. Для начала процедуры тренировки в окне проекта выбираем тренировочный набор данных и нажимаем на *Train*. В диалоге установки параметров обучения *Set Learning parameters* используем все значения по умолчанию и снова нажимаем на *Train*. Когда число ошибок сети *Total Net Error* станет равным нулю, тренировка завершится (рис. 10.17).

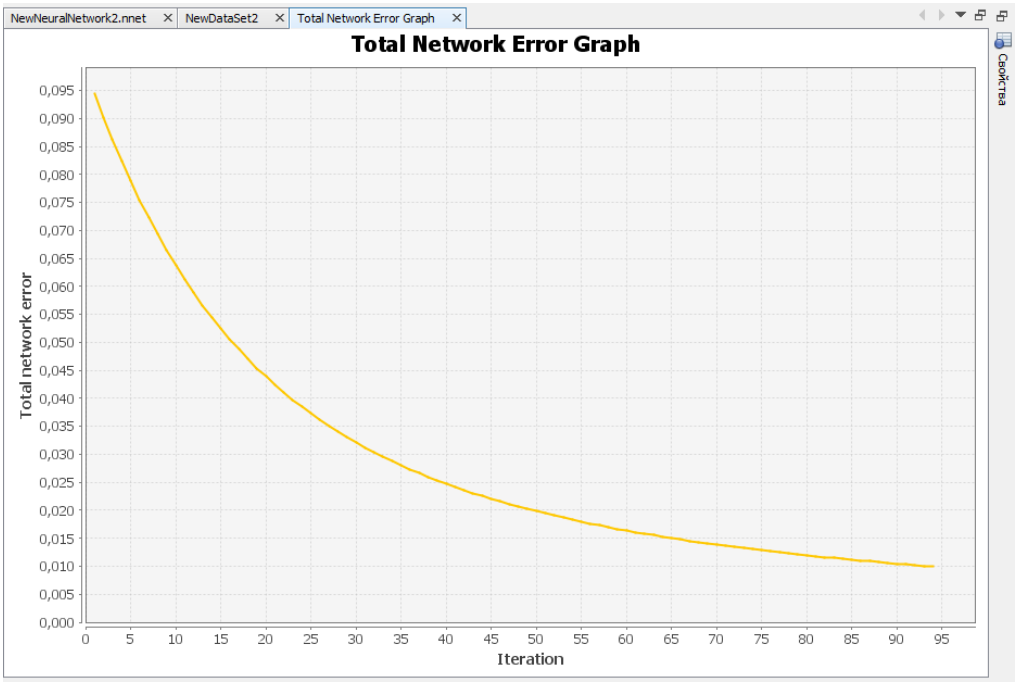

Рис. 10.17. Тренировка сети

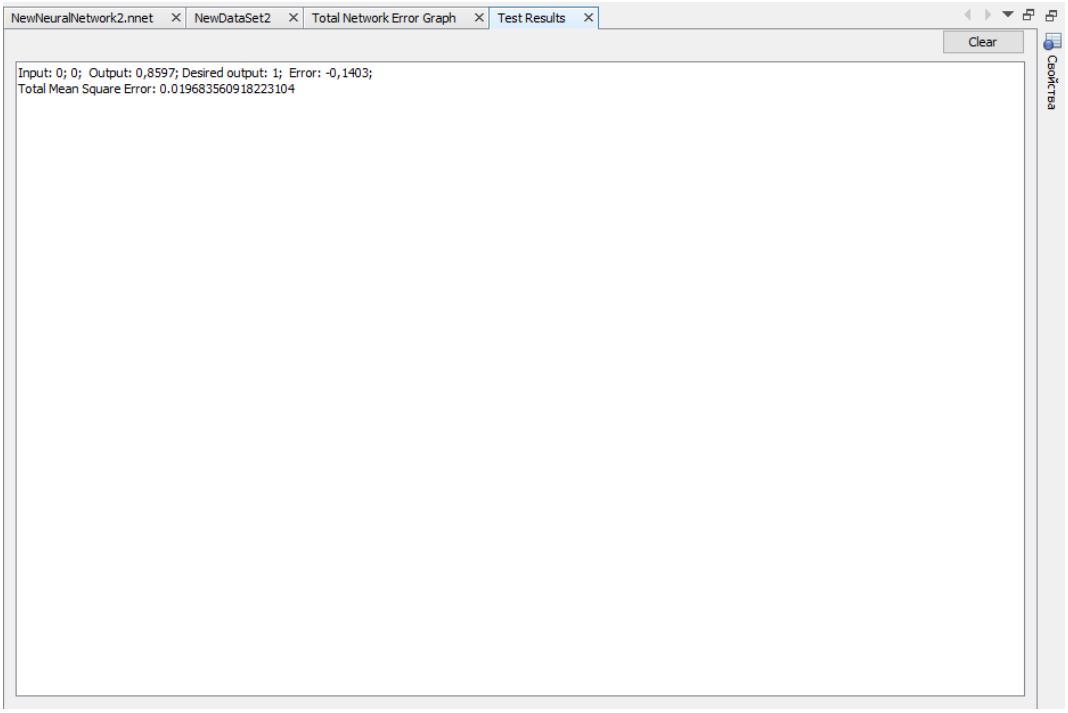

Рис. 10.18. Результаты теста

10. После завершения тренировки можно протестировать нейронную сеть тем же набором тренировочных данных.

Результаты теста будут открываются в новом окне (рис. 10.18).

Контрольные вопросы

1. Какие настройки есть в параметрах при создании многослойной сети персептронов?
2. Как осуществляется тренировка многослойной сети персептронов?
3. Как осуществляется тестирование многослойной сети персептронов?

10.3. Лабораторная работа № 3. Распознавание изображений с помощью нейронных сетей

Время выполнения – 2 часа.

Цель работы: научиться использовать нейронные сети для распознавания изображений.

Задачи работы

1. Создать и обучить нейронную сеть для распознавания изображений с помощью *Neuroph Studio*.
2. Изучить параметры нейронной сети для распознавания изображений.
3. Протестировать сеть персептронов.

Методика выполнения работы

Создание и обучение нейронной сети для распознавания изображений с помощью Neuroph Studio

1. Создайте проект в *Neuroph Studio* с помощью команды *File* → *New Project*. Далее необходимо выбрать *Neuroph Project* и нажать на *Next*.

Введите имя проекта и укажите его расположение, нажмите на *Finish* (рис. 10.19).

Проект создан, теперь можно добавить нейронную сеть.

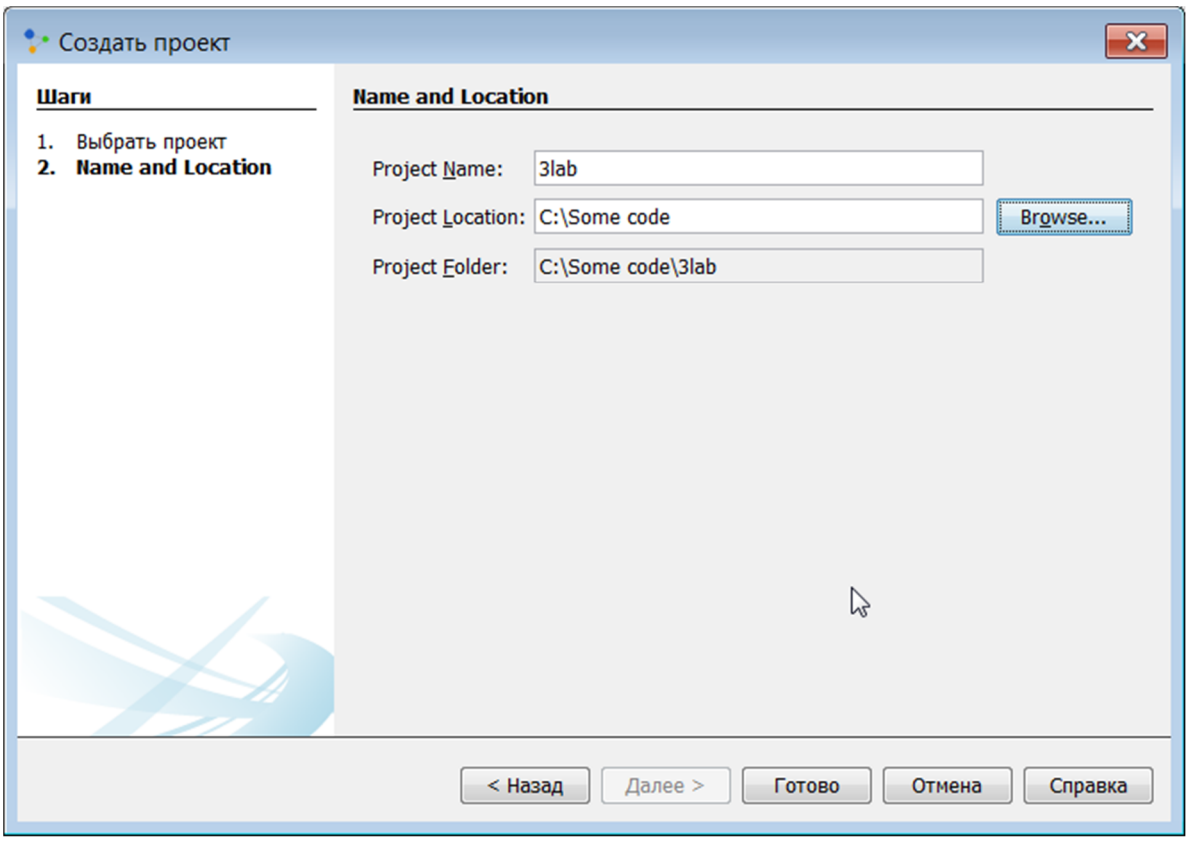

Рис. 10.19. Создание проекта

2. Далее в созданном проекте создаем файл (рис. 10.20), для этого выбираем нужный проект, категорию *Neuroph* и тип файла *Image Recognition*.

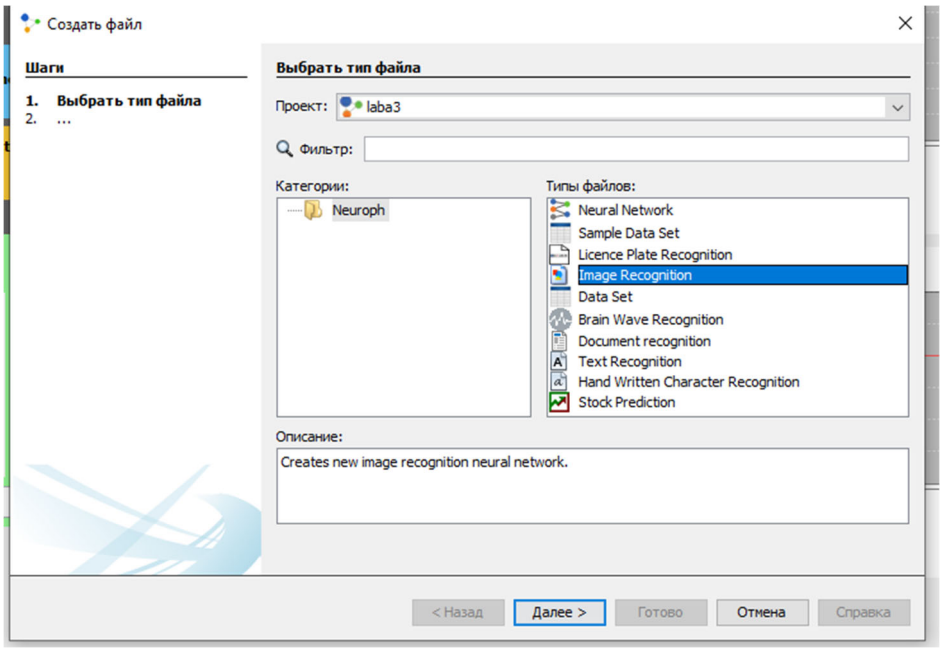

Рис. 10.20. Выбор типа файла

3. Также при создании файла необходимо выбрать изображения, которые нужно распознать, и изображения, которые не должны быть распознаны. Добавление изображения для распознавания показано на рис. 10.21.

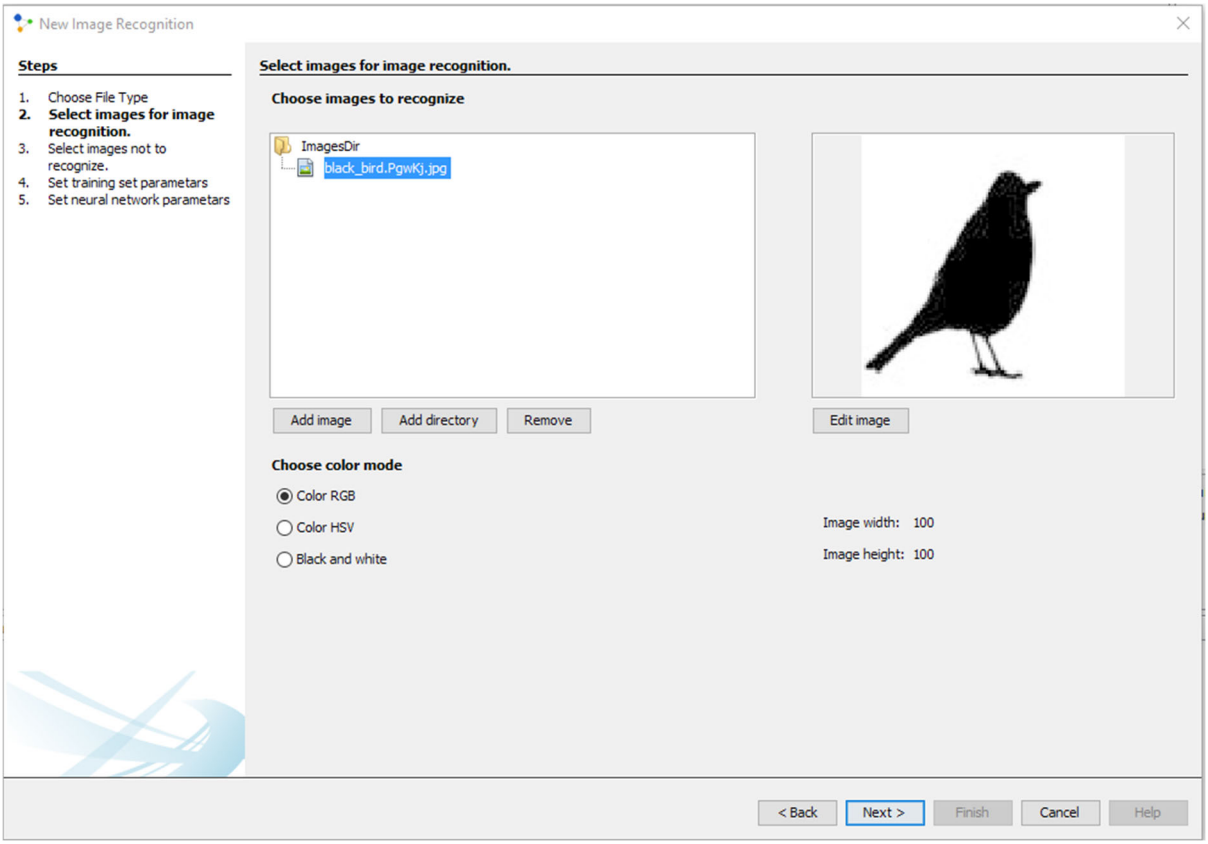

Рис. 10.21. Добавление изображения

4. На следующем этапе необходимо выбрать изображения, которые не должны быть распознаны. Это необходимо для избегания ложных срабатываний. Как правило, это полностью красные, зеленые и голубые изображения, но могут также быть и другие. Добавление изображения, которое не должно быть распознано, показано на рис. 10.22.

5. Затем необходимо ввести название для обучающего набора *Training Set Label* и разрешение изображения *Image Sampling Resolution*, после чего нажать на *Далее*. Если необходимо изменить параметр разрешения изображения, можно воспользоваться разделом *Image Sampling Resolution*. Он позволяет масштабировать предоставляемые изображения. В результате масштабирования изображение будет меньше и обучение будет происходить быстрее. Количество нейронов во входном слое и размер входного вектора будут определяться размерами изображения (рис. 10.23).

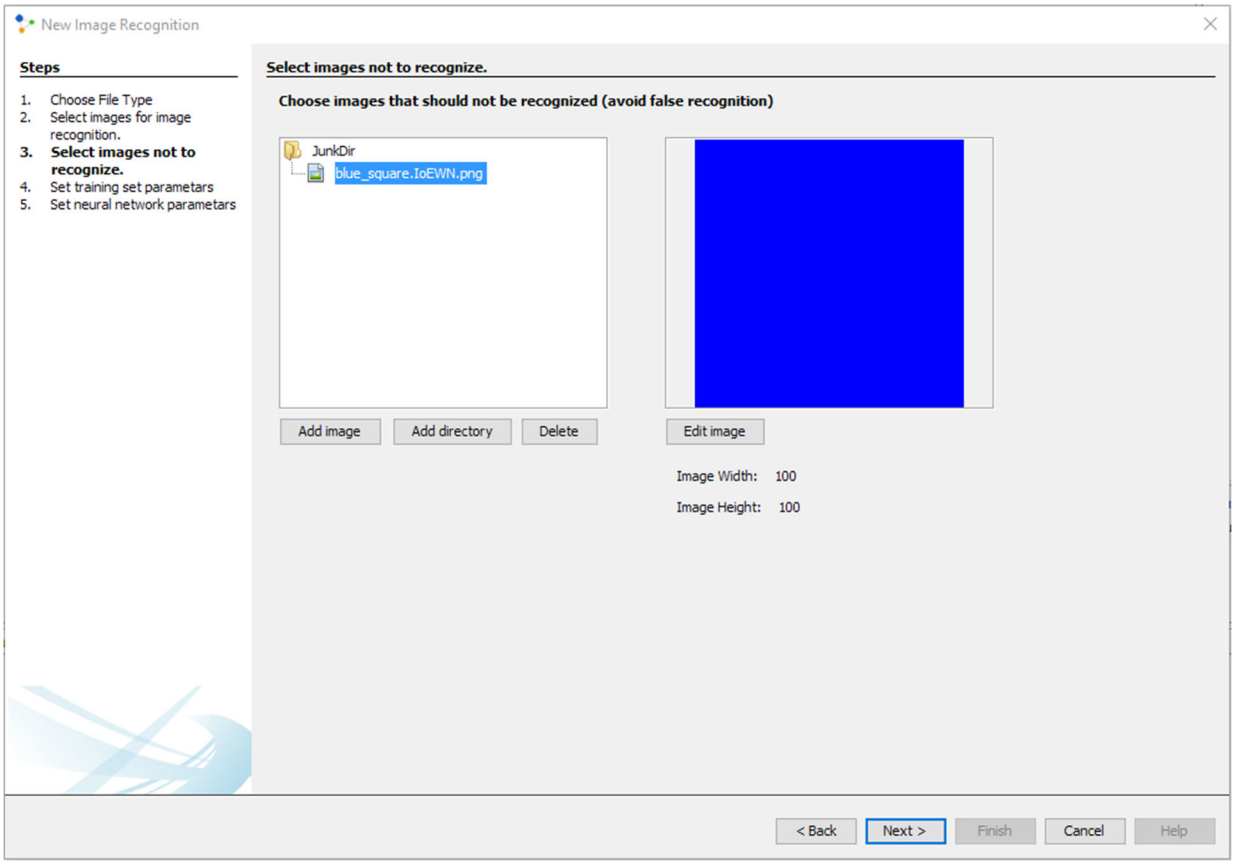

Рис. 10.22. Изображение, которое не должно быть распознано

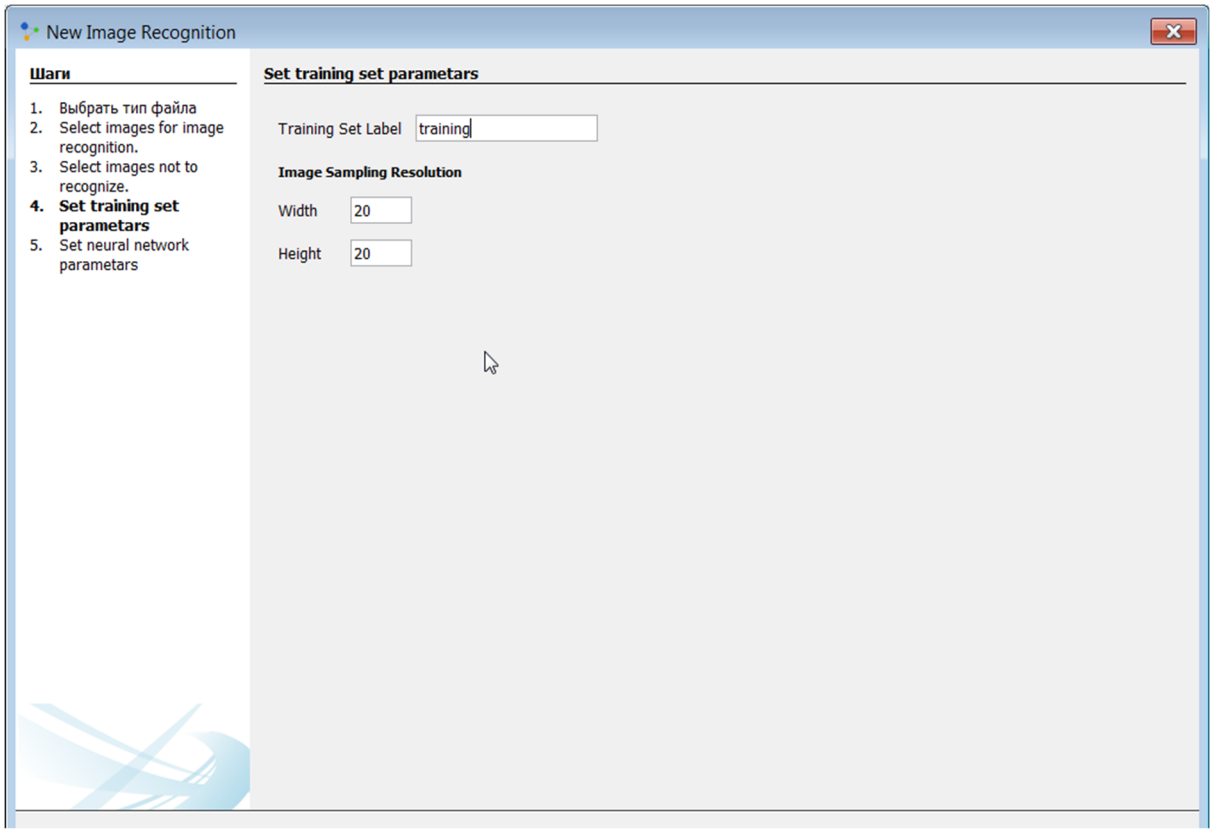

Рис. 10.23. Задание параметров изображения

6. Далее требуется сделать нейронную сеть и указать ее следующие параметры:

– название сети *Network*;

– функция активации. Данный параметр показывает, какие функции будут применяться для вычисления выходного сигнала нейрона. В большинстве случаев по умолчанию можно оставить *Sigmoid*, но иногда *TANH* может дать лучшие результаты;

– число скрытых слоев *Hidden Layers Neuron Counts*. Он предназначен для определения количества скрытых слоев в сети, в том числе количества нейронов для каждого скрытого слоя.

Известно, что между входным слоем и выходным будут располагаться скрытые слои. Для того чтобы успешно изучать учебный набор, необходимо иметь наименьшее возможное количество слоев и нейронов. Экспериментальным путем можно выяснить подходящее количество скрытых нейронов, которое также зависит от количества входных и выходных нейронов.

Для начала можно попробовать изображение 8 × 8 пикселов и один скрытый слой с 12 нейронами, что является значением по умолчанию. Если вы хотите увеличить число нейронов, то необходимо ввести большее число. Если нужно добавить больше чем один слой нейронов, введите количество нейронов в каждом слое через пробел. Например, значение 10 6 4 создаст три скрытых слоя с 10, 6 и 4 нейронами (рис. 10.24).

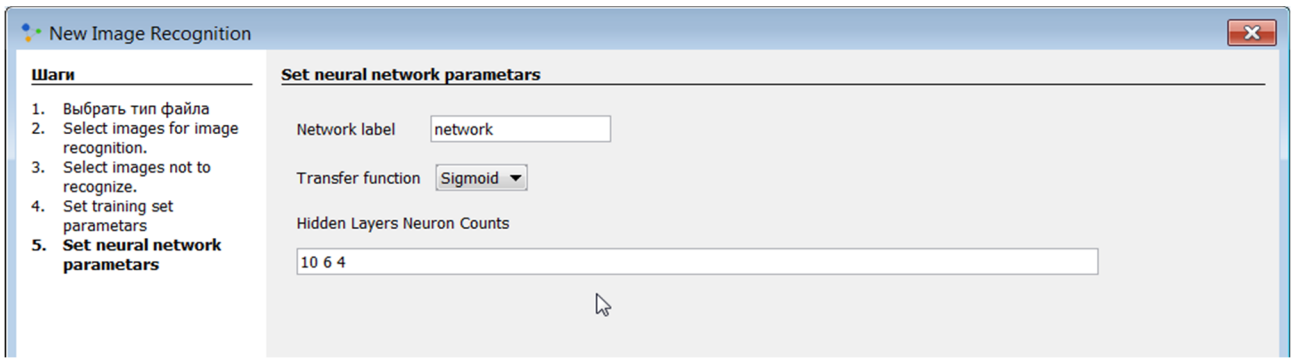

Рис. 10.24. Задание параметров нейронной сети

7. Нажимаем на кнопку *Finish*, чтобы создать нейронную сеть. После нажатия на кнопку открывается новое окно с созданной нейронной сетью (рис. 10.25).

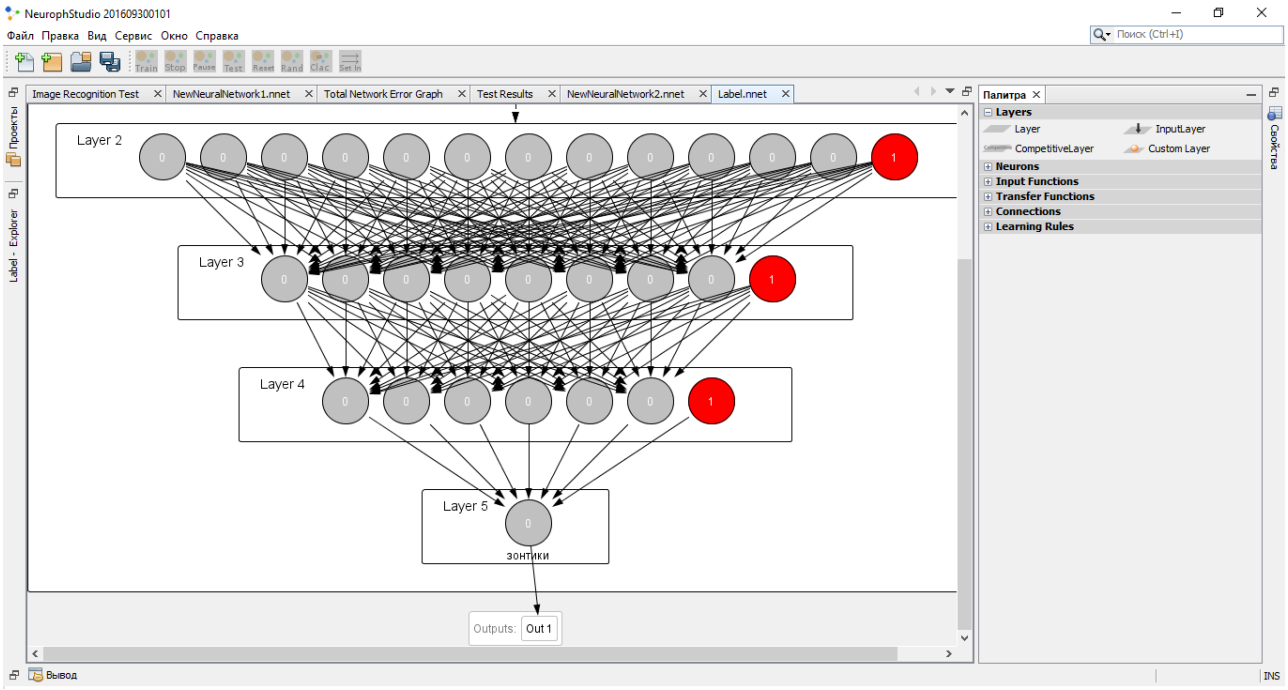

Рис. 10.25. Созданная нейронная сеть распознавания изображения

Обучение сети

1. Для обучения сети необходимо выбрать обучающий набор изображений. Он находится в дереве проекта. Далее нажмите на кнопку *Train* (рис. 10.26).

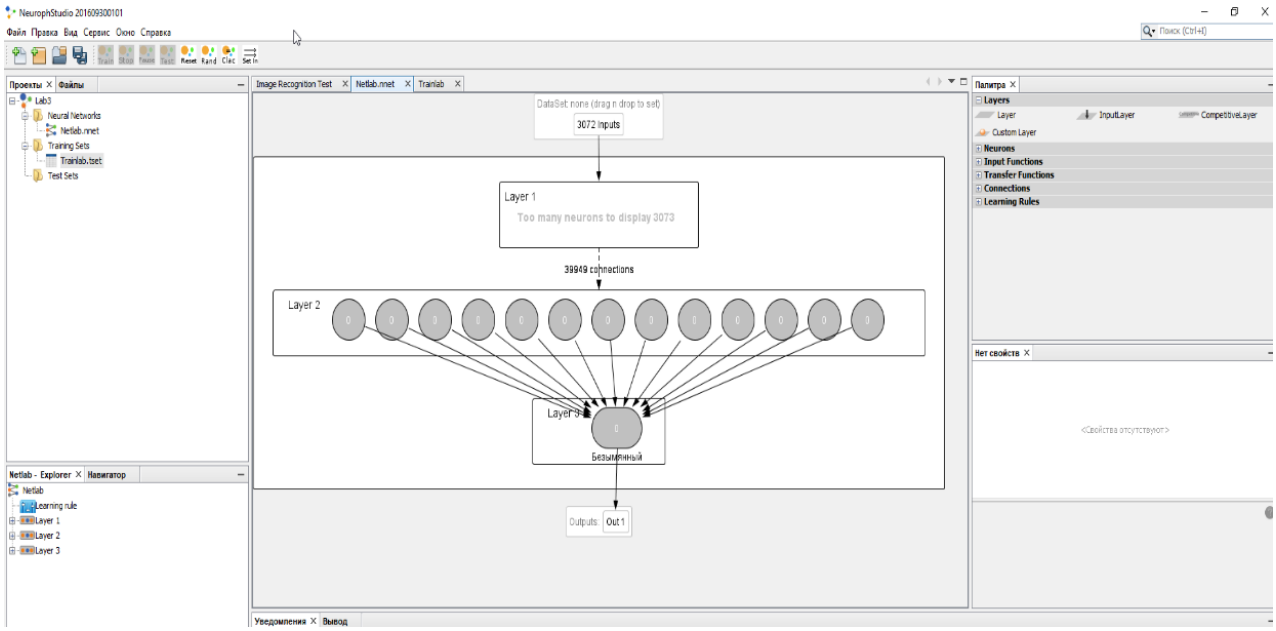

Рис. 10.26. Обучение сети

2. В результате откроется диалоговое окно для настройки параметров обучения. Используйте настройки по умолчанию и нажмите на кнопку *Train*. В итоге начнется тренировка сети и откроется график обучения сети, а также счетчик итераций. Это представлено на рис. 10.27. Можно следить за процессом обучения. Если обучение сети будет застревать, то есть общая ошибка сети не уменьшается, можно попробовать изменить количество нейронов, слоев или параметры для обучения.

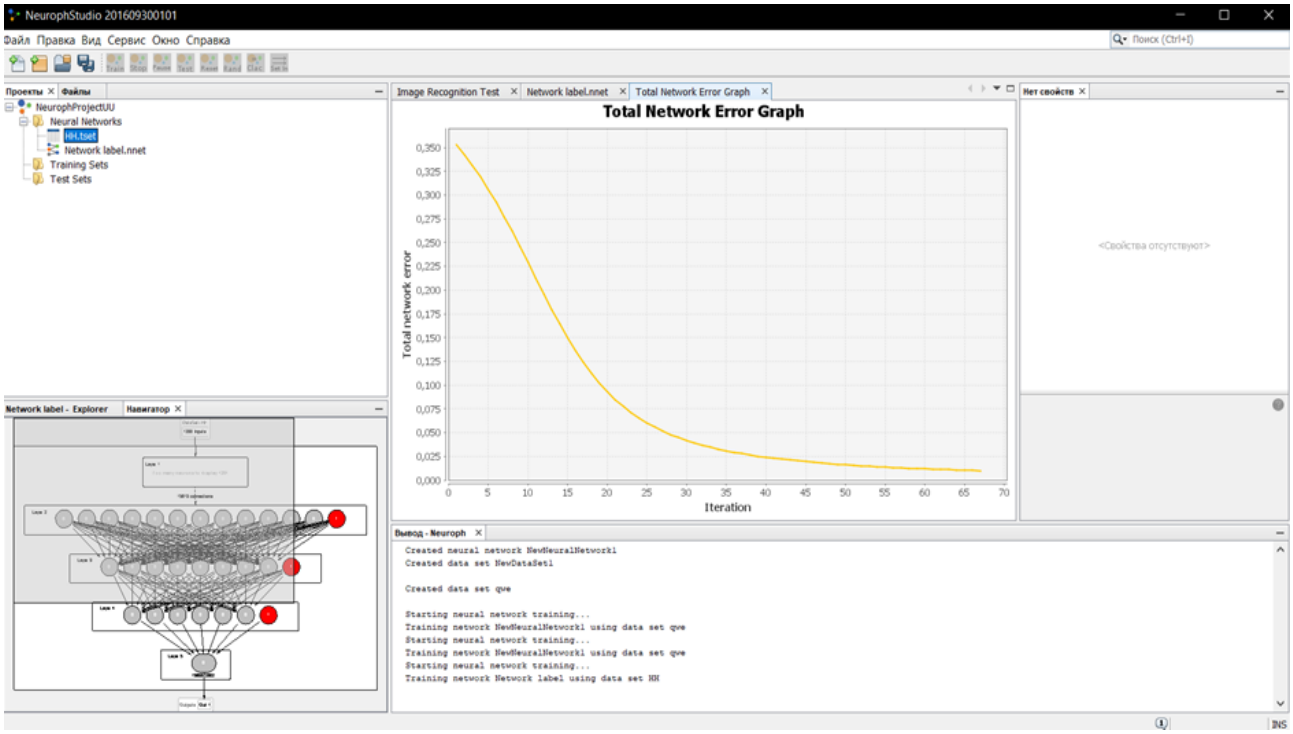

Рис. 10.27. Тренировка сети

Тестирование сети

После окончания тренировки можно испытать нейронную сеть. Для этого нажмите на кнопку *Выбрать тестовое изображение*. Здесь необходимо задать входное изображение для нейронной сети, а на выходах сети будет отображаться список изображений и значения соответствующих выходов нейронов. Узнаваемый образ соответствует нейрону с наибольшим выходным значением. Вы можете проверить весь набор данных, нажав на кнопку *Test whole data set* (рис. 10.28).

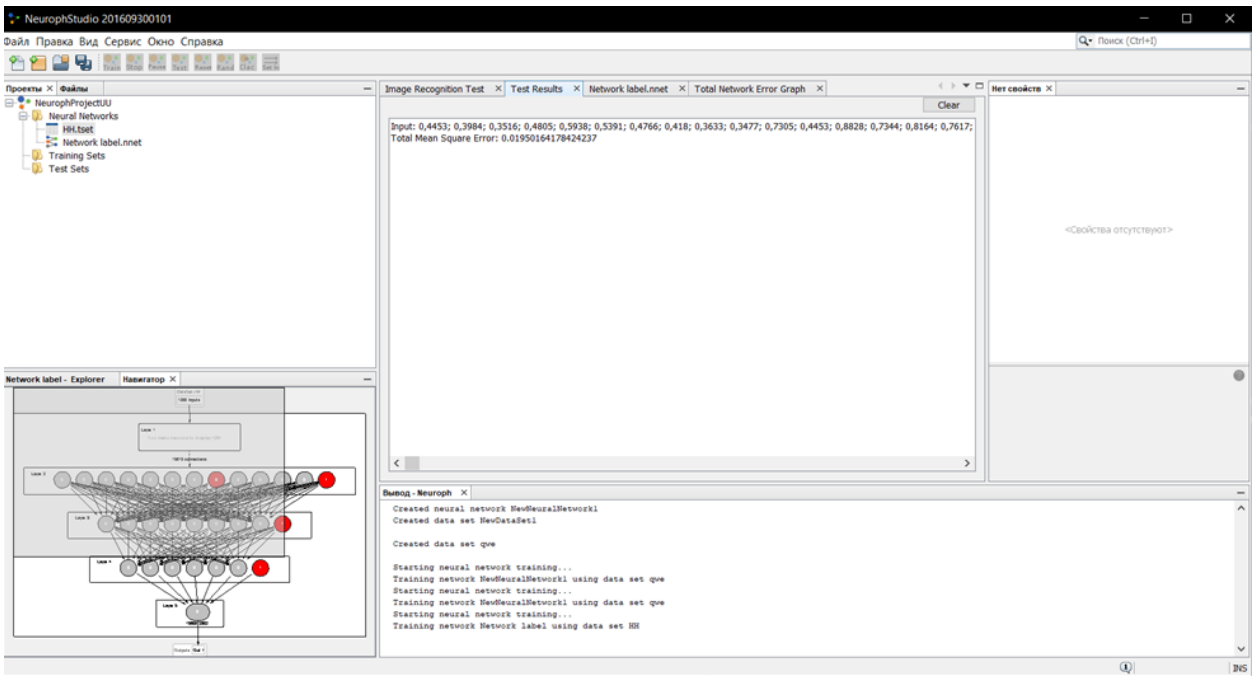

Рис. 10.28. Результаты тестирования

Сохранение нейронной сети

Чтобы сохранить нейронную сети в качестве компонента Java, нажмите *Main menu* → *File* → *Save* и используйте Nnet-расширение. Сеть будет сохранен как сериализованный объект *MultiLayerPerceptron*.

Контрольные вопросы

1. Какой тип файла надо выбрать в проекте для распознавания изображения?
2. Какие параметры можно задать при создании нейронной сети для распознавания изображения?
3. Как осуществляется обучение сети для распознавания изображения?
4. Как осуществляется тестирование сети для распознавания изображения?
5. Как осуществляется сохранение нейронной сети?

10.4. Лабораторная работа № 4. Исследование генетических алгоритмов на примере работы программы DarwinBots

Время выполнения – 4 часа.

Цель работы: изучить функционал программы DarwinBots для исследования процессов работы генетических алгоритмов и осуществить несколько экспериментов в соответствии с вариантом задания, запрограммировать своего робота и сравнить его алгоритм с другими.

Задачи работы

1. Запустить программу DarwinBots и создать новое поле моделирования.

2. Установить начальные параметры в зависимости от варианта.

3. Изучить функционал программы DarwinBots.

4. Снимать результаты мутаций через одинаковые промежутки времени (количество точек сбора данных информации – 50, через произвольные промежутки времени (от 3 до 15 с)).

Общие сведения о программе DarwinBots

Программа DarwinBots предназначена для имитации жизни популяции организмов – ботов (роботов). ДНК ботов можно описывать вручную или на основе специализированного языка программирования. ДНК считывает информацию с множества различных входов, затем модифицирует ее с целью подключения к выводам с действиями. Таким образом, ДНК ботов является большим конечным автоматом.

Код бота (ДНК) можно описать в текстовом файле с расширением формата *.txt. Это позволяет работать с ботом в любом редакторе, в том числе в Блокноте. Суть симуляции заключается в выполнении ДНК-кода за каждый цикл, при этом учитываются все возможные взаимодействия, такие как организмы, мутации, вирусы.

Генетический код любого бота будет состоять из шести основных способностей:

1) к движению вперед;

2) к движению назад;
3) двигаться налево;
4) двигаться направо;
5) двигаться кардинально налево;
6) двигаться кардинально направо.

Основным элементом программы DarwinBots является интерфейс, или мир ботов. Он является полем для моделирования. Мир ботов – это большая недискретная плоскость, общий вид интерфейса которой представлен на рис. 10.29. По умолчанию боты имеют возможность двигаться по всей площади мира. Для настройки мира необходимо зайти в параметры симуляции программы DarwinBots [5].

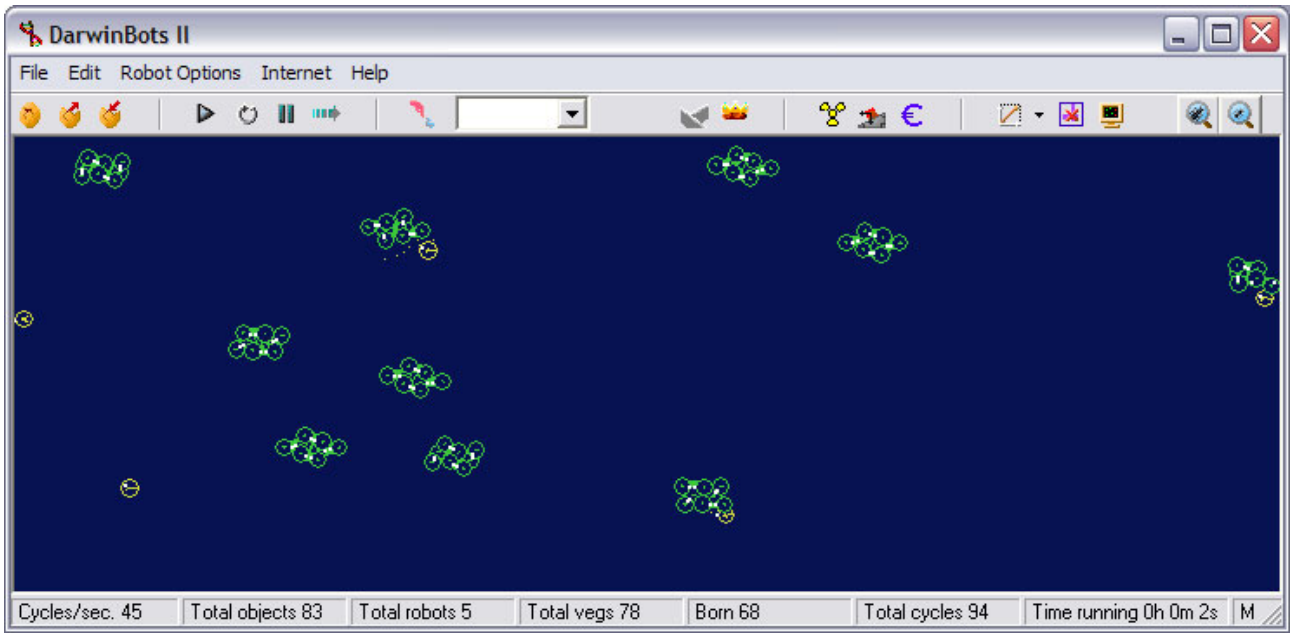

Рис. 10.29. Вид интерфейса, имитирующего искусственную жизнь ботов

Каждый бот показан (рис. 10.30) в виде окружности. Она имеет определенный размер и цвет, с некоторым узором в центре. «Глаз» бота обозначается в виде белой точки, и в него включены 9 простых глазков.

Сетка в виде полей зрения каждого глазка будет появляться при выделении бота на поле для моделирования. Глазки передают в программу значение, примерно соответствующее размеру объекта, попавшего в поле зрения.

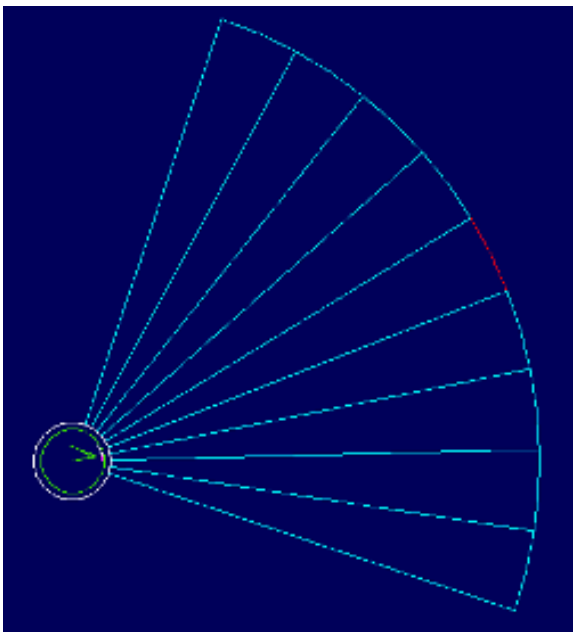

Рис. 10.30. Бот в программе DarwinBots

Система простейших тактильных ощущений позволяет боту чувствовать столкновения или атаки со стороны других ботов. Сенсорные входы ботов анализируются в ДНК, которая написана на языке с использованием абстракции FILO («первым пришел – последним ушел», по принципу поставленных друг на друга тарелок). ДНК большинства ботов имеют от 6 до 20 процедур, именуемых генами, в которых может выполняться до 200 операций. Каждая операция может мутировать [6].

У каждого бота может быть свой источник энергии. Большинство действий уменьшают количество энергии бота. Когда энергия бота равна нулю, бот умрет, а когда она станет выше уровня, установленного в ДНК, он начнет воспроизводиться. Пополнение энергии возможно за счет поглощения других ботов (аналогично охоте в обычных условиях) и энергетических запасов «тела». Если бот выбран как автотрофный, то энергия пополняется автоматически на длительное время.

Бот имеет возможность стрелять нематериальными снарядами, атаковать, удалять мусор, заражаться инфекциями и обмениваться информацией с другими ботами. Снаряд – это точка на плоскости модели, не имеющая массовых и физических измерений.

Боты также могут общаться с другими ботами, образуя сложные многоклеточные структуры, в которых боты могут обмениваться энергией, работать и перемещаться по полю.

Когда бот воспроизводится, его ДНК передается потомству, иногда с некоторыми изменениями, которые влияют на поведение бота. Как и в реальном мире, с изменениями ДНК, эволюция может произойти – следующее поколение ботов может быть более способным атаковать, размножаться, избегать контакта и т. д. Таким образом, подобные мутации передаются из поколения в поколение или исчезают. Со временем в ДНК может накапливаться нежелательный код, который становится бесполезным. Это потребует больше энергии и сделает ДНК более загруженной.

Поскольку программа не решает, какой организм будет расти, способности бота в конечном итоге реализуются за счет комбинации стратегий движения, управления энергией, воспроизводства и так далее. Никаких ограничений на сложность кода ДНК не накладывается [8].

Эволюция DarwinBots заняла много времени. Поколения ботов могут прожить несколько тысяч циклов, и большинство симуляций выполняется около 15 циклов в секунду, поэтому значительный естественный селектор может занять от нескольких часов до нескольких дней.

Основные пункты меню программы DarwinBots представлены на рис. 10.31 и 10.32.

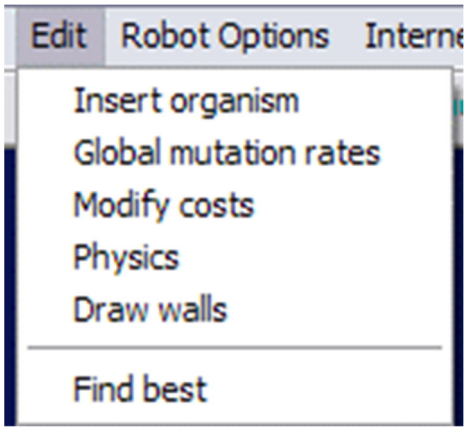

Рис. 10.31. Меню *Edit*

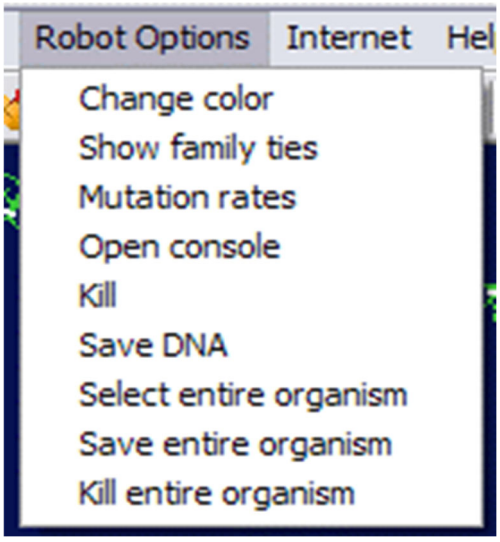

Рис. 10.32. Меню *Robot Options*

Назначение основных команд в меню программы DarwinBots показано в табл. 10.1 и 10.2 [9].

Таблица 10.1

Основные команды меню *Edit*

Команда	Назначение команды
Insert organism	Файл откроется с расширением формата *.dbo (ранее зарегистрированный организм), при этом будет сгенерирована запись
Global mutation rates	Откроется панель для изменения значений мутаций в организмах
Modify costs	Будет открыто диалоговое окно для определения затрат различных вариантов моделирования в программе DarwinBots
Physics	Будет открыто диалоговое окно для физических показателей ботов с целью изменения параметров моделирования
Draw walls	Откроется окно для рисования стен. С помощью этой команды вы можете нарисовать стену, нажав кнопку рисования стены. Для этого нужно отметить несколько точек в области моделирования и обозначить стены. Для завершения необходимо нажать на кнопку *Сделать стену*
Find best	Будет найден наилучший единственный робот, у которого имеется больше всего живых потомков

Таблица 10.2

Основные команды меню *Robot Options*

Команда	Назначение команды
Change color	Можно изменить цвет выбранного робота
Show family ties	Откроется диалоговое окно с инструментами, позволяющими подсчитать количество живых потомков выбранного робота. Кроме этого, имеется возможность графически отобразить степень родства робота с потомками
Mutation rates	Будет открыто диалоговое окно, где можно изменить степень мутации для выбранного робота
Open console	Будет открыто диалоговое окно для ввода команд роботу с помощью консоли
Kill	Данная команда убивает выбранного робота
Save DNA	Можно сохранить ДНК для текущего робота
Select entire organism	Будет расширен выбор для всего организма, т. е. на все ячейки, которые связаны с выбранным организмом
Save entire organism	Будет сохранен выбранный робот или организм в файле формата *.dbo. При этом будет заморожено текущее состояние
Kill entire organism	Убивает выбранный организм

Диалоговые окна для различных вариантов моделирования (*Simulation parametrs and options*) показаны на рис. 10.33–10.37. Вкладка *Species* (рис. 10.33) содержит информацию о видах роботов, куда входят названия видов, описания, настройки внешнего вида, настройки количества особей, начальной энергии, настройки мутационных коэффициентов, а также дополнительные настройки видов. На данной вкладке можно добавлять новые виды, дублировать, переименовывать или удалять имеющиеся виды.

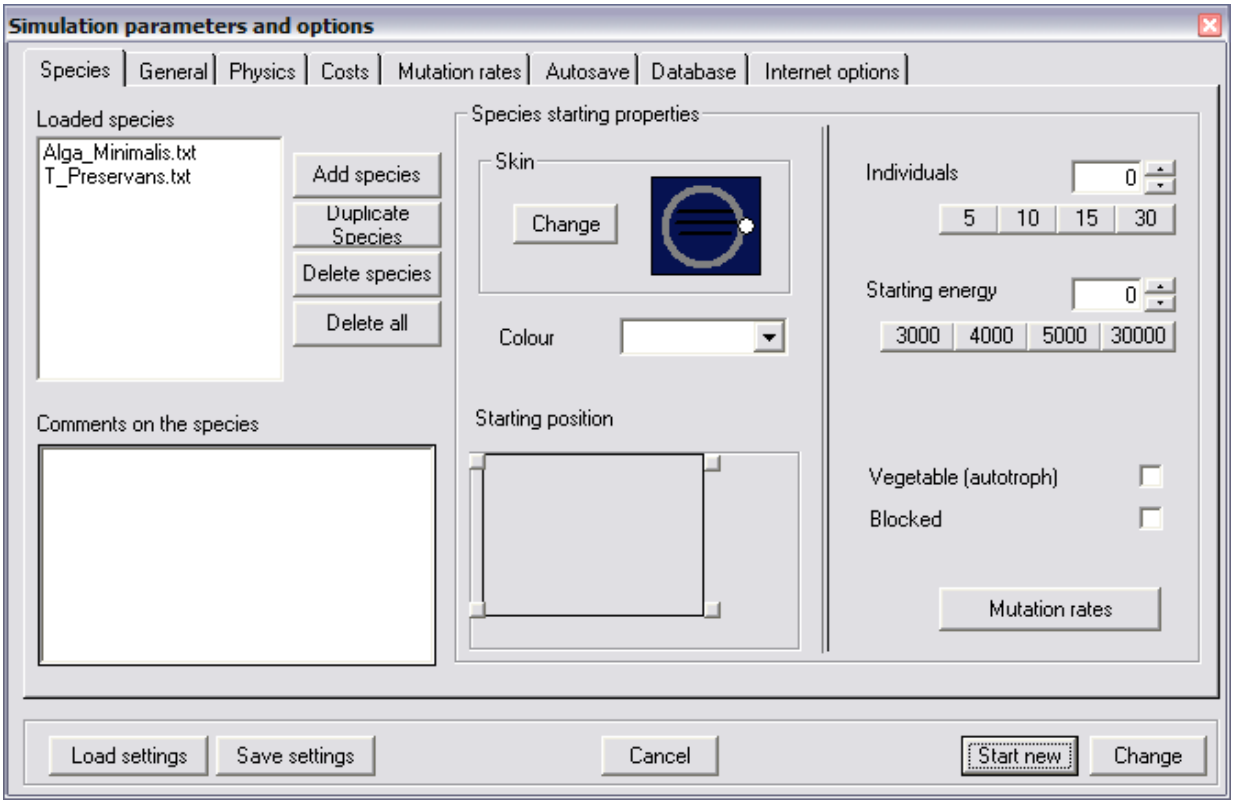

Рис. 10.33. Вкладка *Species*

На вкладке *General* (рис. 10.34) устанавливаются общие свойства области (например, размер поля), ограничение рождаемости (ограничение распространения растительности), настройки энергии света и т. д.

Вкладки *Physics* и *Costs* (рис. 10.35, 10.36) определяют настройки физической среды, в которой перемещаются роботы, настройки движения роботов и настройки энергозатрат на действия роботов.

На вкладке *Mutation rates* (рис. 10.37) можно установить глобальный множитель для коэффициента мутации. С помощью движения ползунка вправо или влево можно повысить или уменьшить коэффициент мутации.

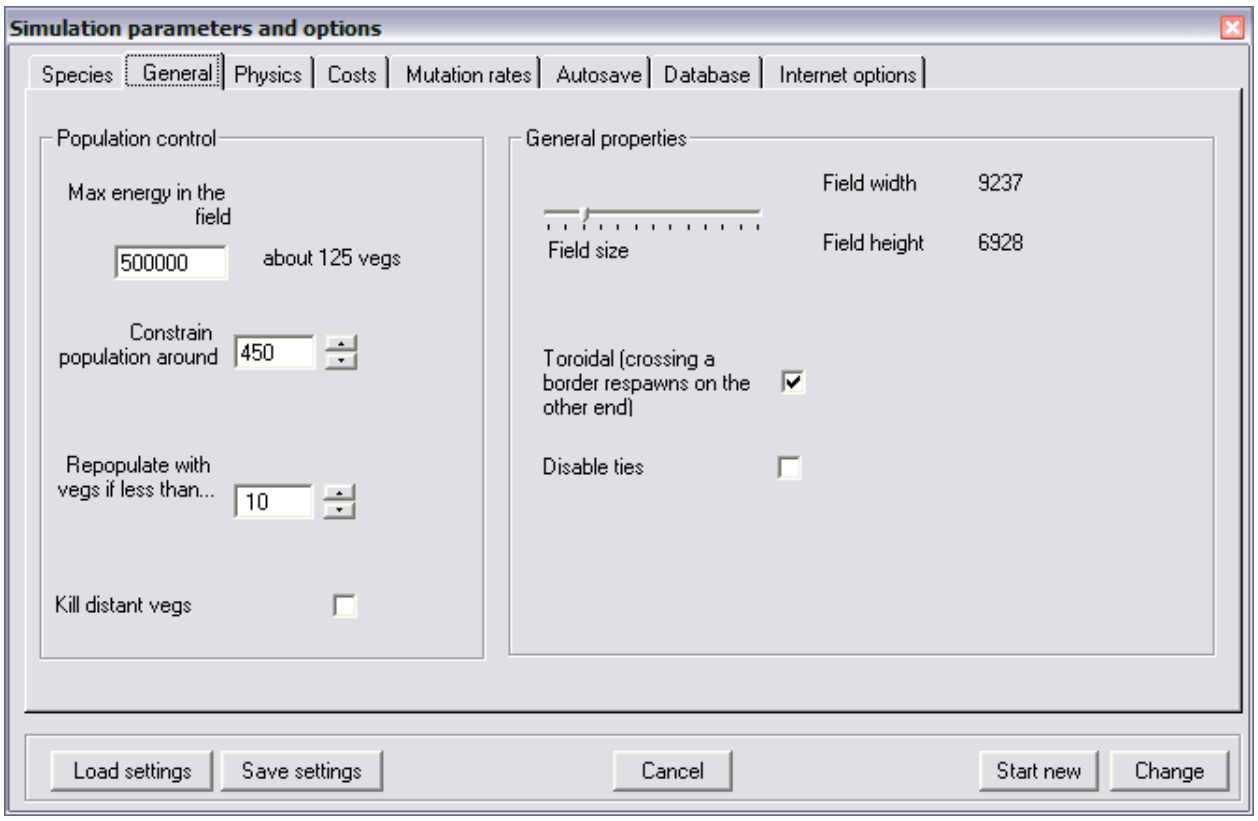

Рис. 10.34. Вкладка *General*

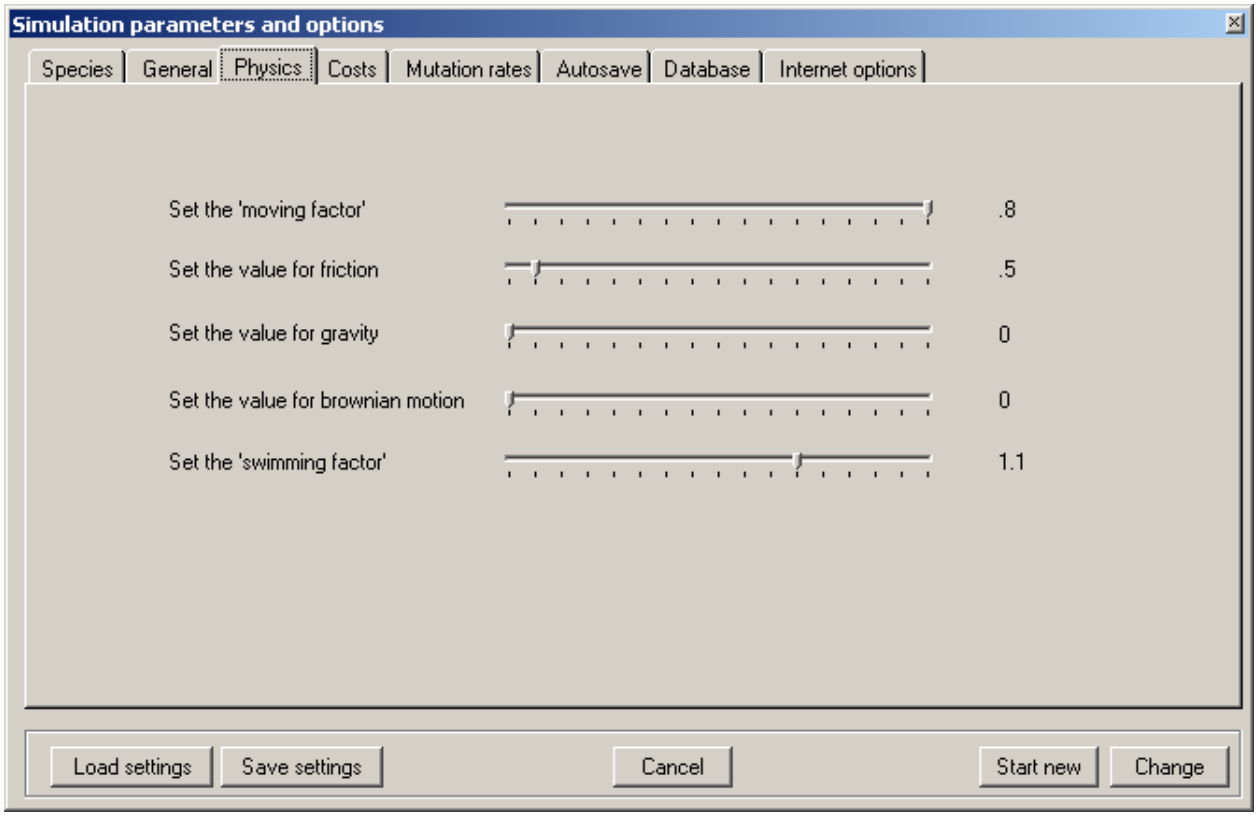

Рис. 10.35. Вкладка *Physics*

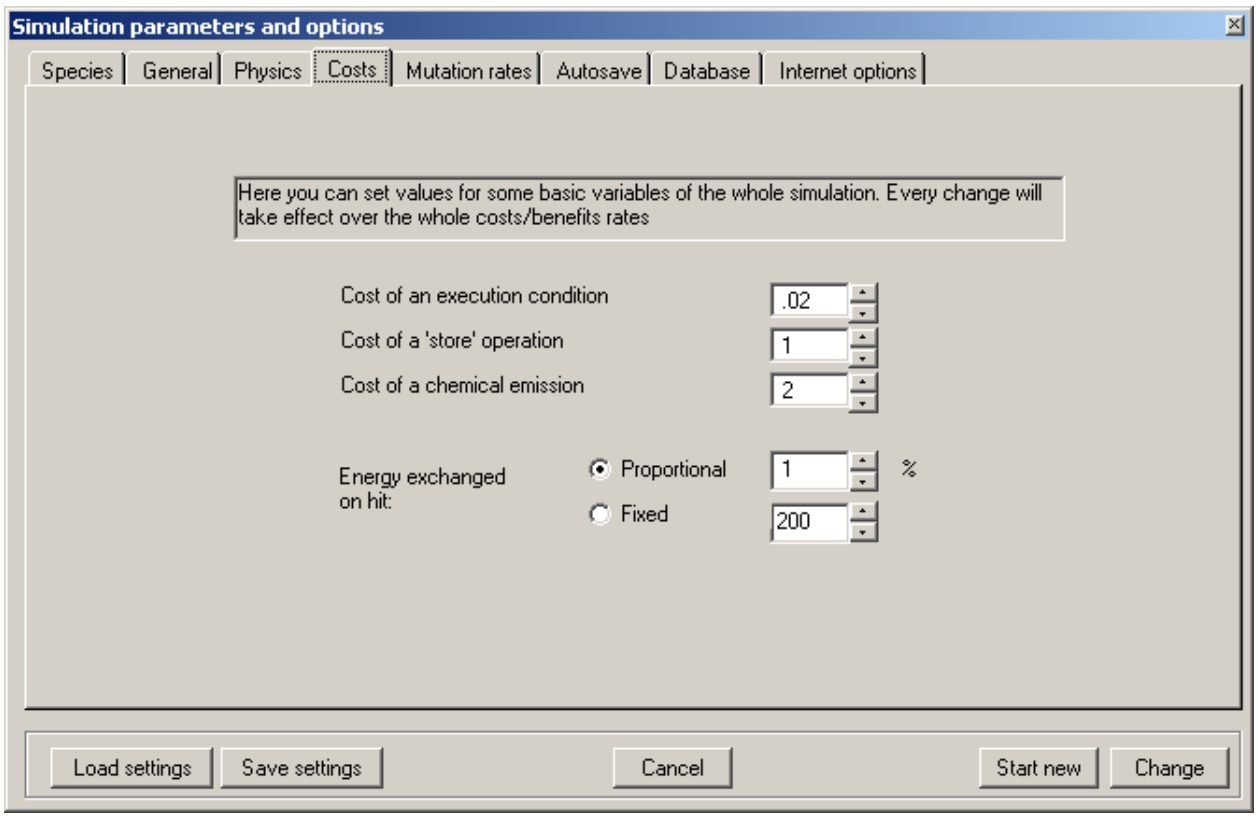

Рис. 10.36. Вкладка *Costs*

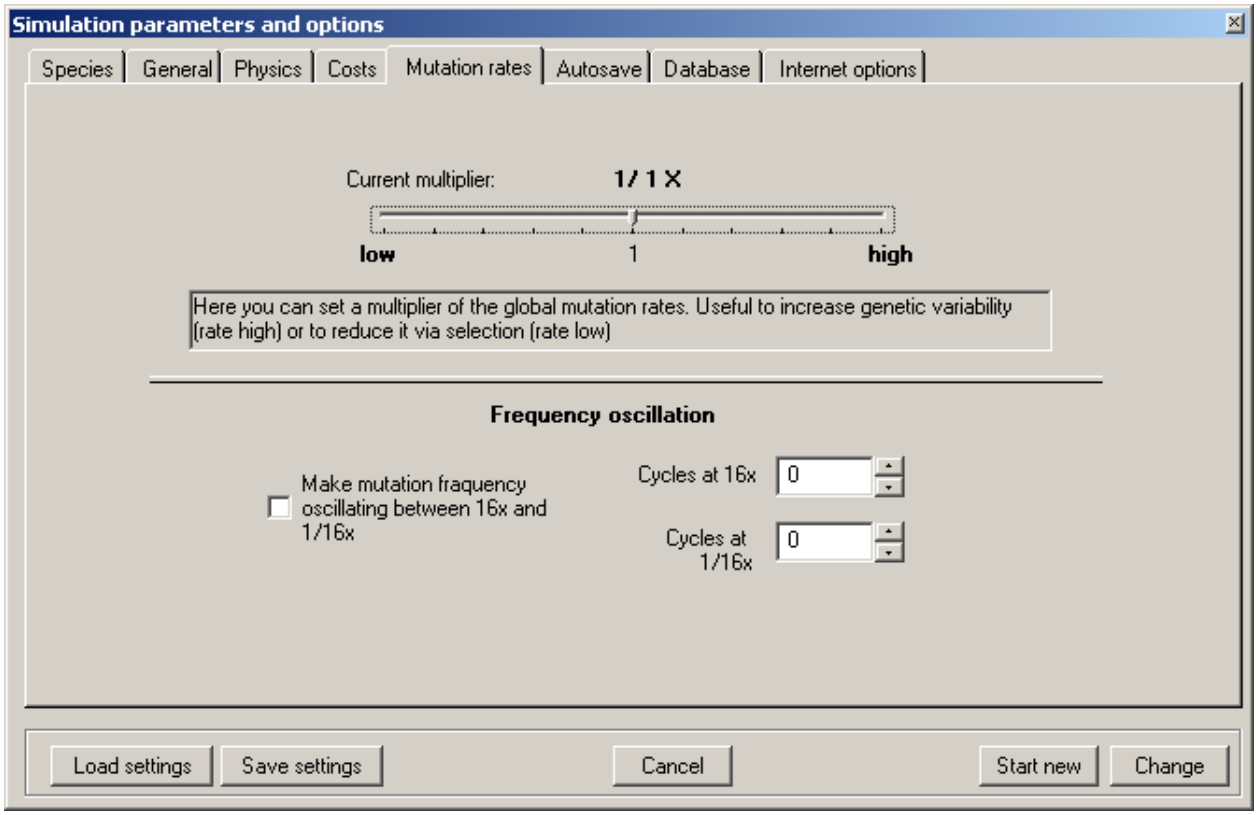

Рис. 10.37. Вкладка *Mutation rates*

Кроме этого, есть возможность для установки колеблющейся нормы мутации. Для этого требуется выбрать кнопку-флажок и далее определить количество циклов основной частоты, которое должно быть выполнено для множителей 16*x* и 1/16*x* [16].

Основные возможности команд для определения возможных симуляций показаны в табл. 10.3–10.7.

Таблица 10.3

Параметры симуляции на вкладке *Species*

Параметр симуляции	Назначение параметра
Loaded species	Показывает список с именами ДНК
Add species	Позволяет добавить новые разновидности в список. Такой файл ДНК обычно имеет формат текстового с расширением *.txt
Duplicate species	У выбранной разновидности можно назначить те же параметры ДНК, что и у исходной
Delete species	Позволяет удалить ДНК из процесса моделирования
Delete all	Позволяет удалить все ДНК из процесса моделирования
Comments	Данное диалоговое окно предназначено для создания комментариев в заголовок ДНК-файла
Skin	Данный параметр может изменить рисунки на поверхности особей. Такие рисунки не будут иметь никакого значения. Рисунок будет изменяться, если робот меняет свою ДНК. С помощью данного параметра можно предоставить информацию о генетическом расстоянии между двумя роботами
Change	Данный параметр позволяет случайным образом изменять начальный рисунок для робота
Colour	Показывает начальный цвет робота
Starting position	Данный параметр позволяет определить начальные позиции для рода или разновидности
Individuals	Данный параметр позволяет определить количество индивидуумов выбранной разновидности. Они должны быть добавлены в начале симуляции
Energy	Количество энергии, которое должно быть предоставлено на старте
Vegetable (autotroph)	Предназначен для определения, должен ли робот быть снабжен отметкой *for free* или нет. Все роботы будут накапливать энергию. Это должно служить для воспроизведения новой энергии в симуляции. Автотрофы – это организмы, рост и размножение которых не зависит от внешних источников органических соединений. Они работают подобно растениям для формирования основы пищевой цепочки

Окончание табл. 10.3

Параметр симуляции	Назначение параметра
Blocked	Параметр предназначен для создания роботов заблокированных разновидностей. Их нельзя двигать или быть ими подвинутыми. Это похоже на растения или бактерии. Они не могут быть изменены мутацией, как и автотрофы
Mutation rates	Предназначен для определения начальных мутационных коэффициентов для данной разновидности. Существует много различных мутационных коэффициентов, как и много типов мутаций ДНК. Первый коэффициент мутации – это коэффициент, согласно которому будет нормироваться мутация. Коэффициент мета-мутации заключается в следующем: зачем проводить время чтобы искать оптимальные коэффициенты мутации, когда они также эволюционируют? Коэффициент мутации увеличивается в $1/x$, где x – это число, которое можно установить в диапазоне от 1 до 32 000. Это число увеличивает частоту, на которой отдельный элемент ДНК будет мутировать (в частности, если имеется 1 000 инструкций в ДНК и коэффициент мутаций инструкции равен 500, то вы будете получать в среднем две специфические мутации на каждое деление)

Таблица 10.4

Параметры симуляции на вкладке *General*

Параметр симуляции	Назначение параметра
Max energy in the field	Это общая сумма энергии, которая может быть в области. Когда этот уровень достигнут, программа прекращает рост растений до падения энергии ниже предела
Constrain population around	Предназначен для того же, что и *Max energy in the field*, но здесь для ограничения используется население. Программа может работать с 1 000 особями
Repopulate with vegs if less than	Предназначен для установки минимального количества растений для моделирования. Если параметр количества растений ниже установленного предела, то программа создаст новое растение
Kill distant vegs	Будут убиты растения, которые находятся далеко от любого робота. Это может ускорить моделирование, но может и воздействовать на развитие новых поведений

Окончание табл. 10.4

Параметр симуляции	Назначение параметра
Field size	Предназначен для установки размера поля. Справа устанавливаются полные вертикальные и горизонтальные единицы. Размер одного поля – 120 единиц
Toroidal	Проверка этого переключателя позволяет определить топологию области. Роботам нельзя пересекать границы области и вновь появляться на противоположной стороне. Робот не может взаимодействовать с теми же роботами, которые находятся на противоположной стороне (например, они не могут видеть их)
Disable ties	Предназначен для отключения создания постоянных связей

Таблица 10.5

Параметры симуляции на вкладке *Physics*

Параметр симуляции	Назначение параметра
Moving factor	Фактор ускорения позволит изменить силу движения робота. Установка этого фактора на 0 запретит роботу прямое перемещение. Все другие виды движения (*hit*, *gravity*, *swim* и т. д.) не будут затронуты
Friction	Показывает уровень силы трения. Высокая сила трения приведет к медленным и тяжелым движениям. Если установить нулевой параметр для силы трения, то роботы станут прыгать вокруг газовых частиц
Gravity	Установка этого значения, кроме нулевого, заставит роботов падать
Brownian motion	Броуновское движение предназначено для предания роботам случайного колебания. Иногда роботы развиваются к состоянию «не делать ничего, пока что-нибудь не случится». Броуновское движение заставит их быть активными
Swimming factor	Силы (жидкой) среды как постоянные силы воздействия. Это необходимо для водоплавающих червей, иначе они будут корчиться, как рыбы, когда их вытаскивают из воды

Таблица 10.6

Параметры симуляции на вкладке *Costs*

Параметр симуляции	Назначение параметра
Cost of an execution condition	Это энергия, которая потребляется любым единичным действием в ДНК. Данный параметр может быть либо полезным, либо нет. Так, для каждого выполняющегося цикла это число, умноженное на общее количество действий в генах, будет вычтено из общей энергии
Cost of a 'store' operation	Стоимость для любой выполненной операции *store*
Cost of a chemical emission	Стоимость для любой эмиссии частицы
Energy exchanged on hit	1) пропорциональное: это количество энергии, которую несет белая частица (выпущенная роботом), вычисленная как процент от полной энергии, выпускаемой роботом; 2) постоянное значение, т. е. частица будет нести указанное количество энергии

Таблица 10.7

Параметры симуляции на вкладке *Mutation rates*

Параметр симуляции	Назначение параметра
Current Multiplier	Устанавливает коэффициент мутации от 1/32*x* до 32*x*
Oscillation Frequency	Показывает частоту случайного изменения для коэффициента мутации (фиксировано 1/16*x* и 16*x* в рамках выбранного коэффициента). Случайное изменение применяется через выбранное количество циклов

Методика выполнения работы

Запуск ботов с различными генами

1. Откройте программу *DarwinBots* и создайте новое поле моделирования. Это представлено на рис. 10.38.

2. Установите начальные параметры симуляции (рис. 10.39). Добавьте коды ДНК (ботов). Данная программа позволяет добавить один из 4 возможных генов: «Поиск пищи», «Поедание пищи», «Избегание объединения» и «Размножение».

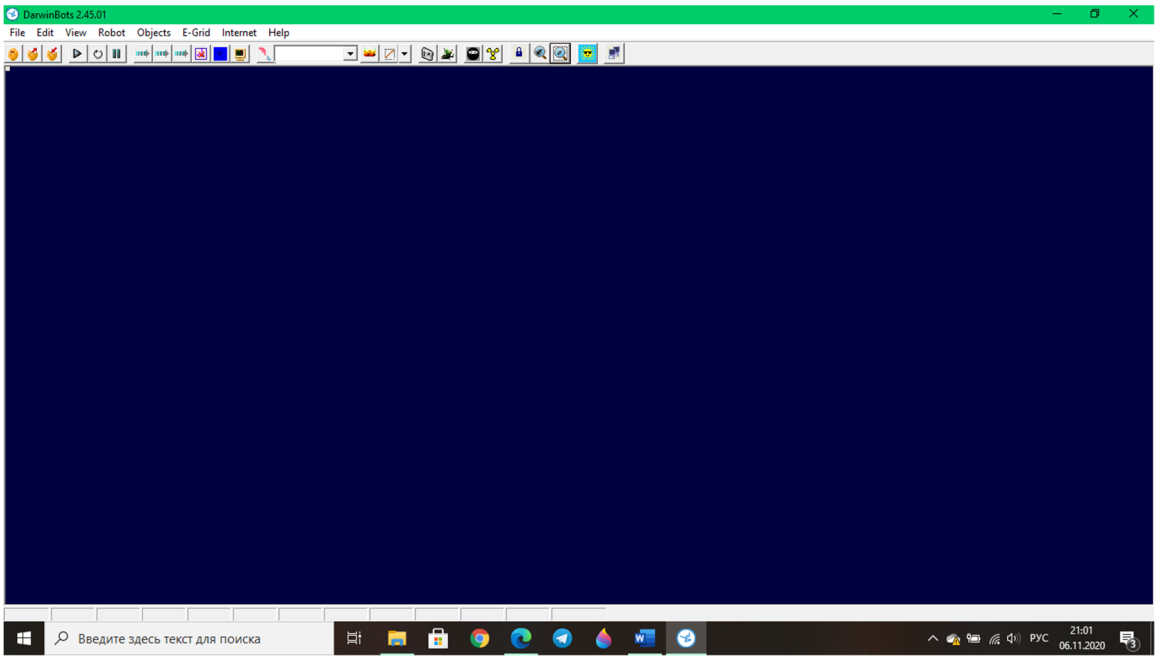

Рис. 10.38. Поле моделирования

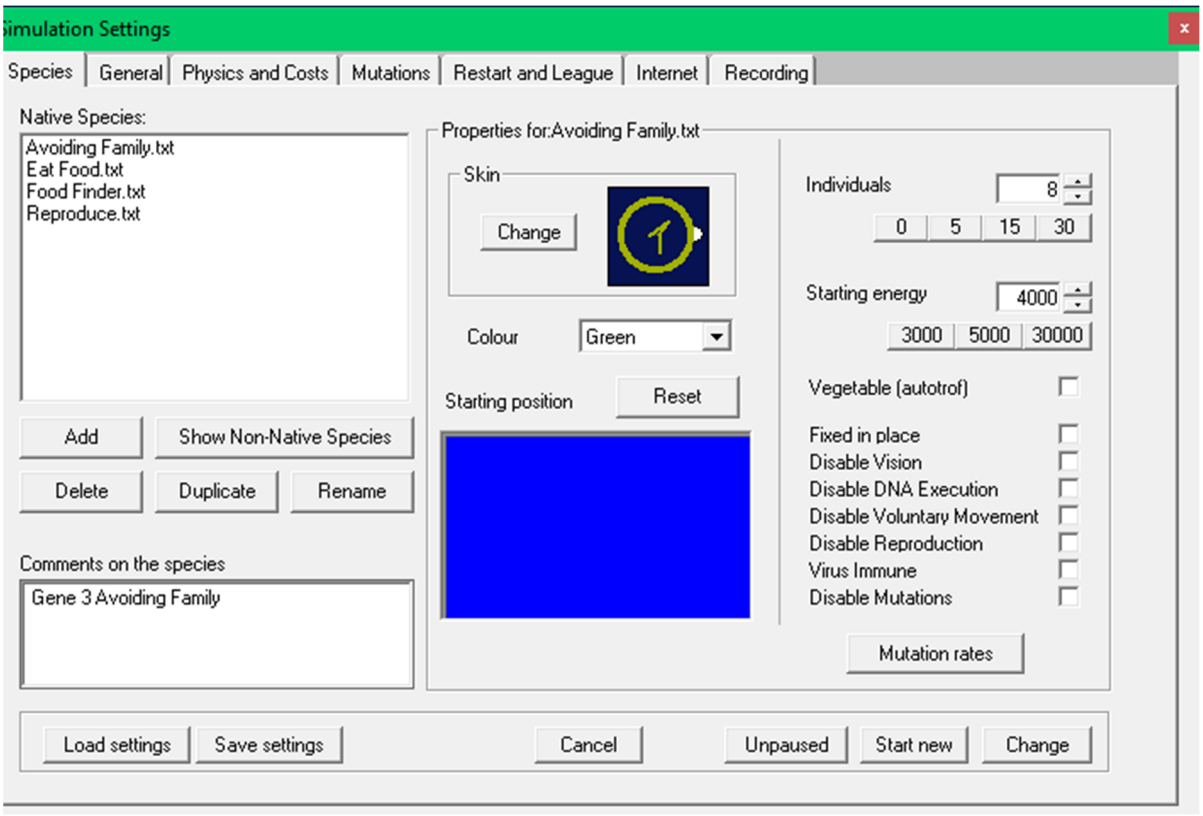

Рис. 10.39. Параметры симуляции

3. Установите в поле *Individuals* 8 особей, а в *Starting energy* – 4 000.

4. Нажмите *Start new*, и на поле моделирования можно наблюдать за ботами с разными наборами генов (рис. 10.40).

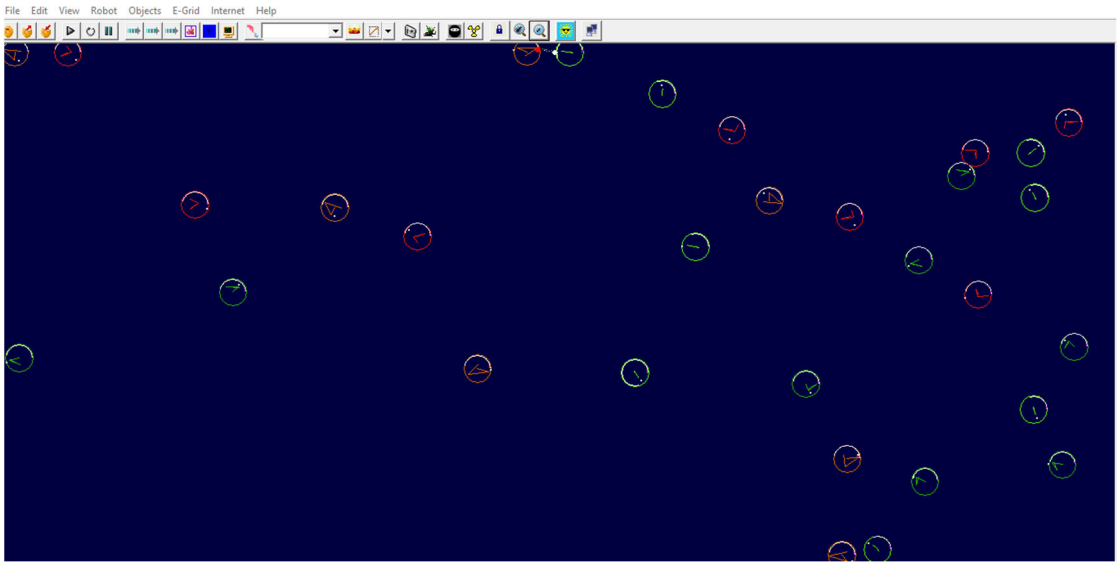

Рис. 10.40. Боты

5. Далее представлено поведение ботов в зависимости от генетического алгоритма. Боты с геном «Поедание пищи» показаны на рис. 10.41. Энергия ботов пополняется путем поглощения других ботов. Некоторые боты накапливают энергию намного быстрее, но в дальнейшем доминирующей становится только одна особь (рис. 10.42).

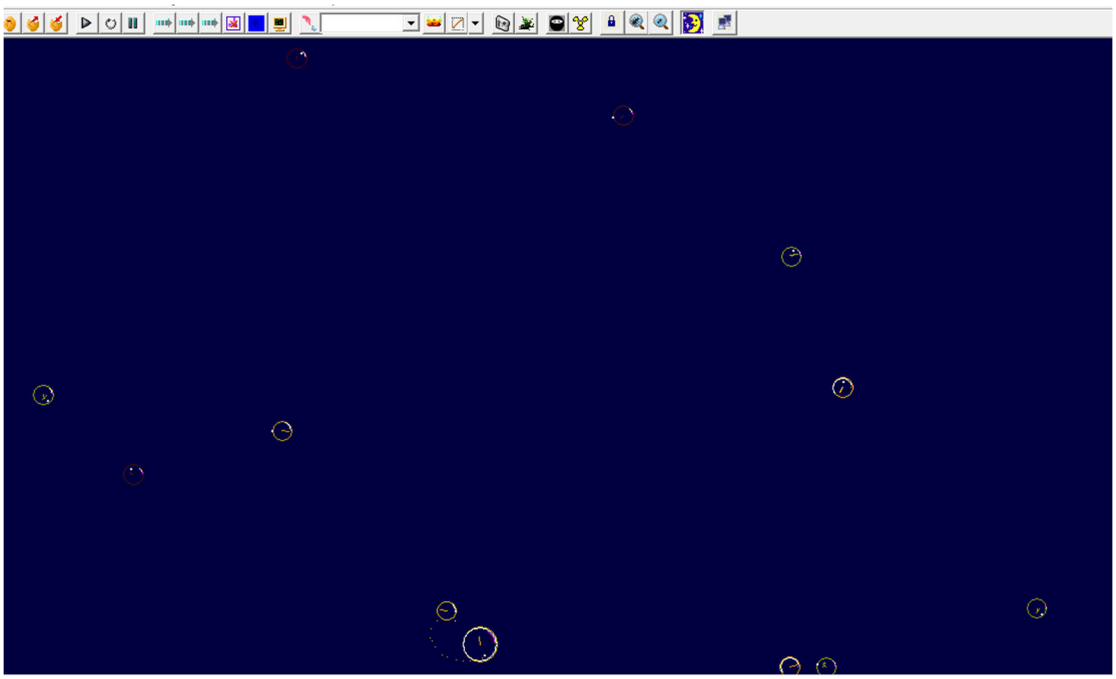

Рис. 10.41. Разновидность ботов «Поедание пищи»

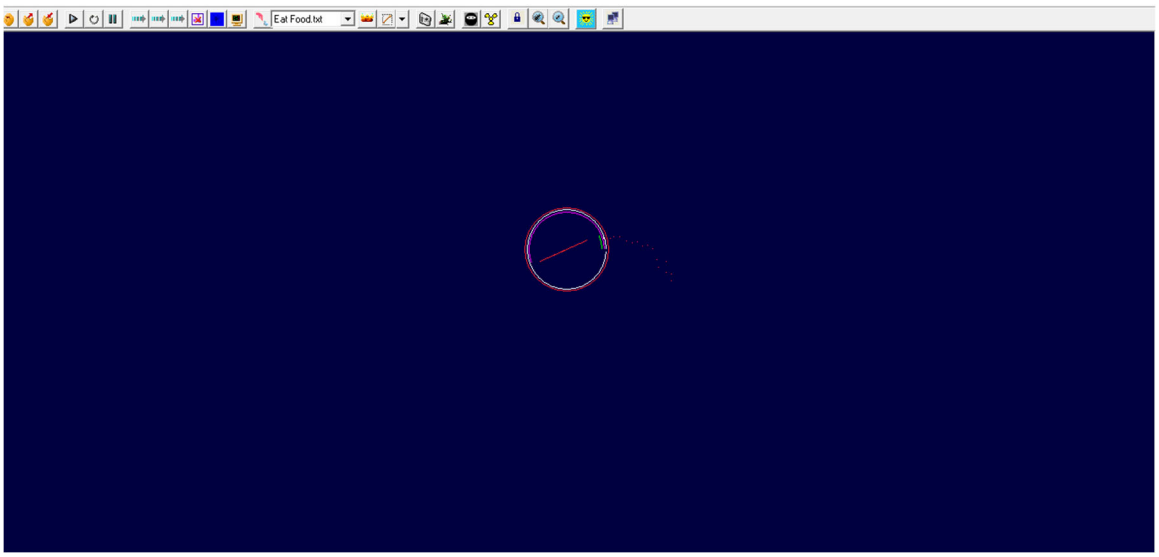

Рис. 10.42. Доминирующая особь

Алгоритм гена «Поедание пищи» показан на рис. 10.43.

```
' Gene 2 Eat Food
cond
*.eye5 50 >
*.refeye *.myeye !=
start
-1 .shoot store
*.refvelup .up store
stop
```

Рис. 10.43. Ген «Поедание пищи»

6. Алгоритм бота с геном «Поиск пищи» (рис. 10.44). Как и в случае с первой разновидностью, боты с геном «Поиск пищи» накапливают энергию путем поглощения других особей, но их реакция несколько медленнее, так как бот атакует только тогда, когда другая особь находится в непосредственной близости.

```
' Gene 1 Food Finder
cond
*.eye5 0 >
*.refeye *.myeye !=
start
*.refveldx .dx store
*.refvelup 30 add .up store
stop
```

Рис. 10.44. Ген «Поиск пищи»

Разновидность ботов «Поиск пищи» показана на рис. 10.45.

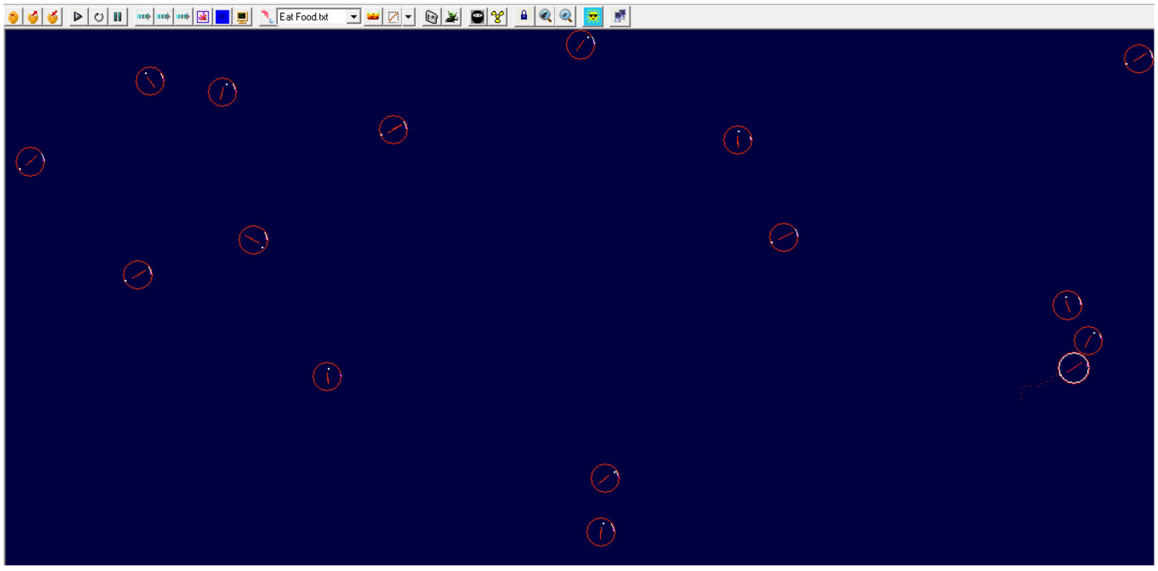

Рис. 10.45. Разновидность ботов «Поиск пищи»

7. Боты с геном «Избегание объединения» (рис. 10.46) избегают взаимодействия с идентичными особями. В условиях, когда есть и другие разновидности, ген позволяет более рационально осматривать окрестности мира в поисках пищи.

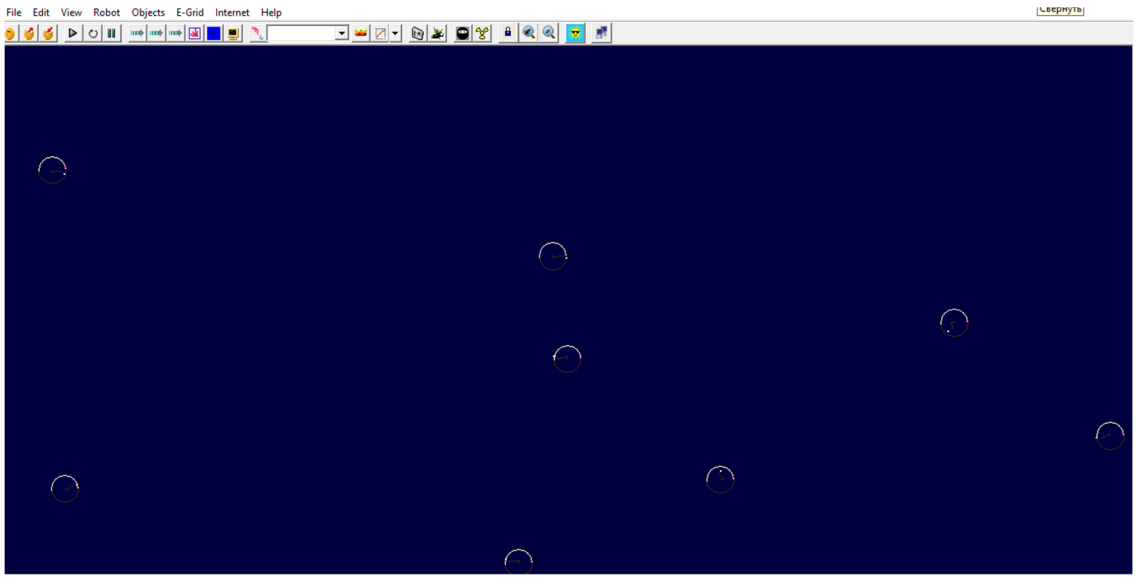

Рис. 10.46. Разновидность ботов «Избегание объединения»

8. Боты с геном «Размножение» (рис. 10.47). Размножение начинается, когда энергия становится выше уровня, определенного в ДНК. Если новое поколение бота велико и забирает всю энергию родителя, родитель погибает.

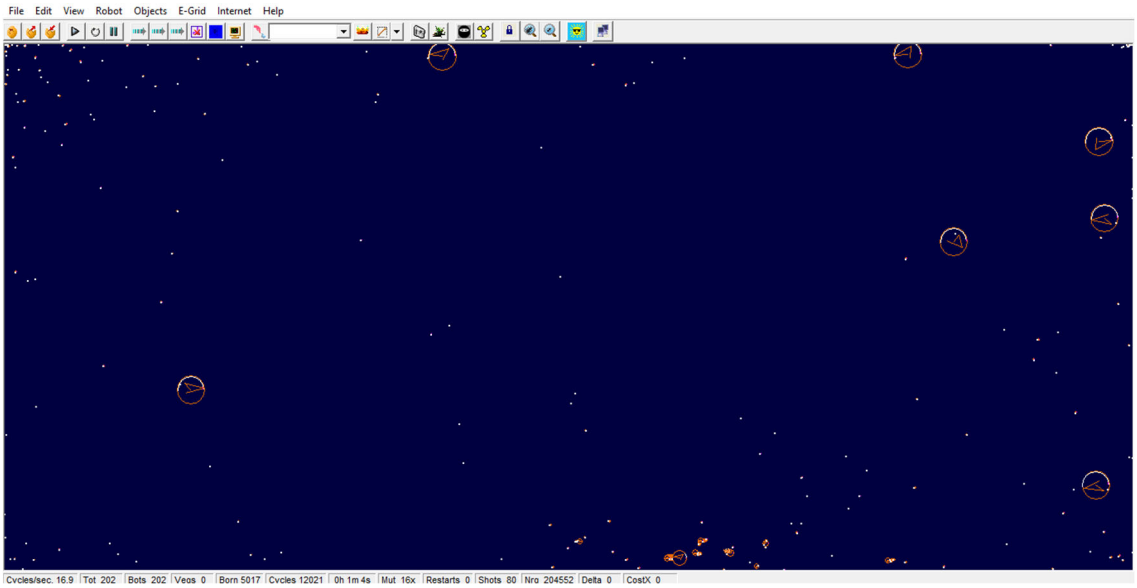

Рис. 10.47. Разновидность ботов «Размножение»

9. Бот в программе *DarwinBots* будет иметь код, показанный на рис. 10.48.

```
start
 100 *.nrg 1000 sub sgn 0 floor 500 *.body sub sgn 0 floor mult .strbody
mult store 100 500 *.nrg sub sgn 0 floor .fdbody mult store 200 *.shell sub 0
floor 200 ceil 200 *.shell sub sgn abs .mkshell mult store 10 *.venom sub 0 floor
10 ceil *.venom 10 sub sgn abs .strvenom mult store *.tiepres *51 *.tiepres sub
sgn abs *.tiepres sgn mult .deltie mult store *51 *.refeye sgn -1 mult 1 add
*.refeye *.myeye sub sgn abs *.eye5 35 sub 0 floor sgn mult mult .tie mult store
*51 *.tie sgn *.tiepres *51 sub sgn abs -1 mult 1 add add sgn .tienum mult store
-1 *.tie sgn *.tiepres *51 sub sgn abs -1 mult 1 add add sgn .tieloc mult store -
1000 *.tie sgn *.tiepres *51 sub sgn abs -1 mult 1 add add sgn .tieval mult store
1 *.robage sgn 1 sub abs .tie mult store .mkshell *.vloc sgn -1 mult 1 add .vloc
mult store -100 *.venval sgn 1 add .venval mult store 999 rnd 1 add *51 sgn -1
mult 1 add 51 mult store 200 *.aim add *.refeye *.myeye sub sgn abs -1 mult 1
add .setaim mult store *.maxvel *.vel sub 0 floor *.robage sgn *.refeye *.myeye
sub sgn abs mult .up mult store *.refxpos *.refypos angle *.robage sgn *.refeye
*.myeye sub sgn abs *.eye5 sgn mult mult .setaim mult store *.refveldx *.refeye
*.myeye sub sgn abs *.eye5 sgn mult .dx mult store *.refeye *.myeye sub sgn
abs -1 mult 1 add 50 mult inc 0 *50 30 sub 0 floor sgn .refeye mult store 0 *50
30 sub 0 floor sgn 50 mult store *.refshell 1 sub 0 floor sgn 3 mult 5 sub *.refeye
sgn mult 1 sub *.refeye *.myeye sub sgn abs *.eye5 35 sub 0 floor sgn mult .shoot
mult store *.shoot 3 add sgn 10 mult 0 floor *.refeye sgn 30 mult add *.refshell
1 sub 0 floor sgn 20 mult sub *.shoot sgn abs .shootval mult store 50 *.body 250
sub sgn 0 floor *.nrg 2000 sub sgn 0 floor *.eye5 30 sub 0 floor sgn -1 mult 1
add mult mult .repro mult store *.thisgene -1 mult 3 add abs *.genes 1 sub sgn
.delgene mult store
 stop
 end
```

Рис. 10.48. Бот в программе *DarwinBots*

10. Код бота для ДНК состоит из одного или нескольких генов. Они исполняются вместе за один цикл моделирования. Гены будут представлены в виде условий, и могут быть созданы в следующем виде, который показан на рис. 10.49.

```
Cond
 <одно или более условий>
 Start
 <операции внутри гена>
 Stop
или
Cond
 <одно или более условий>
 Else
 <операции внутри гена>
 Stop
```

Рис. 10.49. Гены в программе *DarwinBots*

11. Команда *Start* служит для выполнения операций, если все условия верны, а команда *Else* – если все условия ложны. В кодах ботов могут применяться следующие операции сравнения:

1) = (равно);
2) != (не равно);
3) %= (приближенно равно – возвращает истину, если первое число отличается от второго не более, чем на 10 %);
4) !%= (выдает противоположный результат предыдущего сравнения);
5) > (больше);
6) < (меньше);

7) >= (больше или равно);

8) <= (меньше или равно);

9) ~= (использует три числа, возвращает истину, если второе число больше первого на третье число процентов);

10) !~= (выдает противоположный результат предыдущего сравнения).

12. Операции состоят из последовательности операндов и знаков для записи набора. Для разделения операндов в выражении записей служат пробелы. Выражение читается слева направо. Операции выполняются в соответствии с двумя последними встретившимися перед ним операндами, когда в выражении встречается знак операции. Выражение вычисляется по тому же правилу, если результат операции заменяет в выражении последовательность ее операндов, а также ее знак. Результатом вычисления выражения становится результат последней вычисленной операции.

13. Программа *DarwinBots* позволяет собирать статистику в ходе симуляции и представлять ее в виде графиков.

В данной работе был создан график среднего количества мутаций за промежуток времени, выраженный определенным числом точек сбора данных симуляции (рис. 10.50).

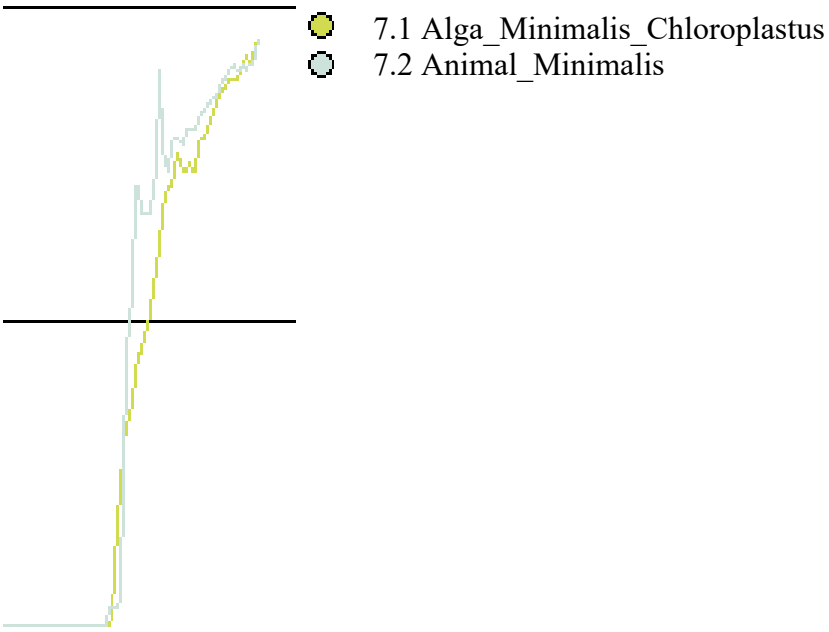

Рис. 10.50. График среднего числа мутаций

Также статистика хода симуляции отражается в строке состояния программы. В этой работе:

– всего объектов – 549;

– всего роботов – 399;

– всего растений – 160;

– общее число циклов – 1 074.

Время – 1 мин 48 с.

Эксперименты для различных вариантов мутаций

1. Снимайте результаты мутаций через одинаковые промежутки времени (количество точек сбора данных информации – 50, через произвольные промежутки времени (от 3 до 15 с)).

Эксперимент 1 (рис. 10.51 и 10.52). Описание эксперимента:

– количество особей – 10;

– энергия – 1 000;

– скорость – высокая;

– тип – Animal_Minimalis.

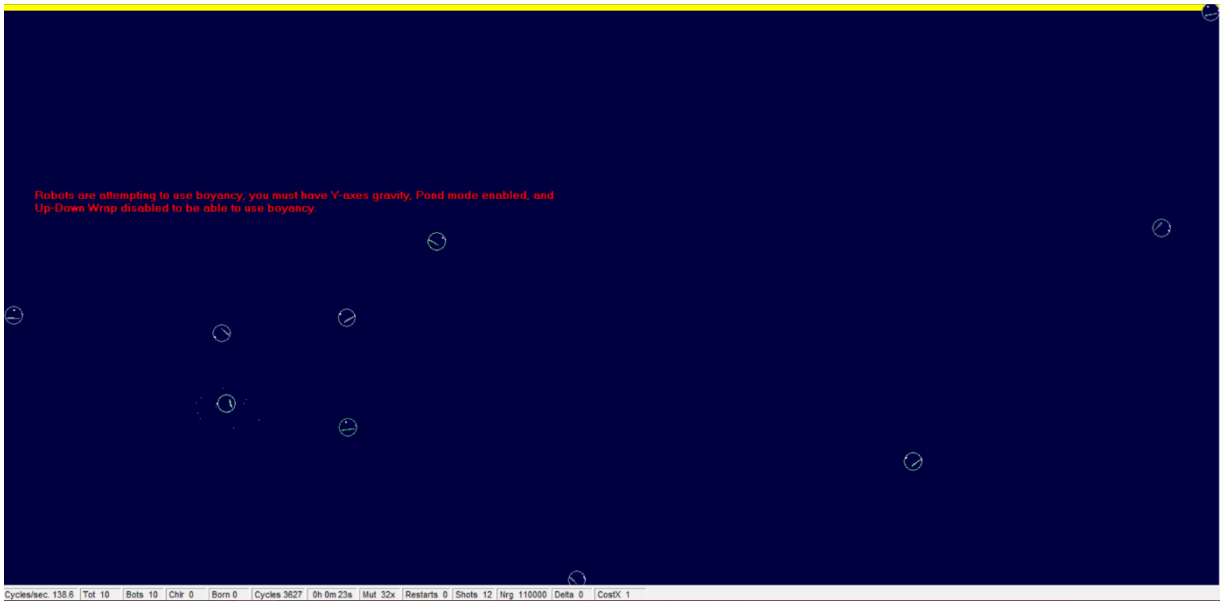

Рис. 10.51. Начало мутации

2. Таким образом, из 10 особей, которые сначала размножились, осталось всего 3, оказавшихся более приспособленными.

Рис. 10.52. Результаты

Результаты мутаций представлены в табл. 10.8.

Таблица 10.8

Результаты эксперимента 1

Cycles/sec.	Tot	Bots	Vegs	Born	Cycles	Time, c	Shots	Nrg
160,3	10	10	0	2	2674	15	1	109025
158,8	10	10	0	95	5345	30	0	104816
148,4	10	10	0	237	7868	45	80	101555
151,1	10	10	0	255	10424	1	12	90106
157,7	10	10	0	256	13083	1,15	13	65087
162,6	9	9	0	256	15815	1,30	11	54271
161,7	9	9	0	256	18522	1,45	11	53991
162,1	9	9	0	256	21255	2	11	53468
166,5	7	7	0	256	24343	2,15	11	42653
203,7	5	5	0	256	27517	2,30	22	28671
174,9	5	5	0	256	30208	2,45	11	28671
177,2	5	5	0	256	33196	3	11	28671
177,1	5	5	0	256	36204	3,15	11	28158
174,7	5	5	10	256	39211	3,30	11	28158
186,1	5	5	0	256	42287	3,45	11	28158

Окончание табл. 10.8

Cycles/sec.	Tot	Bots	Vegs	Born	Cycles	Time, c	Shots	Nrg
177,2	5	5	10	256	45341	4	11	28155
177,5	5	5	0	256	48317	4,15	11	28155
176,3	5	5	0	256	51389	4,30	11	28155
178,2	4	4	0	256	54378	4,45	11	23302
181,3	4	4	0	256	57392	5	12	23302
180,2	4	4	0	256	60408	5,15	12	23302
180,7	4	4	0	256	63414	5,30	12	23302
181,2	4	4	0	256	66478	5,45	12	22102
180,8	4	4	0	256	69486	6	12	21689
180,6	4	4	0	256	72534	6,15	12	21689
180,4	4	4	0	256	75555	6,30	12	21689
181,3	4	4	0	256	78589	6,45	12	21689
180,8	4	4	0	256	81548	7	12	21689
180,2	4	4	0	256	84515	7,15	12	21689
179,6	4	4	0	256	87557	7,30	12	21569
176,4	4	4	0	256	90520	7,45	12	21567
178,9	4	4	0	256	93608	8	12	21567
180,2	4	4	10	256	96611	8,15	12	21567
179,8	4	4	0	256	99592	8,30	12	21567
180,7	4	4	0	256	103550	8,45	12	21567
179,4	4	4	0	256	105532	9	12	21567
179,3	4	4	10	256	108521	9,15	12	21567
180	4	4	0	256	111523	9,30	12	21567
182,8	3	3	0	256	114522	9,45	12	15428
183,7	3	3	0	256	117629	10	12	15424
183,8	3	3	0	256	120658	10,15	12	15424
191,2	3	3	0	256	123795	10,30	12	15424
182,9	3	3	0	256	126865	10,45	12	15428
183,5	3	3	0	256	129979	11	12	15424
183,4	3	3	0	256	133026	11,15	12	15424
183,6	3	3	0	256	136093	11,30	12	15424
183,2	3	3	0	256	139095	11,45	12	15424
183,2	3	3	0	256	142,175	12	12	15424
183,6	3	3	0	256	145265	12,15	12	15424
181,9	3	3	0	256	148277	12,30	12	15424

3. Эксперимент 2 (рис. 10.53 и 10.54). Описание эксперимента:

– количество особей – 15;
– энергия – 3 000;
– скорость – высокая;
– тип – Martian Tank 3.

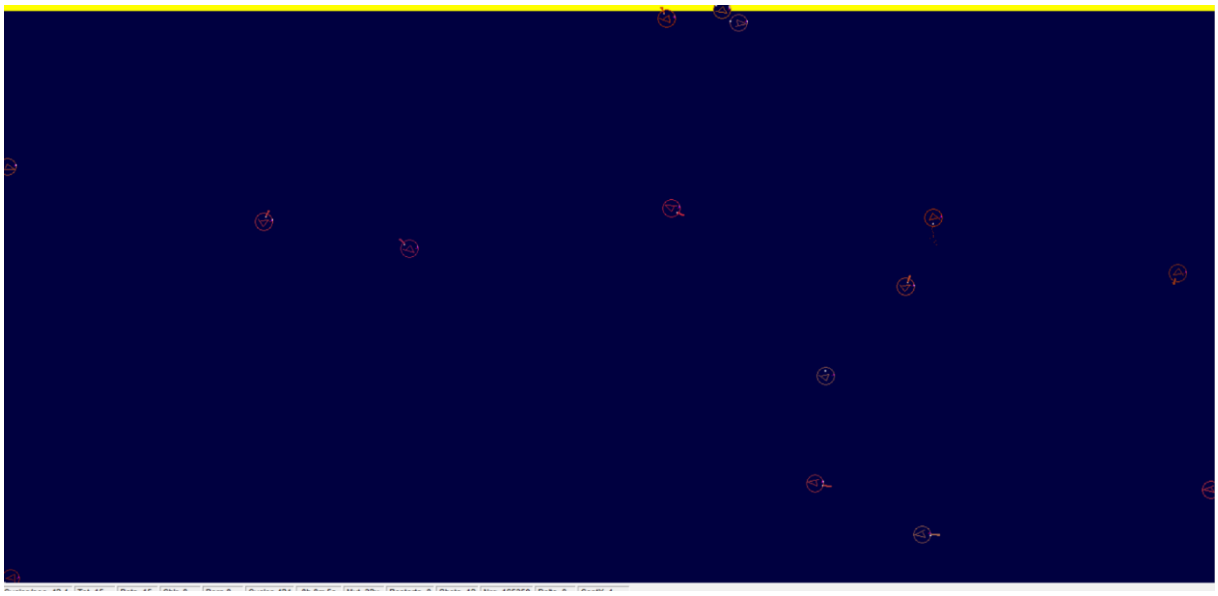

Рис. 10.53. Начало мутации

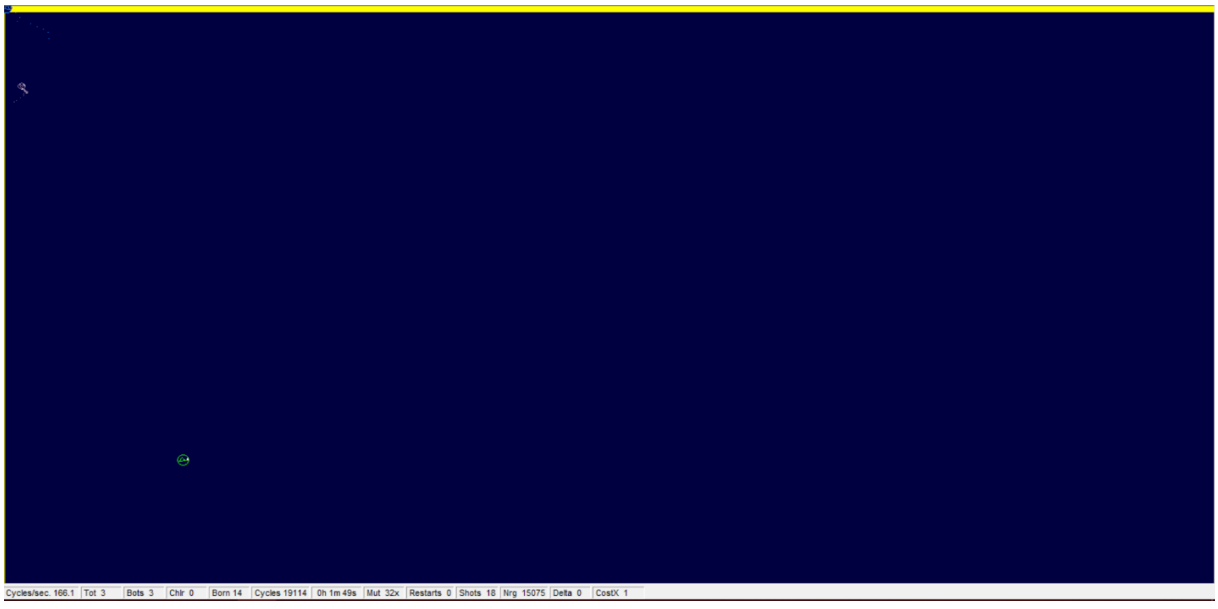

Рис. 10.54. Результаты

Результаты мутаций представлены в табл. 10.9.

Таблица 10.9

Результаты эксперимента 2

Cycles/sec.	Tot	Bots	Vegs	Born	Cycles	Time, c	Shots	Nrg
195	10	10	0	0	2149	10	0	110000
180,5	10	10	0	0	4133	20	0	110000
181	10	10	0	0	6128	30	0	110000
177,2	10	10	0	0	8080	40	0	110000
179,3	10	10	0	0	10037	50	0	110000
179,7	10	10	0	0	12067	1	0	110000
177,7	10	10	0	0	14067	1,10	0	110000
179,5	10	10	0	0	16075	1,20	0	110000
179,7	10	10	0	0	18086	1,30	0	110000
175,3	10	10	0	0	20014	1,40	0	110000
176,4	10	10	0	0	22015	1,50	0	110000
179,3	10	10	0	0	24019	2	0	110000
179,6	10	10	0	0	25990	2,10	0	110000
178,9	10	10	0	0	27961	2,20	0	110000
175,5	10	10	0	0	29921	2,30	0	110000
180,3	10	10	0	0	31892	2,40	0	110000
181,5	8	8	0	0	33903	2,50	0	90191
183,6	8	8	0	0	36003	3	0	90191
182	8	8	0	0	38012	3,10	0	90191
182,1	8	8	0	0	40043	3,20	0	90191
181,7	7	7	0	0	42084	3,30	0	80287
183,4	7	7	0	0	44101	3,40	0	80283
180,9	7	7	0	0	46171	3,50	0	80283
180,7	7	7	0	0	48211	4	0	80283
184,1	7	7	0	0	50274	4,10	0	80283
183,2	7	7	0	0	52305	4,20	0	80283
182,2	7	7	0	0	54356	4,30	0	80280
179,1	7	7	0	0	56377	4,40	0	80280
180,2	7	7	0	0	58400	4,50	0	80280
182,7	7	7	0	0	60432	5	0	80280
183,2	6	6	0	0	62433	5,10	0	70356
183,7	6	6	0	0	64479	5,20	0	70353
184,9	6	6	0	0	66513	5,30	0	70353
185,4	6	6	0	0	68597	5,40	0	70353
181,1	6	6	0	0	70652	5,50	0	70353
184,5	6	6	0	0	72692	6	0	70353

Окончание табл. 10.9

Cycles/sec.	Tot	Bots	Vegs	Born	Cycles	Time, c	Shots	Nrg
183,7	6	6	0	0	74764	6,10	0	69904
182,9	6	6	0	0	76841	6,20	0	69904
184,4	6	6	0	0	78893	6,30	0	69492
181,4	6	6	0	0	80977	6,40	0	69492
181,5	6	6	0	0	83064	6,50	0	69492
183,9	6	6	0	0	85057	7	0	69492
184,4	6	6	0	0	87079	7,10	0	69492
183,8	5	5	0	0	89086	7,20	0	59568
186	5	5	0	0	91174	7,30	0	59568
183,5	5	5	0	0	93224	7,40	0	59568
186,2	5	5	0	0	95321	7,50	0	59568
186,3	5	5	10	0	97355	8	0	59568
185,3	5	5	0	0	99282	8,10	0	59341
182,9	5	5	0	0	101470	8,20	0	59341

4. Эксперимент 3 (рис. 10.55 и 10.56). Описание эксперимента:

– количество особей – 5;

– энергия – 3 000;

– скорость – высокая;

– тип – Republican Wasp.

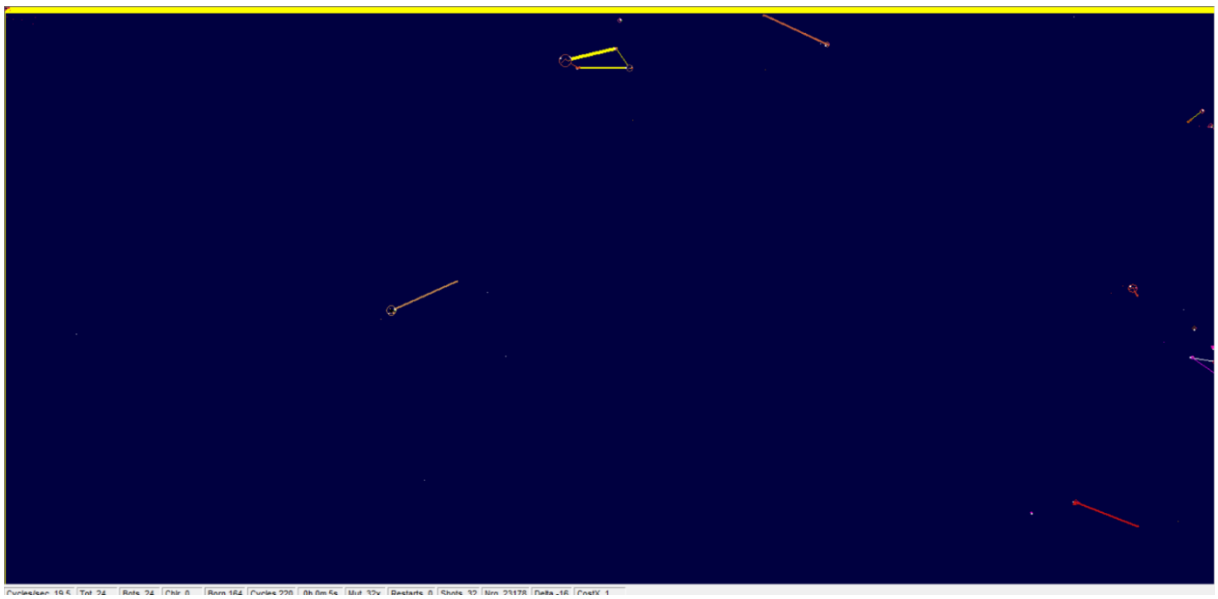

Рис. 10.55. Начало мутации

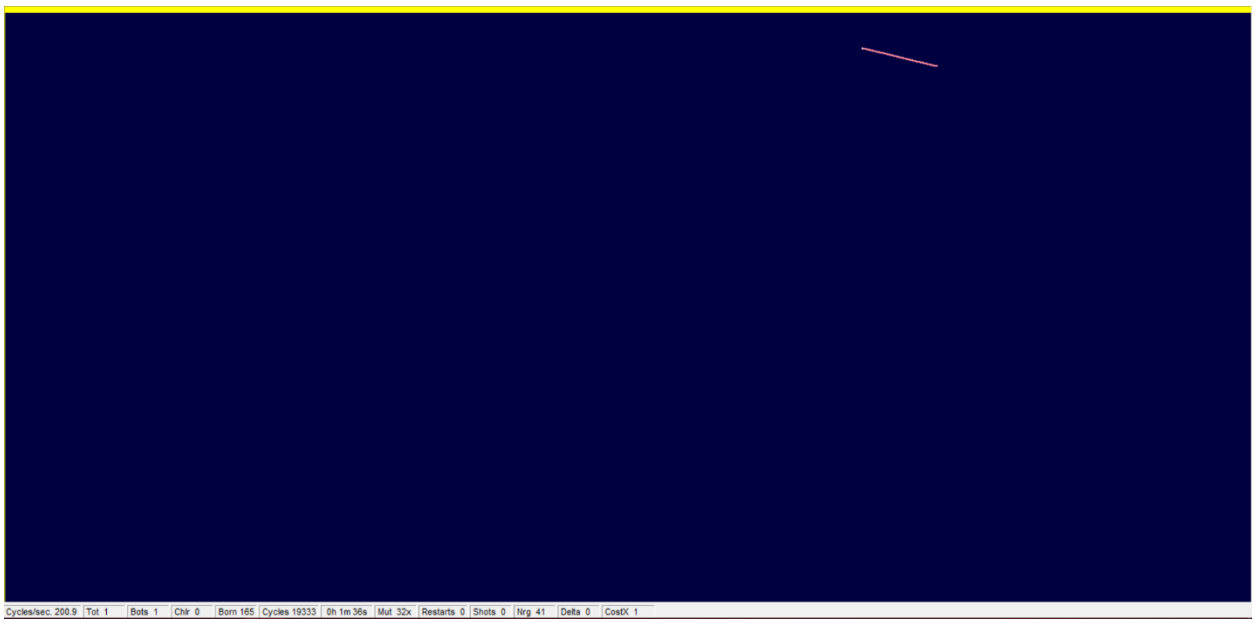

Рис. 10.56. Результаты

Результаты мутаций представлены в табл. 10.10.

Таблица 10.10

Результаты эксперимента 3

Cycles/sec.	Tot	Bots	Vegs	Born	Cycles	Time, c	Shots	Nrg
180,4	5	5	0	0	1989	10	0	180000
170,4	5	5	0	0	3833	20	0	180000
172	5	5	0	0	5727	30	0	180000
172,6	5	5	0	0	7642	40	0	180000
168,4	5	5	0	0	9518	50	0	180000
172	5	5	0	0	11487	1	0	180000
169,4	5	5	0	0	13406	1,10	0	180000
172,7	5	5	0	0	15307	1,20	0	180000
172,3	5	5	0	0	17255	1,30	0	180000
172,7	5	5	10	0	19124	1,40	0	180000
172,7	5	5	0	0	21050	1,50	0	180000
171,6	5	5	10	0	22974	2	0	180000
172,7	5	5	10	0	24894	2,10	0	180000
172,5	5	5	10	0	26804	2,20	0	180000
173	5	5	0	0	28746	2,30	0	180000

Окончание табл. 10.8

Cycles/sec.	Tot	Bots	Vegs	Born	Cycles	Time, c	Shots	Nrg
171,6	5	5	0	0	30681	2,40	12	180000
171,5	5	5	0	0	32521	2,50	12	180000
166,3	5	5	0	0	34415	3	12	180000
170,6	5	5	0	0	36329	3,10	12	180000
168,1	4	4	0	0	38257	3,20	12	169336
174,2	4	4	0	0	40209	3,30	12	159514
174,4	4	4	0	0	42168	3,40	12	159514
174,3	4	4	0	0	44076	3,50	12	159514
176,4	4	4	10	0	46054	4	12	139853
177,5	4	4	0	0	47975	4,10	12	139853
190	4	4	0	0	50088	4,20	12	139853
178	4	4	0	0	52079	4,30	12	139853
177,9	4	4	10	0	54094	4,40	12	139853
177,7	4	4	0	0	56102	4,50	12	139853
177,5	4	4	0	0	58098	5	12	139853
177,8	4	4	0	0	60067	5,10	12	130035
179,2	4	4	0	0	62078	5,20	12	130035
179,2	4	4	0	0	64092	5,30	12	130035
176,9	4	4	0	0	66082	5,40	12	120220
177,7	4	4	0	0	68105	5,50	12	120220
180	4	4	10	0	70154	6	12	120220
190,5	4	4	0	0	72205	6,10	12	120220
179,9	4	4	0	0	74183	6,20	12	110409
177,5	4	4	0	0	76242	6,30	12	110409
178,7	4	4	0	0	78229	6,40	12	100674
175,2	4	4	0	7	80257	6,50	25	87331
176	4	4	0	9	82188	7	12	82081
182,7	4	4	0	9	84202	7,10	10	76041
188,1	4	4	0	9	86286	7,20	10	56418
189,2	4	4	10	9	88454	7,30	10	56418
186,2	4	4	0	9	90523	7,40	10	56418
189,4	4	4	0	9	92672	7,50	10	56418
188,8	4	4	0	9	94781	8	10	56418
185,2	3	3	0	11	96902	8,10	11	34890
186,2	3	3	0	11	98975	8,20	10	34680

5. Самостоятельно проведите несколько экспериментов, устанавливая различные варианты мутаций и загружая разных ботов.

Контрольные вопросы

1. Перечислите основные параметры на вкладке *Species*.
2. Перечислите основные параметры на вкладке *General*.
3. Перечислите основные параметры на вкладке *Physics*.
4. Перечислите основные параметры на вкладке *Costs*.
5. Перечислите основные параметры на вкладке *Mutation rates*.
6. Какие гены для ботов существуют в программе *DarwinBots*?

10.5. Лабораторная работа № 5. Решение задач управления и наблюдения методами нечеткой логики

Время выполнения – 4 часа.

Цель работы: изучить инструментальные средства *FisPro* (Guaje Fuzzy), а также основы проектирования нечетких систем управления с помощью данного программного продукта.

Задачи работы

1. Создать нечеткую систему управления процессом подачи тепла в зависимости от измеренного значения температуры в *FisPro*.
2. Научиться задавать входные (измеряемые) и выходные (вычисляемые) переменные.
3. Сформировать базу правил инструментальными средствами *FisPro*.
4. Построить поверхности выхода.
5. Создать в графической форме представления функции принадлежности для входной переменной.

Перечень обеспечивающих средств

Для выполнения работы необходимы:

– конспект лекций;

– персональный компьютер с установленными на нем приложениями Microsoft Office Word и FisPro.

Методика выполнения работы

Создание нечеткой системы управления процессом подачи тепла в зависимости от измеренного значения температуры в FisPro

1. Главное окно программы *FisPro* представлено на рис. 10.57. Для начала работы необходимо выполнить команду *Fis* → *New*. В поле *Name* необходимо задать имя для новой системы. В данном случае введите *Управление подачей тепла*.

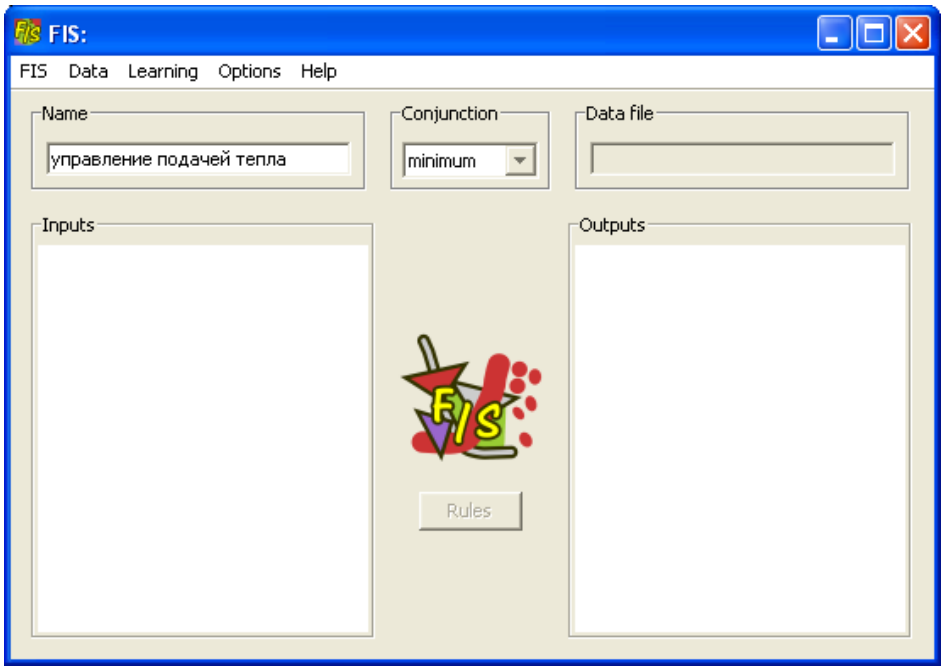

Рис. 10.57. Окно приложения *FisPro*

2. Теперь необходимо задать входные (измеряемые) и выходные (вычисляемые) переменные. Это можно сделать с помощью ввода данных в окна *Inputs* и *Outputs* программы *Fis*.

3. Далее в открывшемся окне необходимо задать имя переменной, например, *Температура*, и далее в меню *Range* требуется указать диапазон для изменения значений заданной переменной. Это диалоговое окно представлено на рис. 10.58.

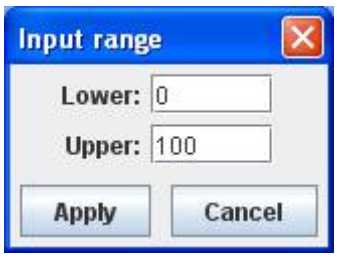

Рис. 10.58. Диапазон изменения значений
переменной *Температура*

4. После нажатия на кнопку *Apply* требуется выполнить команду *MFs* → *New MF*. Здесь необходимо установить термы и функции принадлежности переменной. Диалоговое окно для установки терм и функций показано на рис. 10.59. Название терма укажите в поле *Name*, в поле *Type* – тип функции принадлежности (*trapezoidal* – трапецеидальная функция принадлежности, *triangular* – треугольная и т. д.).

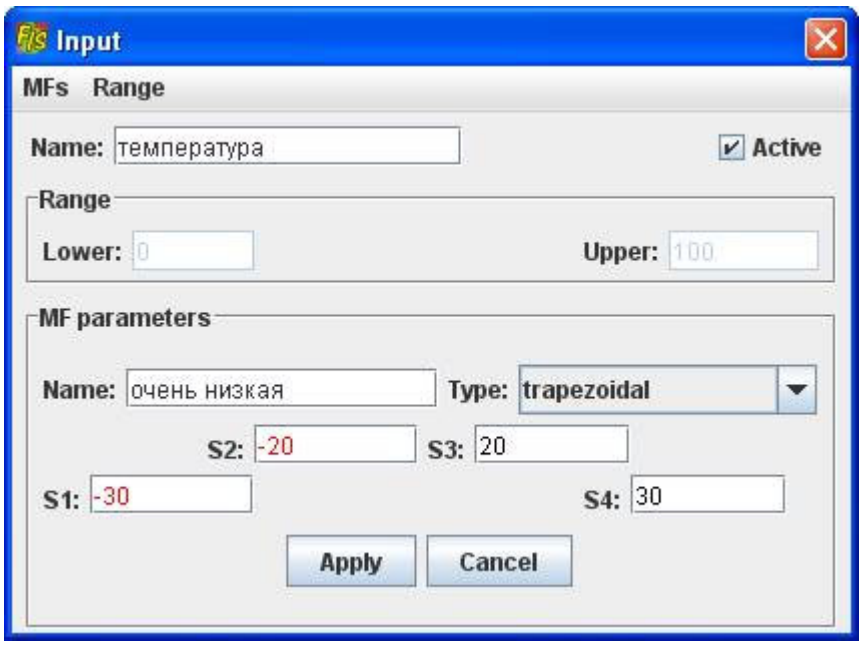

Рис. 10.59. Окно *Input*

5. Для лингвистической переменной *Температура* задайте следующие термы в соответствии с табл. 10.11.

Таблица 10.11

Термы для переменной *Температура*

Название терма (Name)	Тип функции принадлежности (Type)	Значение терма (Params)
Очень низкая	Трапецеидальная	[–30; –20; 20; 30]
Низкая	Треугольная	[10; 30; 50]
Средняя	Треугольная	[30; 50; 70]
Высокая	Треугольная	[50; 70; 90]
Очень высокая	Трапецеидальная	[70; 80; 120; 130]

6. Диалоговое окно для редактора функций принадлежности переменной *Температура* представлено на рис. 10.60.

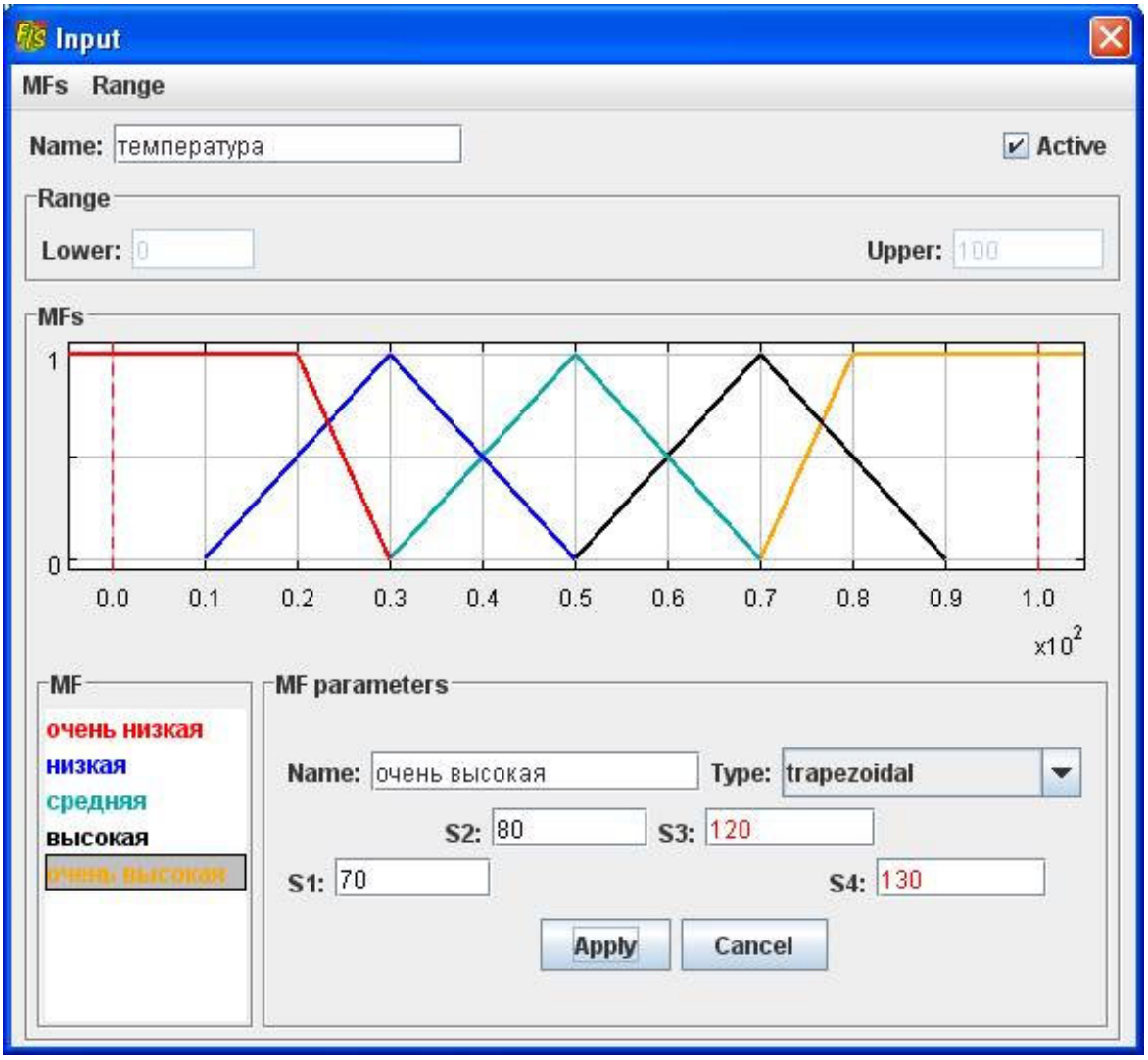

Рис. 10.60. Термы и функции принадлежности переменной *Температура*

7. Также необходимо задать термы и определить функции принадлежности для выходной переменной *Подача тепла*. Это показано на рис. 10.61.

Исходные данные приведены в табл. 10.12.

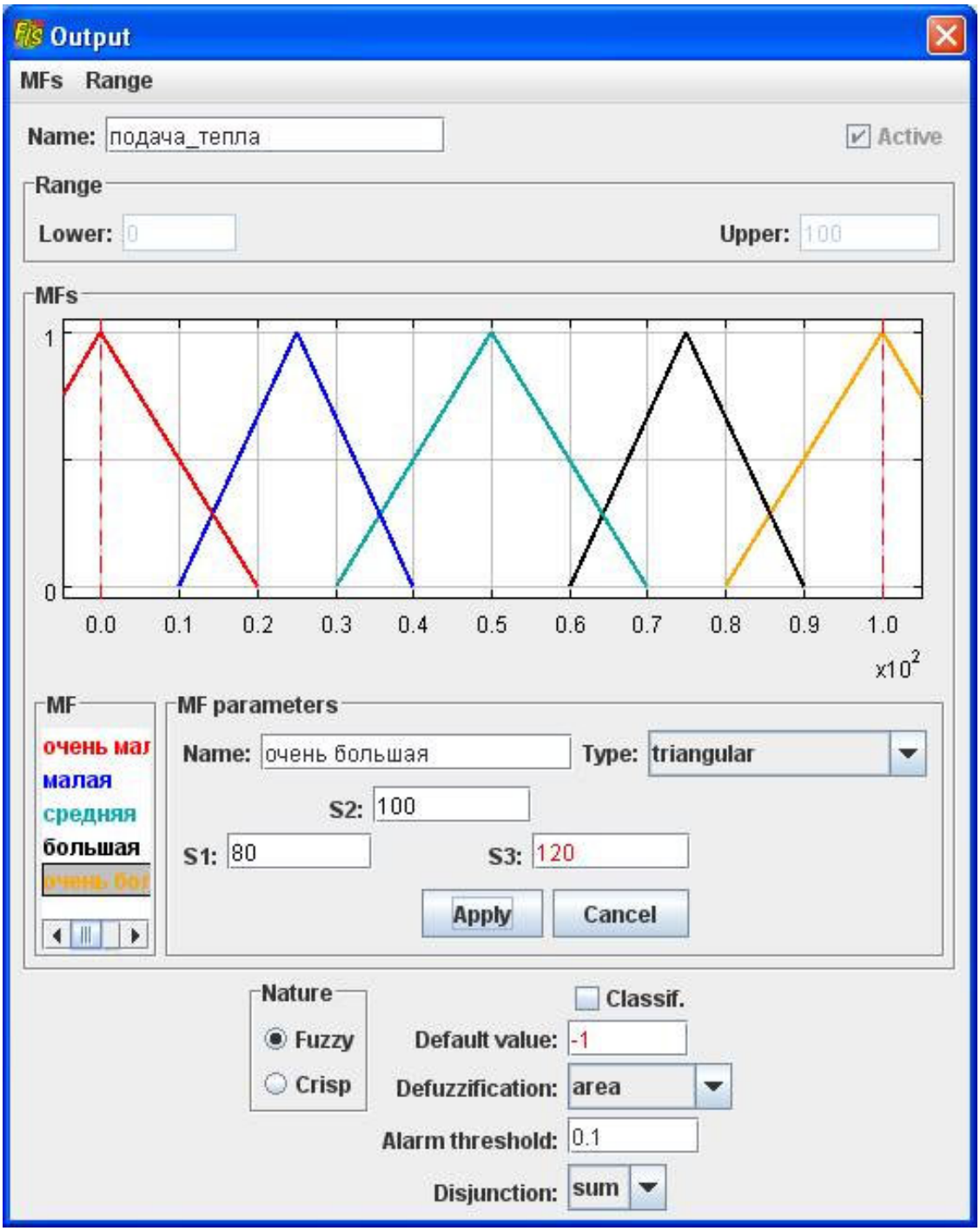

Рис. 10.61. Значения выходной переменной

Таблица 10.12

Исходные данные для выходной переменной *Подача тепла*

Название терма	Тип функции принадлежности	Значение терма
Очень малая	Треугольная	[–20; 0; 20]
Малая	Треугольная	[10; 25; 40]
Средняя	Треугольная	[30; 50; 70]
Большая	Треугольная	[60; 75; 90]
Очень большая	Треугольная	[80; 100; 120]

8. Для того чтобы создать базу правил инструментальными средствами *FisPro*, необходимо сформулировать предложения в формате «ЕСЛИ – ТО». В эти правила должна входить заданная переменная *Температура*. Таким образом, правила будут выглядеть так.

1. ЕСЛИ температура=очень низкая ТО подача воды=очень большая.
2. ЕСЛИ температура=низкая ТО подача воды=большая.
3. ЕСЛИ температура=средняя ТО подача воды=средняя.
4. ЕСЛИ температура=высокая ТО подача воды=малая.
5. ЕСЛИ температура=очень высокая ТО подача воды=очень малая.

9. Чтобы внести эти правила в базу правил, необходимо нажать на кнопку *Rules* в главном окне программы, далее выполнить команду *New Rule* меню *Rules*. На рис. 10.62 изображено окно редактора базы знаний после ввода пяти правил.

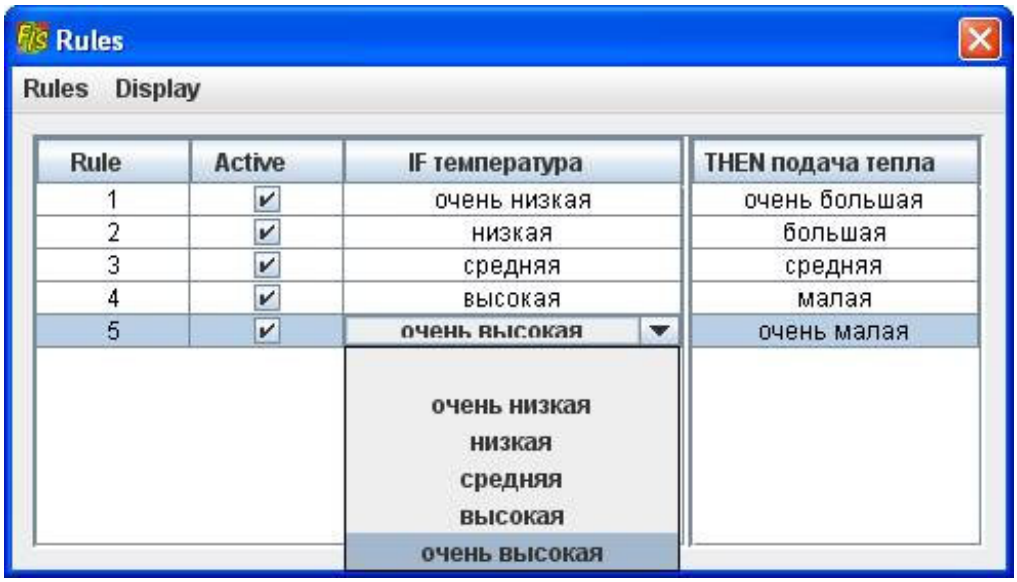

Рис. 10.62. Окно редактора базы правил

10. После создания базы правил требуется начать этап для логического вывода. Для этого требуется выполнить команду *Infer*, которая находится в меню *Fis* главного окна программы *FisPro*.

На данном диалоговом окне в левой части в графической форме представлены функции принадлежности для входной переменной *Температура*, а в правой – выходной переменной *Подача тепла* (рис. 10.63). Изменить значения входной переменной можно с помощью движения бегунка или же задавать числовые значения непосредственно в поле *Температура*.

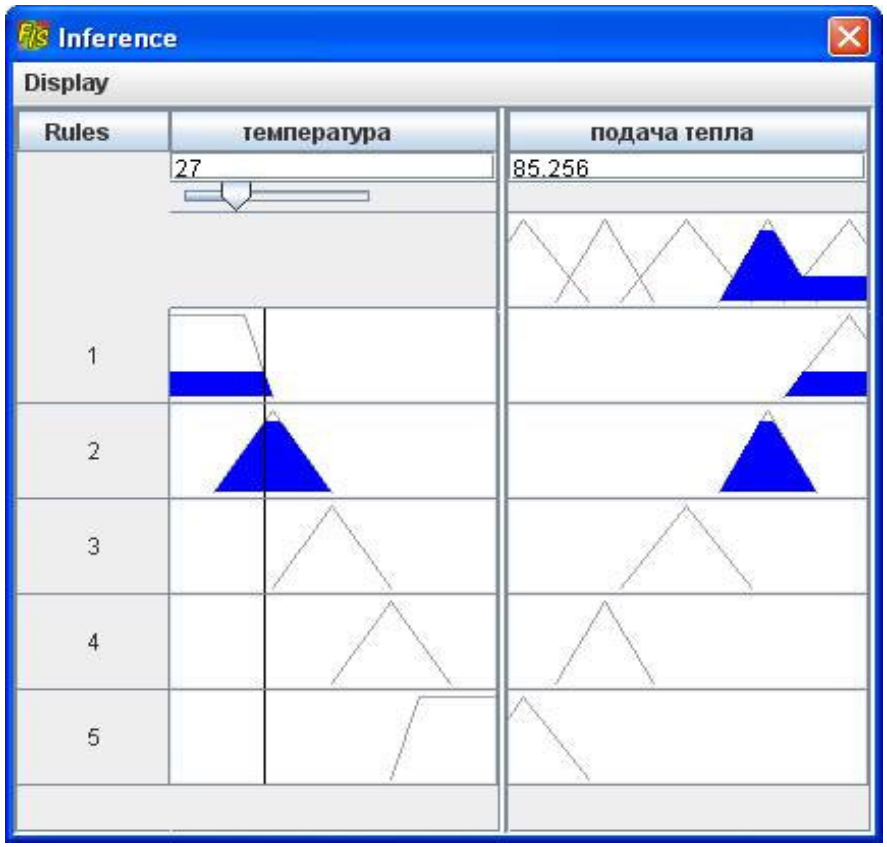

Рис. 10.63. Логический вывод

11. Чтобы просмотреть диалоговое окно поверхности выхода, требуется выполнить команду *System Response/Section* пункта меню *Fis* в главном окне программы. Это окно показано на рис. 10.64.

12. Самостоятельно создайте модель движения автомобиля по трассе и нечеткую модель контроля уровня воды в баке.

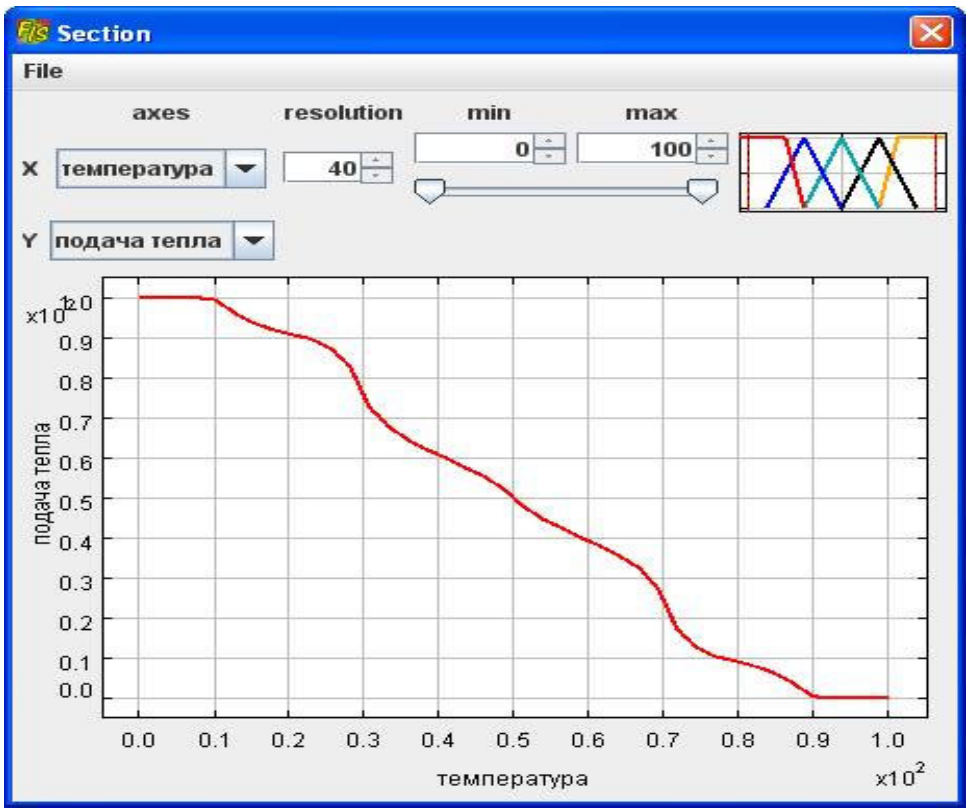

Рис. 10.64. Просмотр поверхности выхода

Контрольные вопросы

1. Как задать входные (измеряемые) и выходные (вычисляемые) переменные?
2. Как указать диапазон для изменения значений заданной переменной?
3. Как установить термы и функции принадлежности переменной?
4. Перечислите основные параметры для редактора функций принадлежности? В чем их назначение?
5. Как создать базу правил инструментальными средствами *FisPro*?
6. В чем назначение этапа логического вывода переменной?

10.6. Лабораторная работа № 6. Создание нечеткой системы для распознавания функции «исключающее "или"»

Время выполнения – 2 часа.

Цель работы: изучить инструментальные средства Neural Network Wizard, а также основы проектирования нечетких систем управления с помощью данного программного продукта.

Задачи работы

1. Создать набор данных и подготовить обучающие примеры.
2. Задавать входные и выходные переменные.
3. Использовать различные методы нормализации.
4. Устанавливать различные параметры конфигурации для нейронной сети.
5. Задавать параметры обучения для нейронной сети.

Методика выполнения работы

Для выполнения работы необходимы:

– конспект лекций;

– персональный компьютер с установленными на нем приложениями Microsoft Office Word и Neural Network Wizard.

Задания

Нейросети являются программами, работающими по самообучающимся алгоритмам, поэтому перед использованием их необходимо предварительно обучить. Обучение производится на основании специального набора данных – обучающей выборки. Для обучения нейросети необходимо подавать на ее вход данные, для которых заранее определены выходные значения. Эти данные находятся в текстовом файле специального вида.

1. В начале необходимо создать набор данных и подготовить обучающие примеры. Исходные данные для обучения нейронной сети должны быть представлены в текстовом файле с разделителем (*Tab* или пробел). Пример текстового файла показан на рис. 10.65.

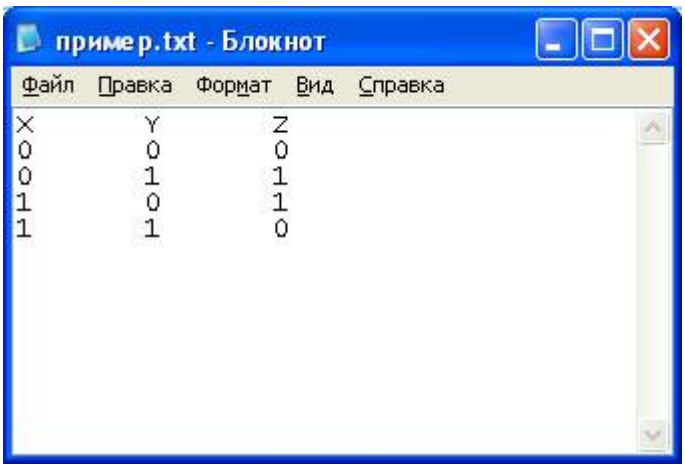

Рис. 10.65. Данные для обучения НС

2. Затем в диалоговом окне программы выберите файл, который будет содержать обучающую выборку. Это представлено на рис. 10.66. Данные, которые содержатся в этом файле, будут использованы для обучения сети. Здесь можно открыть файл формата *.txt для обучения нейронной сети или файл формата *.nnw. Это будет обученная нейронная сеть [17].

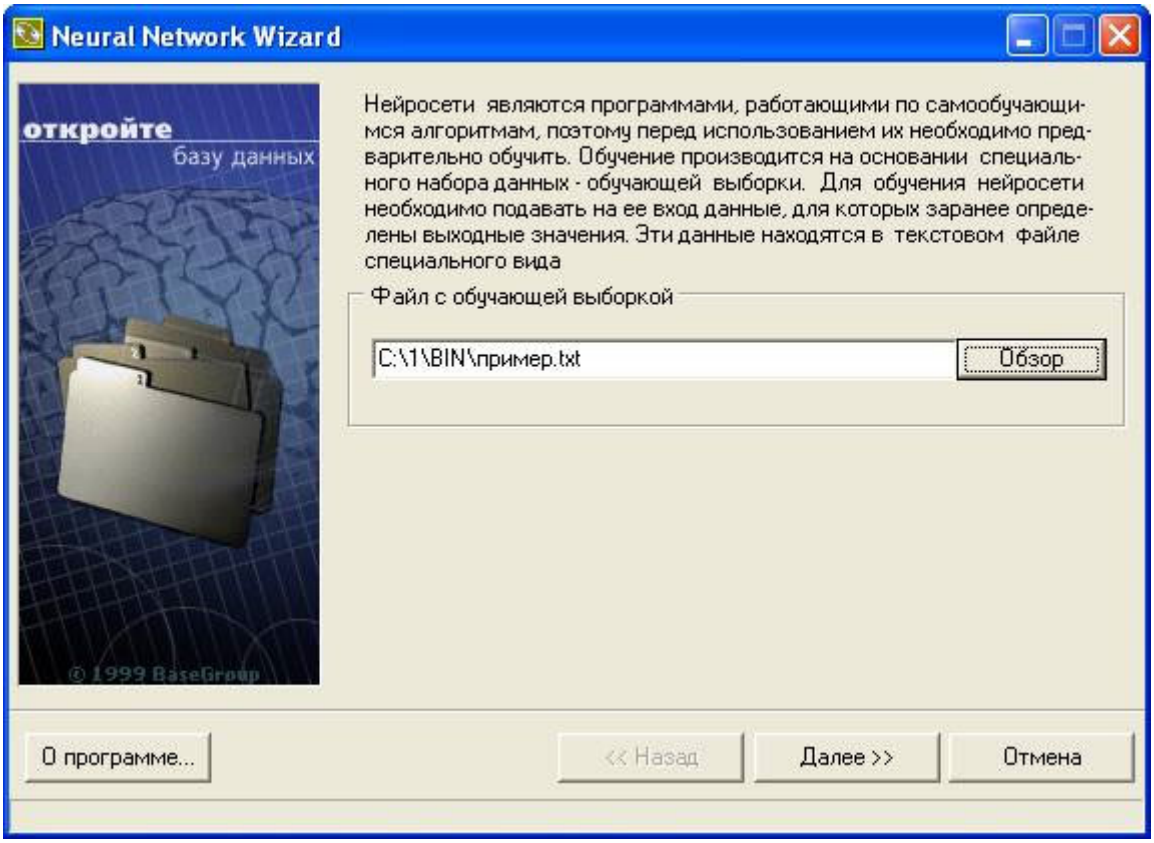

Рис. 10.66. Подключение txt-файла

3. В следующем диалоговом окне программы, которое представлено на рис. 10.67, можно задать входные и выходные переменные. Также здесь можно провести процедуру для нормализации данных.

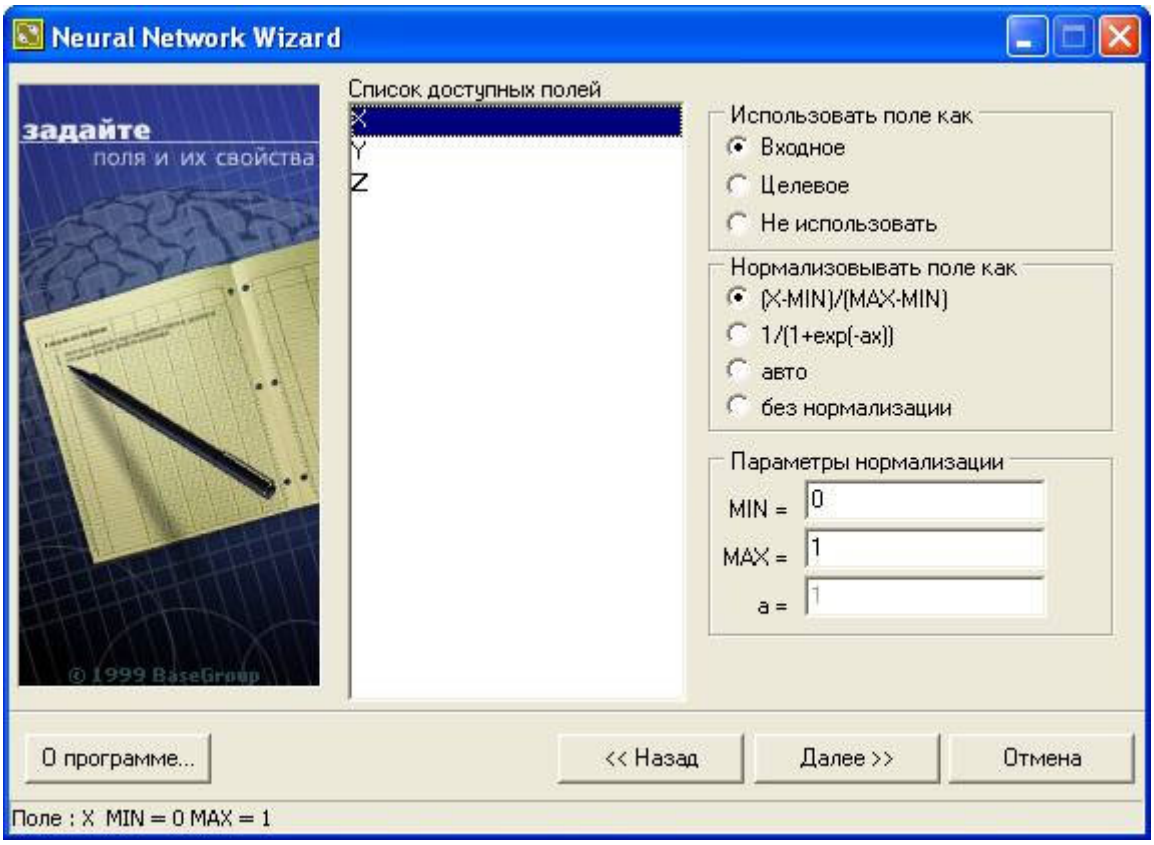

Рис. 10.67. Задание входных и выходных переменных

На этом диалоговом окне можно задать следующие параметры:

а) *Использовать поле как.* Как правило, нейронная сеть состоит из входного, выходного и скрытых слоев. Количество нейронов в первом и последнем слое зависит от того, сколько полей вы хотите определить как входные и выходные. Поля, которые отмечены как «не использовать», для обучения и тестирования нейронной сети применяться не будут [18];

б) *Нормализовать поле как.* На вход нейронной сети требуется подавать информацию в нормализованном виде. Это числа в диапазоне от 0 до 1. В программе реализованы следующие методы нормализации:

– (XMIN)/(MAX-XMIN) – линейная нормализация;

– 1/(1+exp(-ax)) – экспоненциальная нормализация;

– «Авто» – нормализация, основанная на статистических характеристиках выборки;

– «Без нормализации» – нормализация не производится;

в) *Параметры нормализации.* В данном случае задаются значения, которые используются в формулах нормализации.

В нашем случае входными данными будут являться *X* и *Y*, а выходной переменной – *Z*. Нормализация исходных данных производиться не будет.

4. На следующем этапе требуется создать нейронную сеть. Для этого в диалоговом окне, которое представлено на рис. 10.68, задаем параметры конфигурации для нейронной сети.

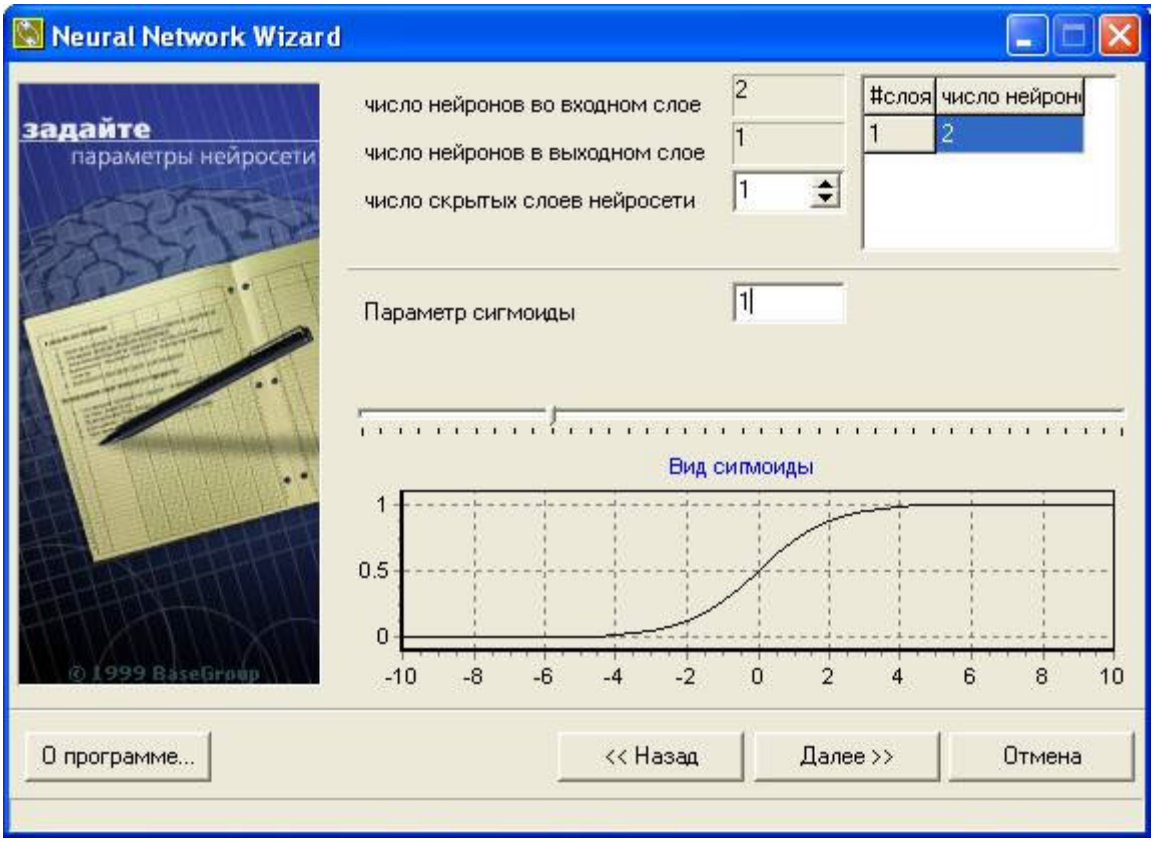

Рис. 10.68. Окно задания параметров нейронной сети

В данном диалоговом окне можно задать следующие параметры:

– *Число скрытых слоев нейросети* – нейронная сеть должна состоять из нескольких слоев: входного, выходного и скрытых (скрытых слоев может быть несколько). Данный параметр служит для указания количества скрытых слоев. Основного принципа для определения таких слоев нет, обычно задается от 1 до 3 скрытых слоя. Известно, что чем более нелинейная задача, тем больше скрытых слоев должно быть;

– *Слои, число нейронов* – данный параметр служит для установки количества нейронов в каждом скрытом слое. Основных правил для определения количества нейронов в скрытых слоях нет. Число связей между нейронами должно быть меньше количества примеров в обучающей выборке;

– *Параметр сигмоиды* – в Neural Network Wizard в качестве функции активации используется сигмоидальная функция (сигмоида). Сигмоида применяется для обеспечения нелинейного преобразования данных. В противном случае нейросеть сможет выделить только линейно разделимые множества. Чем выше параметр, тем больше функция активности походит на пороговую функцию. Параметр сигмоиды подбирается экспериментально [19].

Для рассматриваемого примера будет использована сеть с одним скрытым слоем, содержащим 2 нейрона (как известно, задача моделирования функции XOR является нелинейной).

5. Обучение сети происходит после создания нейронной сети. Для этого необходимо задать параметры обучения в окне на рис. 10.69.

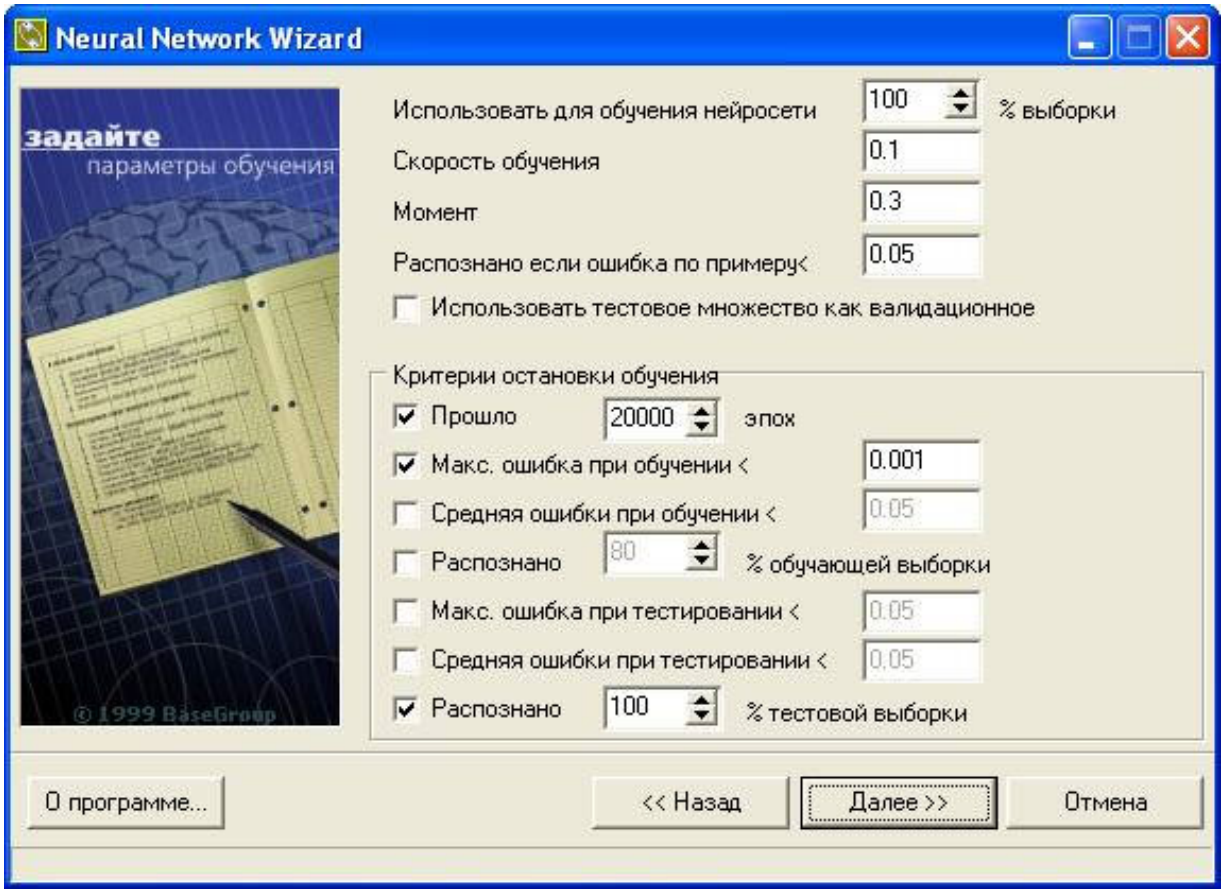

Рис. 10.69. Окно задания параметров обучения

В данном случае существуют следующие параметры обучения нейронной сети:

– *Использовать для обучения сети % выборки* – все примеры, которые будут подаваться на вход нейросети, будут делиться на два множества: обучающее и тестовое. Данный параметр предназначен для определения процентов примеров, которые будут использоваться для обучающей выборки. Записи, которые будут применяться для тестирования, выбираются случайно, но пропорции должны сохраняться;

– *Скорость обучения* – параметр должен определять амплитуду коррекции весов на каждом шаге обучения;

– *Момент* – параметр, который определяет степень воздействия i-й коррекции весов на i+1-ю;

– *Распознано если ошибка по примеру* < – если результат прогноза будет отличаться от значения для обучаемого множества на величину меньше указанной, то пример считается распознанным;

– *Использовать тестовое множество как валидационное* – при выборе этой опции обучение будет прекращено, как только ошибка на тестовом множестве начнет увеличиваться. Это помогает избежать ситуации переобучения сети;

– *Критерии остановки обучения* – осуществляется выбор условия завершения процесса обучения нейронной сети [20].

Далее, в окне на рис. 10.70, можно запустить процесс обучения и наблюдать ход обучения нейронной сети. Параметры обучения:

– *Пуск обучения/остановка обучения* – запуск процесса обучения. В таблице над кнопкой можно наблюдать, как меняется ошибка обучения;

– *Распределение ошибки* – на диаграмме отображается распределение ошибки. Зеленые столбцы – ошибка на обучающей выборке, красные – на тестовой выборке. Чем правее столбец, тем выше значение ошибки. Шкала от 0 до 1. Чем выше столбец, тем больше примеров с указанной ошибкой;

– *Распределение примеров в обучающей/тестовой выборке* – на этих графиках можно отслеживать, насколько результаты, предсказанные нейронной сетью, совпадают со значениями в обучающей (слева) и тестовой (справа) выборке. Каждый пример обозначен на графике точкой. Если точка попадает на выделенную линию (диагональ), то значит, сеть предсказала результат с достаточно высокой точностью. Если точка находится выше диагонали, зна-

чит, сеть недооценила, ниже – переоценила. Необходимо добиваться, чтобы точки располагались как можно ближе к диагонали.

6. Проверка результатов обучения. После того как сеть обучена, можно проверить результаты обучения (рис. 10.71).

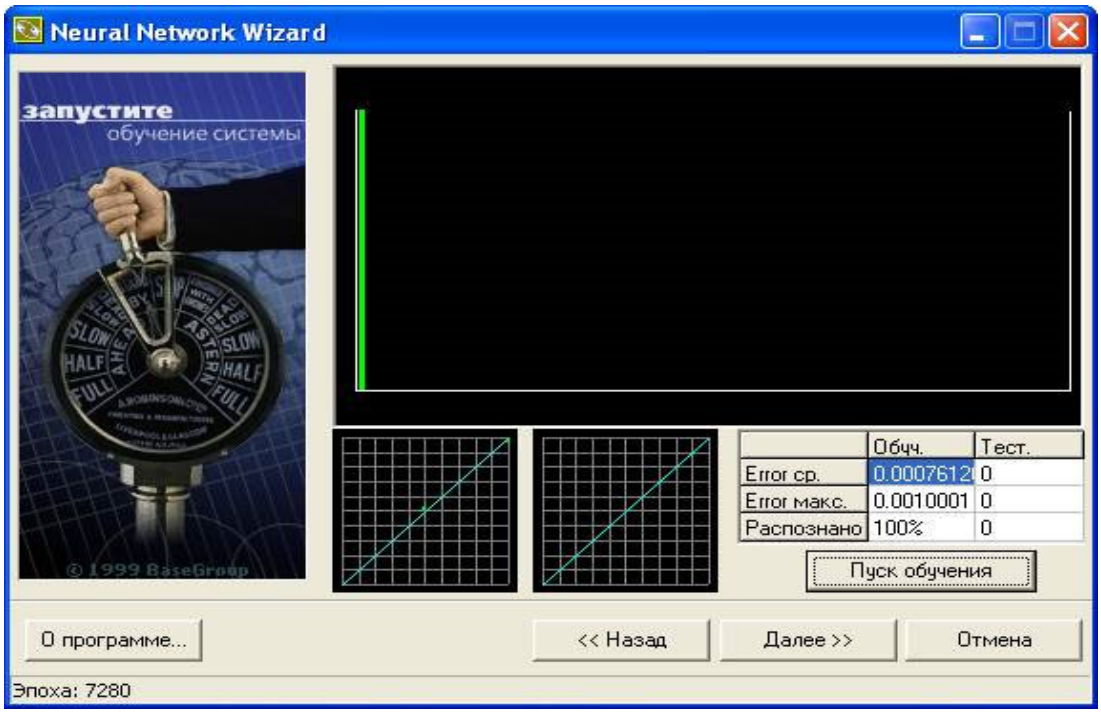

Рис. 10.70. Ход и результаты обучения сети

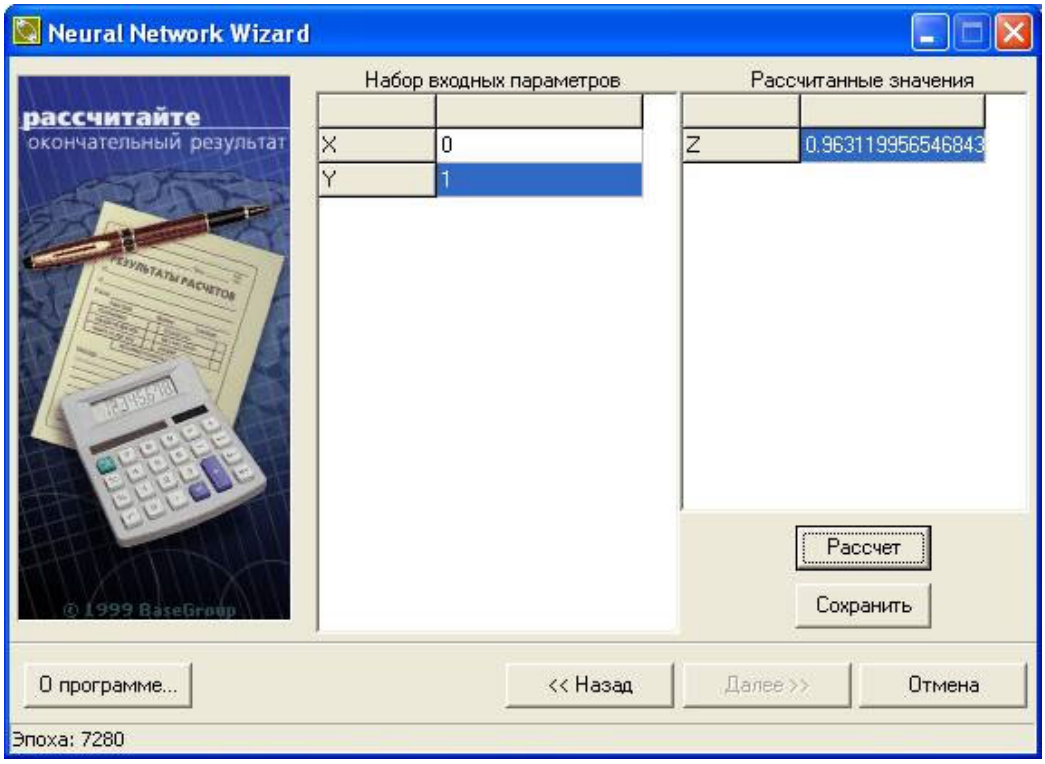

Рис. 10.71. Проверка обучения

Результаты обучения представлены в табл. 10.13.

Таблица 10.13

Результаты обучения

X	*Y*	Целевой выход *Z*	Выход сети	Ошибка
0	0	0	0,1	0,1
0	1	1	0,96	0,004
1	0	1	0,97	0,003
1	1	0	0	0

Контрольные вопросы

1. Как создать набор данных и подготовить обучающие примеры?
2. Как задавать входные и выходные переменные?
3. Как использовать различные методы нормализации?
4. Как устанавливать различные параметры конфигурации для нейронной сети?
5. Как задавать параметры обучения для нейронной сети?

10.7. Лабораторная работа № 7. Решение задач на планирование в среде CLIPS

Время выполнения – 4 часа.

Цель работы: ознакомиться с основными принципами и возможностями программирования на языке CLIPS для получения навыков работы в данной оболочке. Разработать программу для составления планов действий технической системой в заданной предметной области.

Задачи работы

1. Изучить интерфейс программы CLIPS.
2. Изучить основы программирования на языке CLIPS.

3. Выполнить в режиме отладки (включить просмотр правил и фактов) программу «Робот и ящик» (предварительно набрать исходный текст программы из листинга 1 в текстовом редакторе и сохранить под именем robot.clp, затем загрузить в CLIPS).

4. Модифицировать программу «Робот и ящик» таким образом, чтобы ящик необходимо было передвинуть в комнату *C*. Выполнить в режиме отладки.

5. Разработать программу на планирование действий технического объекта согласно варианту задания, выданного преподавателем.

Перечень обеспечивающих средств

Для выполнения работы необходимы:

– конспект лекций;

– персональный компьютер с установленными на нем приложениями Microsoft Office Word и CLIPS.

Общие теоретические сведения

Задача планировщика заключается в определении последовательности действий для модуля решения. Например, это могут быть системы управления. Для традиционного планирования, которое основано на знаниях, создание плана требует организации частей знаний и частичных планов в процедуру решения. Планирование будет использоваться в экспертных системах при рассуждении о событиях, которые происходят во времени. Планирование применяется в следующих областях: производство, управление, робототехника, а также в задачах для понимания естественного языка.

Планы будут создаваться на основе поиска в пространстве возможных действий. Это будет до тех пор, пока не установится последовательность, которая будет необходима для решения задачи.

Такое пространство показывает состояние мира, изменяющееся при выполнении каждого действия. Поиск решения закончится, когда будет достигнуто целевое состояние для описания мира [1].

Методика выполнения работы

Разработка программы для планирования действий робота «Робот и ящик»

1. Задача звучит следующим образом: пусть имеются 2 комнаты – *A* и *B*, в комнате *A* находится робот, в комнате *B* – ящик. Необходимо вытолкнуть ящик в комнату *A*.

2. Для решения данной задачи запускаем приложение *CLIPS*. Оно показано на рис. 10.72.

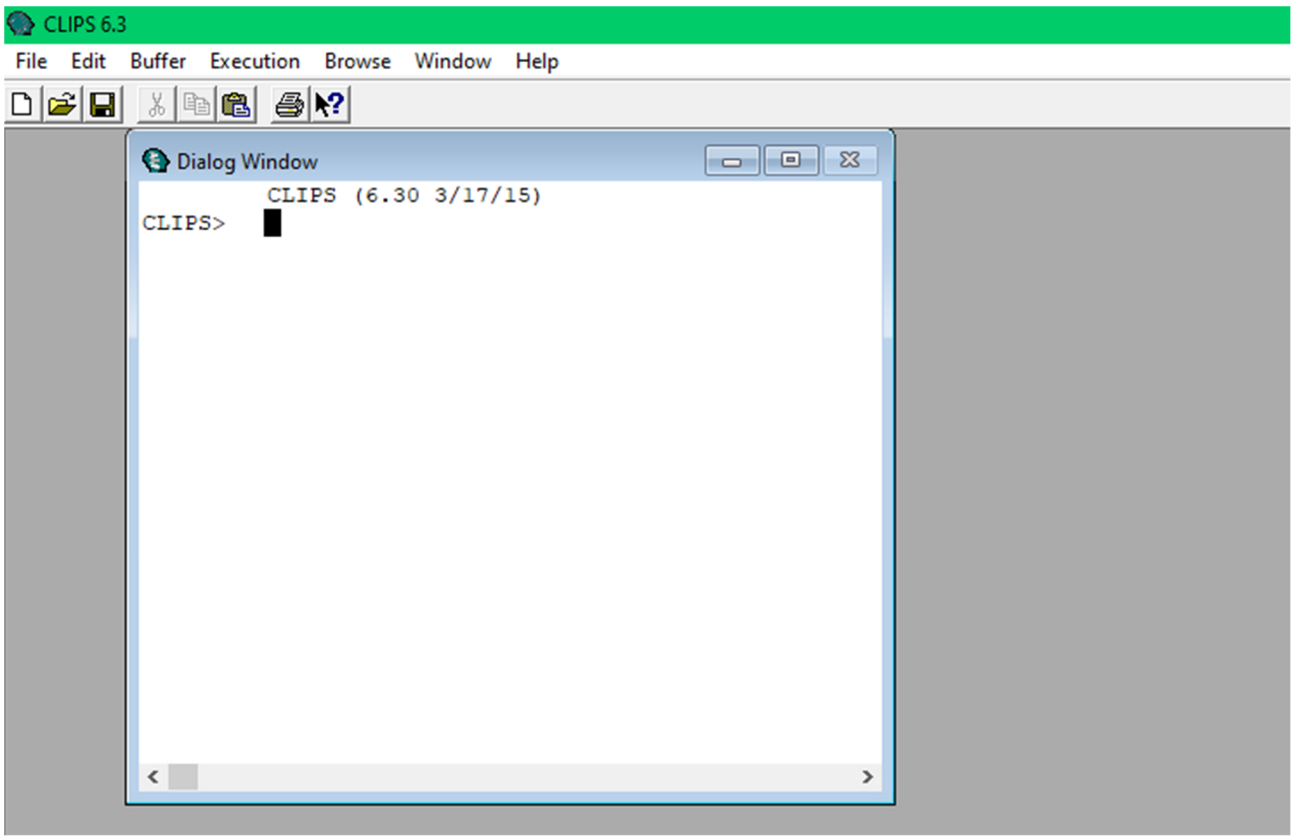

Рис. 10.72. Запуск приложения *CLIPS*

3. Работа ведется в режиме интерпретатора, где можно использовать множество команд. Решение данной задачи происходит с помощью шаблонных фактов.

Для этого введем шаблон, который определял бы местоположение предмета:

```
(deftemplate in
(slot object (type SYMBOL))
(slot location (type SYMBOL))
)
```

В данном случае слот *Object* используется для установки названия предмета или робота, а слот *location* – для задания места, где этот предмет или робот будет находиться.

4. Затем требуется задать роботу конкретную цель действий. Для этого необходимо создать шаблон *goal*, представленный на рис. 10.73.

```
(deftemplate goal
 (slot object (type SYMBOL))
(slot from (type SYMBOL))
 (slot to (type SYMBOL)))
```

Рис. 10.73. Шаблон *goal*

Слот *object* будет задавать название объекта, который требуется переместить, а слоты *from* и *to* необходимы для определения откуда и куда переместить предмет.

5. С помощью шаблонов *in* и *goal* создаются начальные факты. Они показаны на рис. 10.74.

```
(deffacts world
(in (object robot) (location RoomA))
(in (object box) (location RoomB))
(goal (action push) (object box) (from RoomB) (to RoomA))
)
```

Рис. 10.74. Начальные факты

Первый факт означает, что робот находится в комнате *A*. Второй факт, что ящик в комнате *B*, третий – перетащить ящик из комнаты *B* в *A* [6].

6. На заключительном этапе для создания программы является задание правил. В данном случае требуется создать три правила, которые осуществляли бы следующие действия робота:

– перемещение робота в комнату, где находится объект;

– перемещение робота с объектом в комнату, указанную в цели;

– остановка программы, если цель достигнута.

Реализуем первое действие. Оно показано на рис. 10.75.

```
(defrule move
(goal (object ?X) (from ?Y))
(in (object ?X) (location ?Y))
?robot-position <- (in (object robot) (location ~?Y))
=>
(modify ?robot-position (location ?Y))
)
```

Рис. 10.75. Перемещение робота в комнату, где находится объект

В этом правиле используются три предпосылки. В первой предпосылке применяется шаблон *goal*, который задает значения для переменных *?X* и *?Y*.

Во второй определяется наличие объекта *?X* в комнате *?Y*.

В третьей предпосылке будет проверяться соответствие местоположения робота не соответствует *?Y* и запоминаться ссылка на данный факт в переменной *?robot-position*. В случае если все предпосылки данного правила истинны, то с помощью оператора *modify* меняется значение слота *location* на значение переменной *?Y* факта *?robot-position*, т. е. робот перемещается в комнату, где находится объект, который необходимо переместить.

7. Аналогично будет реализовано правило для перемещения робота с ящиком в комнату, указанную в цели. Правило показано на рис. 10.76.

```
(defrule push
(goal (object ?X) (from ?Y) (to ?Z))
^object-position <- (in (object ?X) (location ?Y))
'.'robot-position <- (in (object robot) (location ?Y))
=>
(modify ?robot-position (location ?Z)) (modify ?object-position (location ?Z)) )
```

Рис. 10.76. Перемещение робота с объектом в комнату

В данном случае изменяются два факта, ссылки на которые задаются в переменных *?object-position* и *?robot-position*: значение слота *location* меняется на значение переменной *?Z*, обозначающей, куда необходимо переместить предмет роботом.

Остановка выполнения программы в *CLIPS* осуществляется с помощью команды (*halt*). Условием остановки является наличие факта, что предмет, указанный в цели (слот *object*), находится в комнате, указанной в слоте *to*. Остановка программы в случае достижения цели представлена на рис. 10.77.

```
(defrule stop
(goal (object ?X) (to ?Y))
(in (object ?X) (location ?Y))
=>
(halt) )
```

Рис. 10.77. Остановка программы по достижении цели

8. На рис. 10.78 представлен листинг программы «Робот и ящик».

```
(deftemplate goal
(slot action (type SYMBOL))
(slot object (type SYMBOL))
(slot from (type SYMBOL))
(slot to (type SYMBOL)) )

(deftemplate in
(slot object (type SYMBOL))
(slot location (type SYMBOL)) )

(deffacts world
(in (object robot) (location RoomA))
(in (object box) (location RoomB))

(goal (action push) (object box) (from RoomB) (to RoomA)) )

(defrule stop
(goal (object ?X) (to ?Y))
(in (object ?X) (location ?Y))
=>
(halt) )

(defrule move
(goal (object ?X) (from ?Y))
(in (object ?X) (location ?Y))
?robot-position <- (in (object robot) (location ?Z&~?Y))
=>
(modify ?robot-position (location ?Y)) )
  (defrule push
  (goal (object ?X) (from ?Y) (to ?Z))
  (in (object ?X) (location ?Y))
  ?object-position <- (in (object ?X) (location ?Y))
  ?robot-position <- (in (object robot) (location ?Y))
  =>
  (modify ?robot-position (location ?Z))
(modify ?object-position (location ?Z)) )
```

Рис. 10.78. Листинг программы «Робот и ящик»

9. Для запуска программы по планированию действий робота «Робот и ящик» необходимо скопировать текст программы в текстовый редактор (*Notepad*) и сохранить файл *Robot1* с расширением clp (рис. 10.79).

10. Загрузка данного файла в программу *CLIPS* осуществляется с помощью пункта меню *File\Load*.

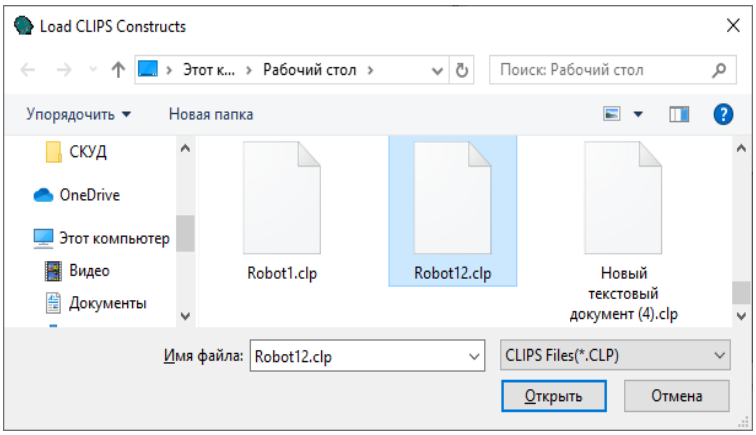

Рис. 10.79. Выбор загружаемого файла

11. Запускаем интерпретатор (рис. 10.80).

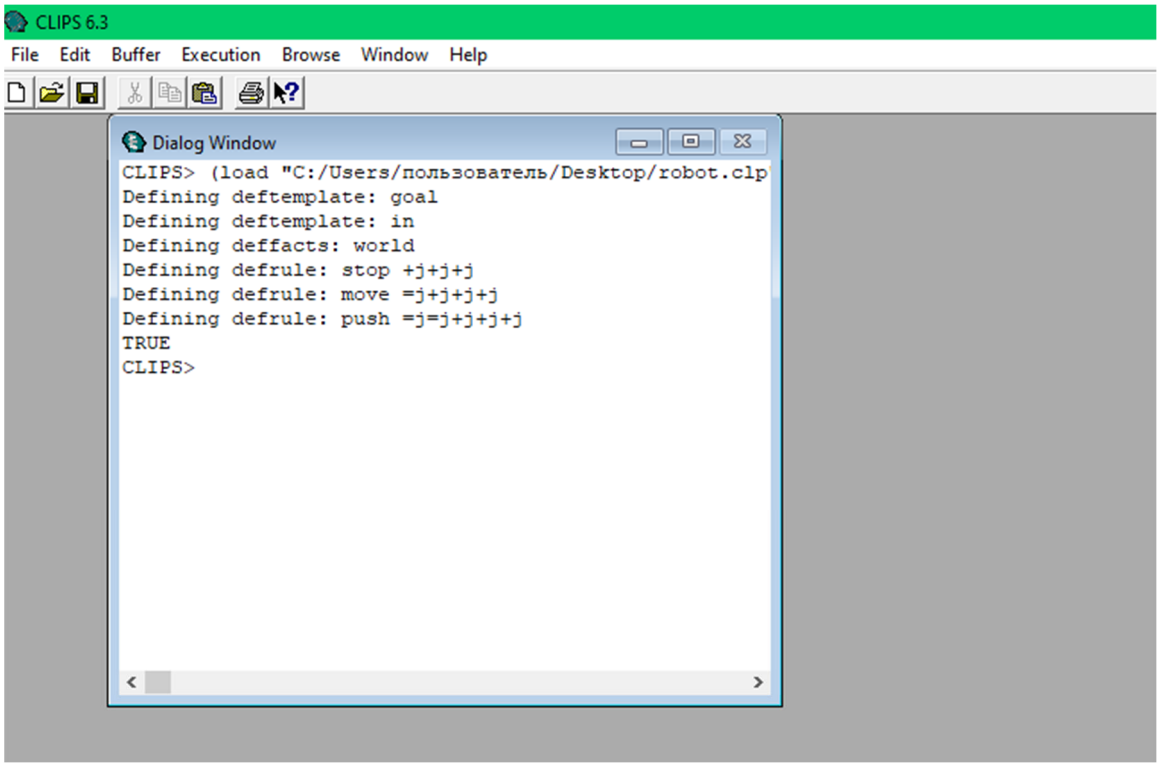

Рис. 10.80. Параметры загруженного файла

12. В меню *Execution/Watch* требуется установить флажок *Rules*, чтобы наблюдать результат трассировки процесса выполнения (рис. 10.81).

Рис. 10.81. Окно *Watch Options*

13. Вводим команду *Reset* в командной строке интерпреатора либо выбираем в меню команду *Execution/Reset* (рис. 10.82). Команда *Reset* обеспечит перезагрузку описания исходного состояния в рабочую память.

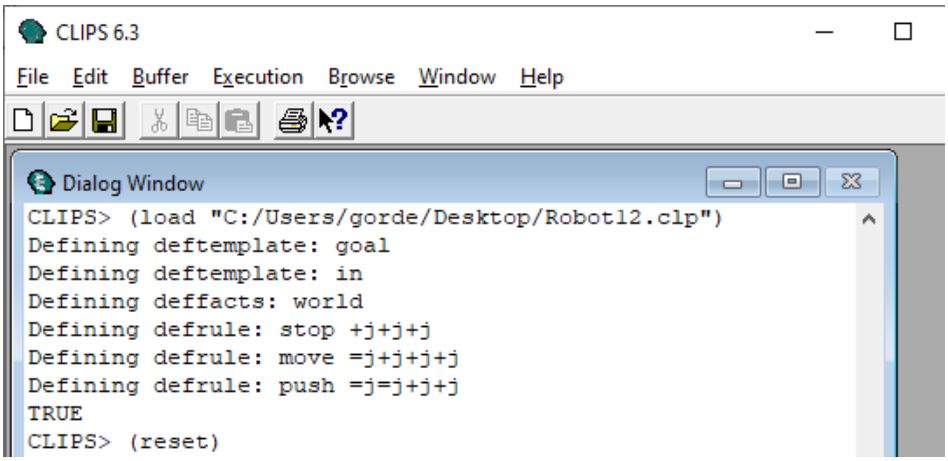

Рис. 10.82. Ввод команды *Reset*

14. Для запуска интерпретатора вводим в окно команду *Run*. Она запустит программу на выполнение (рис. 10.83).

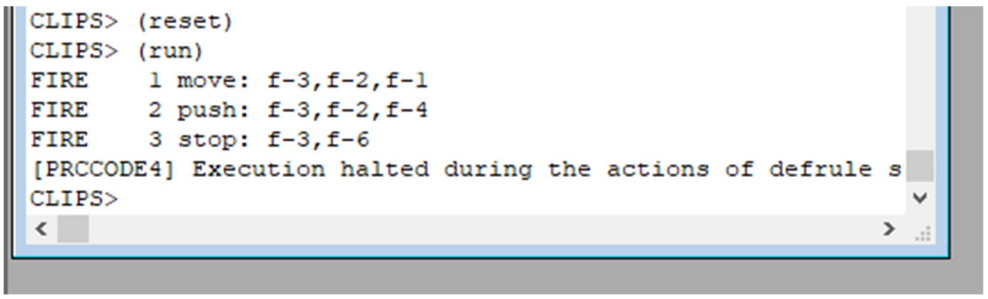

Рис. 10.83. Трассировка

В карте трассировки каждая строка представляет добавление вектора в рабочую память и удаление вектора из рабочей памяти. Строки, которые начинаются с *FIRE*, представляют активизацию какого-либо правила (рис. 10.84).

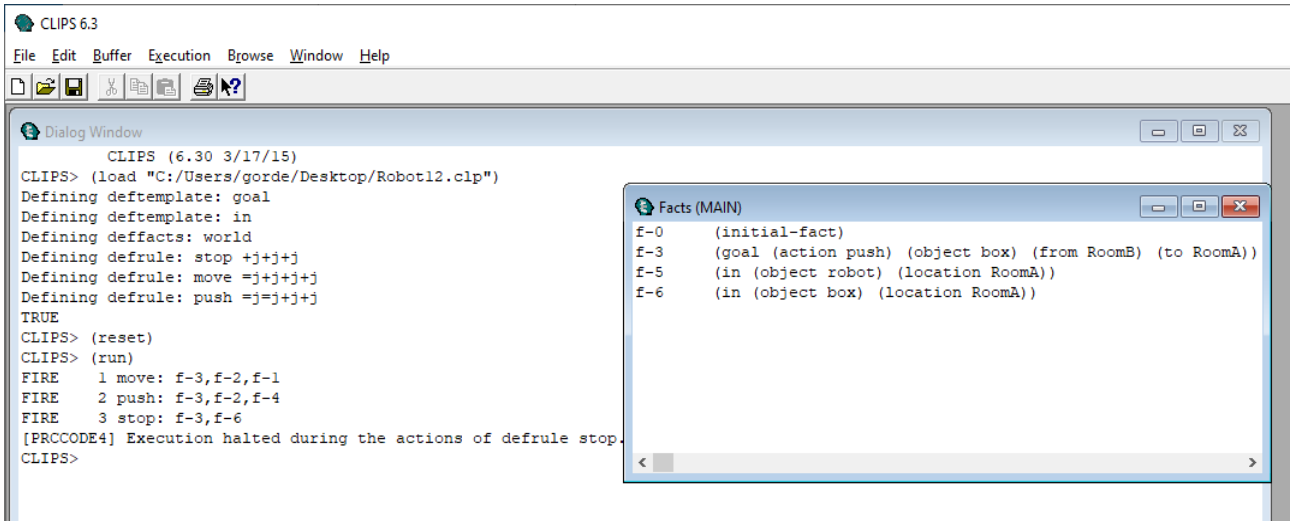

Рис. 10.84. Факты

Модификация программы «Робот и ящик»

1. Для того чтобы ящик был передвинут в комнату *C*, в листинге кода программы, во-первых, добавляем в факты наличие комнаты *C*, во-вторых, изменяем цель: вытолкнуть ящик в комнату *C*, а не *A* (рис. 10.85).

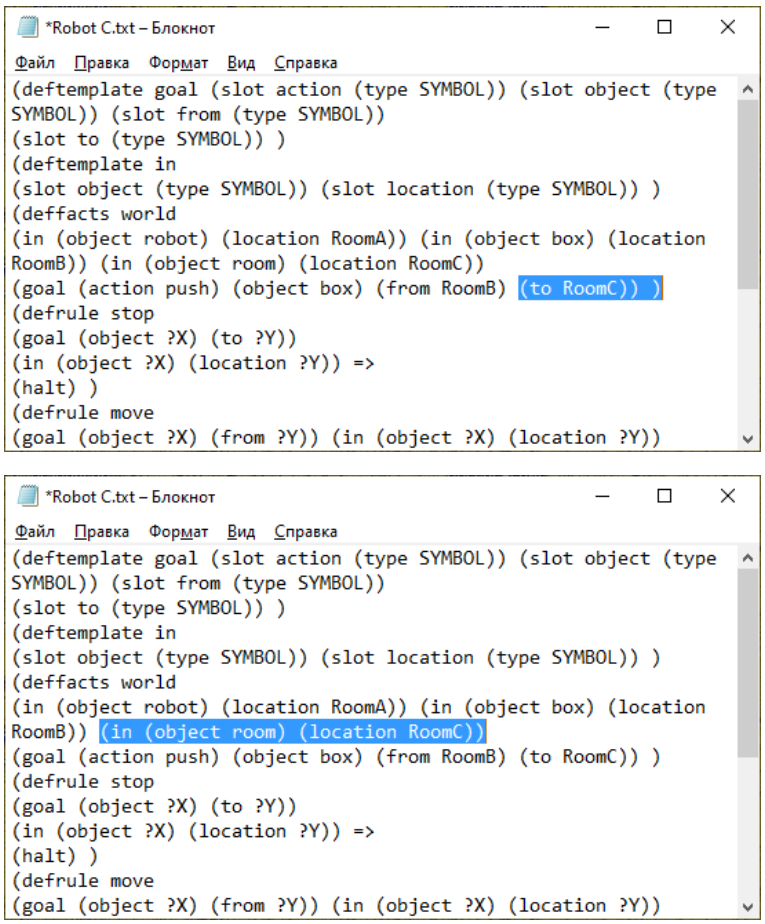

Рис. 10.85. Изменение кода программы

2. Открываем программу *CLIPS* и с помощью команды *Load* загружаем код программы (рис. 10.86).

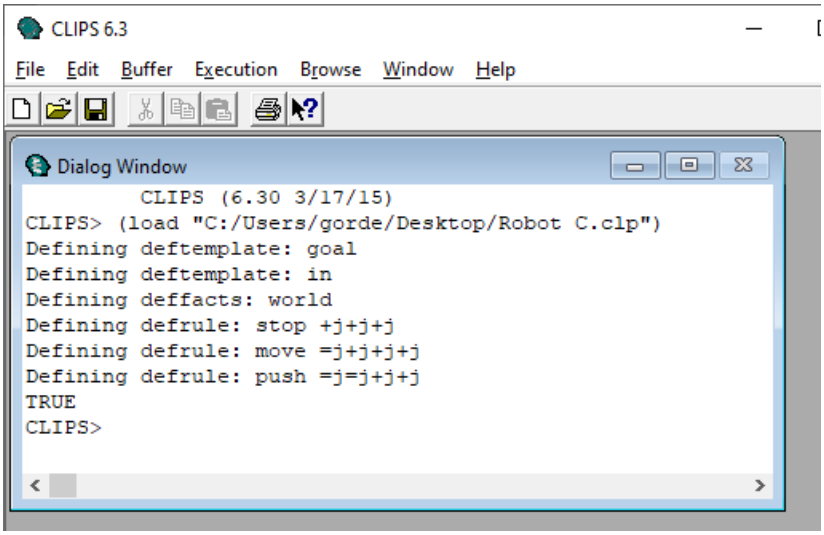

Рис. 10.86. Загрузка кода

3. Вводим команду *Reset* в командной строке интерпреатора. Она обеспечит перезагрузку описания исходного состояния в рабочую память. Для запуска интерпретатора вводим в окно команду *Run* (рис. 10.87, 10.88).

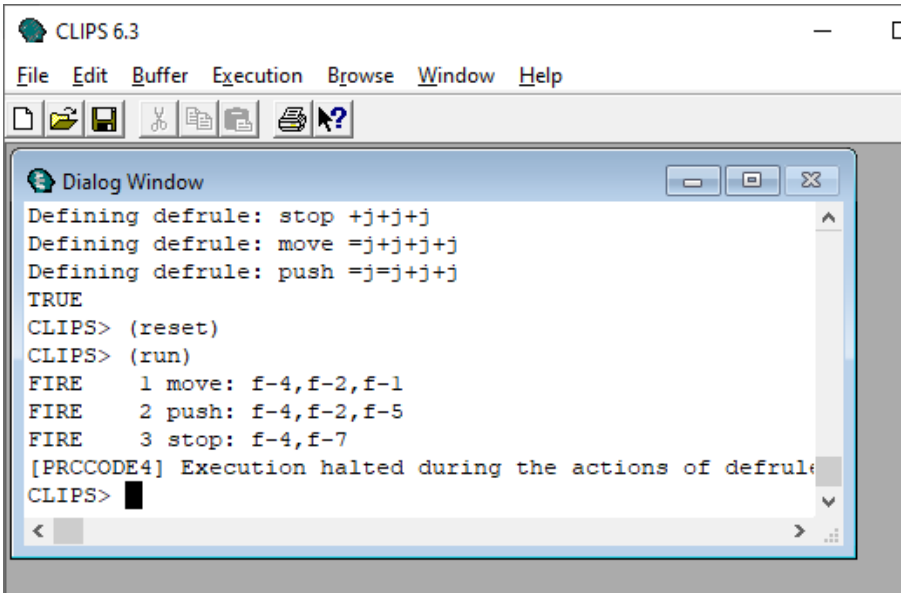

Рис. 10.87. Ввод команд *Reset* и *Run*

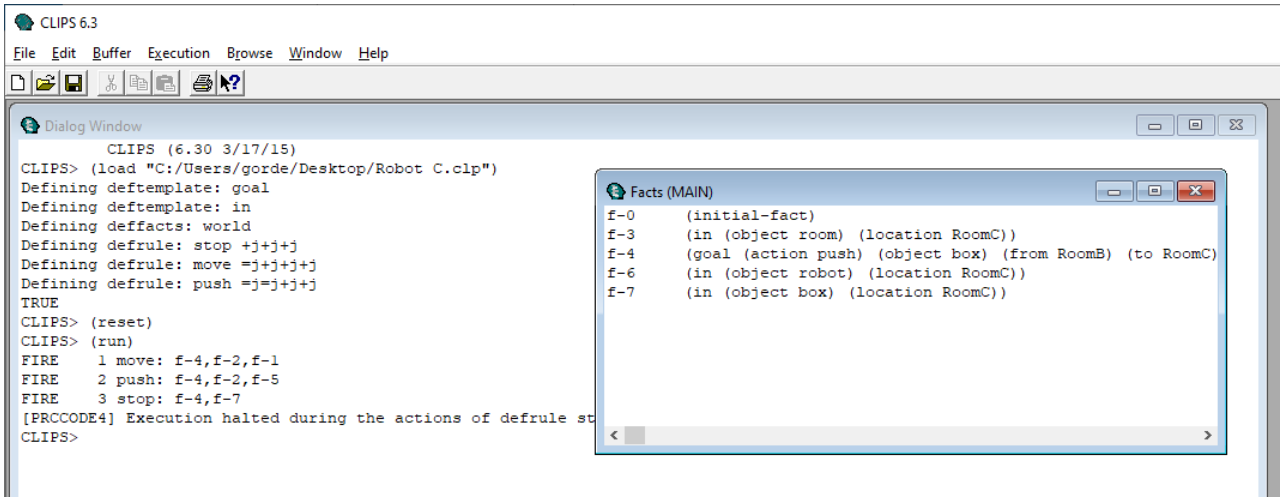

Рис. 10.88. Факты

4. Код модифицированной программы «Робот и ящик» показан на рис. 10.89.

```
(deftemplate goal
(slot action (type SYMBOL))
(slot object (type SYMBOL))
(slot from (type SYMBOL))
(slot to (type SYMBOL)) )

(deftemplate in
(slot object (type SYMBOL))
(slot location (type SYMBOL)) )

(deffacts world
(in (object robot) (location RoomA))
(in (object box) (location RoomB))
(in (object room) (location RoomC))
(goal (action push) (object box) (from RoomB) (to RoomC)))

(defrule stop
(goal (object ?X) (to ?Y))
(in (object ?X) (location ?Y))
=>
(halt) )

(defrule move
(goal (object ?X) (from ?Y))
(in (object ?X) (location ?Y))
?robot-position <- (in (object robot) (location ?Z&~?Y))
=>
(modify ?robot-position (location ?Y)) )

(defrule push
(goal (object ?X) (from ?Y) (to ?Z))
(in (object ?X) (location ?Y))
?object-position <- (in (object ?X) (location ?Y))
?robot-position <- (in (object robot) (location ?Y))
=>
(modify ?robot-position (location ?Z))
(modify ?object-position (location ?Z)) )
```

Рис. 10.89. Модифицированная программа «Робот и ящик»

Варианты заданий

В каждой задаче необходимо создать базу фактов для процесса, которые должны выполняться последовательно.

1. Авиаперевозки.

Требуется организовать перевозку грузов с помощью воздушного транспорта. Необходимо предусмотреть варианты для разгрузки и загрузки самолета, а также следующие параметры:

– взлет;

– перелет из одного аэропорта в другой;

– посадку;

– учет веса груза;

– грузоподъемность самолета.

2. Шиномонтажная мастерская.

Роботу необходимо проводить диагностику и замену колеса с пробитой покрышкой. Для каждой марки автомашины будет определен свой класс колес. Количество колес ограничено, при отсутствии каких-либо колес программа должна известить об этом.

3. Требуется спланировать действия для создания пирамиды из блоков (*A*, *B*, *C*…). Необходимо учитывать следующие параметры:

– блоки разного размера;

– блоки разной формы, которые устанавливаются по заранее определенным правилам. Они задаются в базе фактов.

4. Необходимо разработать программу перевозки грузов лифтом. Лифт может перемещаться между этажами, согласно вызовам перевозить грузы, учитывая их вес.

5. Автомат по продаже воды.

Автомат имеет следующие параметры:

– объем газированной воды ограничен;

– автомат имеет несколько видов сиропа;

– автомат выдает воду с сиропом;

– сироп выдается в соответствии с номиналом монеты и запросом (вишневым, лимонным и т. п.).

6. Музыкальный автомат.

В наличии имеется определенное количество пластинок. Автомат должен выполнять следующие требования:

– по определенному в базе фактов правилу устанавливать нужную пластинку;

– если пластинки нет в базе, необходимо сообщить об этом;

– необходимо предусмотреть счетчик времени;

– по окончании времени проигрывания пластинки она должна автоматически убираться.

7. Автомат разлива воды в бутылки.

Имеется определенное количество воды в бутылке заданной емкости. Требуется разлить воду в бутылки. Автомат должен удовлетворять следующим параметрам:

– вода закончилась;

– не хватает емкости;

– наполнять емкость, если воды меньше, чем объем заданной емкости.

8. Закачка файлов.

Присутствует список файлов, которые находятся в очереди для скачивания. Для каждого файла определен их размер. На каждом шаге программы будет скачано определенное количество байт. Необходимо учитывать требование: если количество байт, оставшееся для полного скачивания файла, меньше квоты, то остаток должен передаваться следующему файлу.

9. Гирлянда.

Гирлянда имеет лампочки различных цветов, формы и т. д. Требуется создать базу знаний, которая содержала бы правила:

– включение и выключение определенных лампочек в момент времени;

– под моментом времени считать один шаг программы.

10. Кофе-машина.

В кофе-машину загружаются зерна кофе. Машина должна определить порцию зерен, перемолоть их и сварить кофе.

Требуется учитывать возможность для ввода нескольких видов кофе, а также добавление сливок и сахара по запросу. Количество ингредиентов для приготовления кофе, такие как сахар, сливки, вода, должно быть огра-

ничено. В случае если какой-либо вид кофе получить невозможно, автомат должен выдать сообщение об этом.

11. Процесс сборки изделия.

Для сборки изделия имеются в наличии детали, которые участвуют в сборке. Каждая деталь в процессе сборки должна использоваться в определенной последовательности. Когда изделие будет собрано, автомат должен переходить к сборке нового изделия. Процесс прекращается только тогда, когда сборка невозможна (закончились или не хватает какой-либо детали).

12. Продажа билетов.

В продаже имеется определенное количество билетов на различные мероприятия. Необходимо разработать автомат, который по запросу и заданному количеству монет смог бы выдавать билеты. Автомат должен удовлетворять условиям: выдавать сдачу и оповещать об отсутствии билетов на запрашиваемое мероприятие.

13. Библиотека.

Необходимо разработать автомат для выдачи книг из заданной базы по запросу абонента. Книга может отсутствовать в библиотеке или быть на руках у другого абонента.

14. Комплексный обед.

Имеется меню с ценами. Необходимо разработать автомат, который по предъявленным запросам (перечень блюд, закусок и напитков и их количество) смог бы составить комплексные обеды и предъявить цену.

Контрольные вопросы

1. В чем заключается задача планировщика?
2. Как осуществляется поиск решения?
3. В чем назначение слотов *object*, *from* и *to*?
4. В чем назначение шаблонов *in* и *goal*?
5. Как загрузить файл в программу *CLIPS*?
6. Для чего нужно устанавливать флажок *Rules*?

10.8. Лабораторная работа № 8. Решение задач из логики высказываний на CLIPS

Время выполнения – 2 часа.

Цель работы: составить программу на языке CLIPS, которая позволит представить факты и правила в виде логических выражений.

Задачи работы

1. Изучить теоретический материал для решения логических задач на основе экспертных систем.
2. Реализовать примеры программ логического доказательства.
3. Реализовать вариант задания решения логической задачи.
4. Ответить на контрольные вопросы.

Перечень обеспечивающих средств

Для выполнения работы необходимо изучить общие теоретические сведения, последовательно выполнить инструкции задания. Допускается полученные результаты (отчет) представить в электронной форме в виде текстового файла.

Для выполнения работы необходимо использовать программу CLIPS.

Общие теоретические сведения

Для решения логических задач на основе экспертных систем необходимо применять аппарат алгебры высказываний. Он позволяет представлять факты и правила в виде логических выражений.

Высказывание – это повествовательное предложение, о котором можно сказать, истинно оно или ложно в данном месте и в данное время. Логические значения высказываний – 1 («истина») или 0 («ложь»).

Логические операции над высказываниями: отрицание (унарная операция); конъюнкция, дизъюнкция, импликация, эквивалентность (бинарные операции).

Отрицанием высказывания x называется высказывание $\bar{x}$, которое истинно, если x ложно, и ложно, если x истинно. Читается «не x» или «неверно, что x».

Конъюнкцией высказываний *x* и *y* называется высказывание *x* & *y*, которое истинно, если *x* и *y* истинны, и ложно, если хотя бы одно из них ложно. Читается «*x* и *y*».

Дизъюнкцией высказываний *x* и *y* называется высказывание *x* v *y*, которое истинно, если хотя бы одно из высказываний *x* или *y* истинно, и ложно, если оба они ложны. Читается «*x* или *y*».

Импликацией высказываний *x* и *y* называется высказывание $x \rightarrow y$, которое ложно, если *x* истинно, а *y* ложно, и истинно во всех остальных случаях. Читается «из *x* следует *y*» или «если *x*, то *y*».

Эквивалентностью высказываний *x* и *y* называется высказывание $x \leftrightarrow y$, которое истинно, если оба высказывания *x* и *y* одновременно истинны или ложны, и ложно во всех остальных случаях. Читается «для того чтобы *x*, необходимо и достаточно, чтобы *y*» или «*x* тогда и только тогда, когда *y*».

Значения логической операции можно описать с помощью таблицы, связывающей значения операндов и операции. Такая табл. 10.14 называется таблицей истинности [3].

Таблица 10.14

Таблица истинности для логических операций

X	*У*	*X*	*X & У*	*X* V *У*	$X \rightarrow У$	$X \leftrightarrow У$
1	1	*0*	1	1	1	1
1	0	*0*	0	1	0	0
0	1	1	0	1	1	0
0	0	1	0	0	1	1

Высказывания подразделяются на элементарные и составные.

Составное высказывание, которое может быть получено из элементарных высказываний путем применения логических операций, называется *формулой алгебры логики*.

Порядок выполнения бинарных логических операций: сначала – конъюнкция, затем – дизъюнкция и в последнюю очередь – импликация и экви-

валентность. Логические значения формулы алгебры логики могут быть описаны с помощью таблицы истинности.

Формула, истинная при всех значениях входящих в нее переменных, называется *тождественно истинной*, или *тавтологией*.

Формула, ложная при всех значениях входящих в нее переменных, называется *тождественно ложной*, или *противоречием*.

Формула, истинная хотя бы на одном наборе значений входящих в нее переменных и не являющаяся тождественно истинной, называется *выполнимой*, или *опровержимой*.

Две формулы алгебры логики называются *равносильными*, если они принимают одинаковые логические значения на любом наборе значений входящих в них элементарных высказываний.

Равносильность формул $L1$ и $L2$ обозначается как $L1 \equiv L2$ [9].

Основные равносильности:

$$\begin{aligned}
&x \ \& \ 0 \equiv 0;\\
&x \& 1 \equiv x;\\
&x \& x \equiv x;\\
&x \& \ (y \vee x) \equiv x;\\
&x \vee y \& x \equiv x;\\
&x \vee 0 \equiv x;\\
&x \vee 1 \equiv 1;\\
&x \vee x \equiv x.
\end{aligned} \tag{10.1}$$

Равносильности, выражающие одни логические операции через другие:

$$\begin{aligned}
&x \& y \equiv \overline{x} \vee \overline{y};\\
&x \& y \equiv x \vee y;\\
&x \to y \equiv \overline{x} \vee y;\\
&x \vee y \equiv \overline{x} \ \& \ \overline{y};\\
&x \vee y \equiv x \& y;\\
&x \leftrightarrow y \equiv \ (x \to y) \ \ \& \ \ (y \to x).
\end{aligned} \tag{10.2}$$

Равносильности, выражающие основные законы алгебры логики:

$$
\begin{aligned}
&x \,\&\, y \equiv y \,\&\, x;\\
&x \,\&\, \left(y \,\&\, z\right) \equiv \left(x \,\&\, y\right) \,\&\, z;\\
&x \,\&\, \left(y \vee z\right) \equiv x \,\&\, y \vee x \,\&\, z;\\
&x \vee y \equiv y \vee x;\\
&x \vee \left(y \vee z\right) \equiv \left(x \vee y\right) \vee z;\\
&x \vee y \,\&\, z \equiv \left(x \vee y\right) \,\&\, \left(x \vee z\right).
\end{aligned}
\qquad (10.3)
$$

Методика выполнения работы

Примеры программ логического доказательства.

1. Алеша, Боря и Гриша нашли в земле старинный сосуд. Рассматривая удивительную находку, каждый высказал по два предположения:

– Алеша: это сосуд греческий и изготовлен в V в.;

– Боря: это сосуд финикийский и изготовлен в III в.;

– Гриша: это сосуд не греческий и изготовлен в IV в.

Учитель истории сказал ребятам, что каждый из них прав только в одном из двух предположений. Где и в каком веке изготовлен сосуд?

Эти высказывания можно свести к следующему: сосуд финикийский и изготовлен в V в. или сосуд греческий и изготовлен в III или IV в.

Теперь создаем правила на основе высказываний, которые в будущем преобразуем в программный код:

– если сосуд изготовлен не в V в., то он греческий;

– если сосуд изготовлен в V в., то он финикийский;

– сосуд не может быть одновременно греческим и финикийским;

– сосуд не может быть изготовлен одновременно в III и IV в.;

– сосуд не может быть изготовлен одновременно в IV и V в.;

– сосуд не может быть изготовлен одновременно в III и V в.

Начинаем писать программу, используя язык программирования CLIPS.

```
(deftemplate stud
(slot name (type SYMBOL))
(slot where (type SYMBOL)(default Unk))
```

```
(slot when (type SYMBOL)(default Unk))
(slot truth (type SYMBOL))
)

(deffacts situation
(stud (name Alesha)(when V)(where Greek)(truth No))
(stud (name Borya)(when III)(where Phinic)(truth No))
(stud (name Grisha)(when IV)(where Phinic)(truth No))
)

(defrule stop
(declare (salience -1))
(stud (when ?X)(truth Yes))
(stud (when ?Y)(truth No))
(stud (when ?Z)(truth No))
=>
(halt)
)

(defrule is_greek
?X <- (stud (when ~V)(where Phinic)) =>
(modify ?X (where Greek)(truth Yes))
)

(defrule is_phinic
?X <- (stud (when V)(where Greek)) =>
(modify ?X (where Phinic)(truth Yes))
)

(defrule not_greek_and_phinic
?X <- (stud (when ?Z)(where Greek)(truth Yes))
?Y <- (stud (when ?Z)(where Phinic)(truth Yes)) =>
(modify ?X (truth No)) (modify ?Y (truth No))
)
```

```
(defrule not_3_and_4
?X <- (stud (when III)(where ?Z)(truth Yes))
?Y <- (stud (when IV)(where ?Z)(truth Yes)) =>
(modify ?X (truth No)) (modify ?Y (truth No))
)

(defrule not_3_and_5
?X <- (stud (when III)(where ?Z)(truth Yes))
?Y <- (stud (when V)(where ?Z)(truth Yes)) =>
(modify ?X (truth No)) (modify ?Y (truth No))
)

(defrule not_4_and_5
?X <- (stud (when IV)(where ?Z)(truth Yes))
?Y <- (stud (when V)(where ?Z)(truth Yes)) =>
(modify ?X (truth No)) (modify ?Y (truth No))
)
```

Вывод результатов работы программы в консоли:

```
CLIPS> (reset)
<== f-0   (initial-fact)
==> f-0   (initial-fact)
==> f-1   (stud (name Alesha) (where Greek) (when V) (truth No))
==> f-2   (stud (name Borya) (where Phinic) (when III) (truth No))
==> f-3   (stud (name Grisha) (where Phinic) (when IV) (truth No))
CLIPS> (run)
FIRE  1 is_greek: f-3
<== f-3   (stud (name Grisha) (where Phinic) (when IV) (truth No))
==> f-4   (stud (name Grisha) (where Greek) (when IV) (truth Yes))
FIRE  2 is_greek: f-2
<== f-2   (stud (name Borya) (where Phinic) (when III) (truth No))
==> f-5   (stud (name Borya) (where Greek) (when III) (truth Yes))
FIRE  3 not_3_and_4: f-5,f-4
<== f-5   (stud (name Borya) (where Greek) (when III) (truth Yes))
==> f-6   (stud (name Borya) (where Greek) (when III) (truth No))
```

<== f-4 (stud (name Grisha) (where Greek) (when IV) (truth Yes))
==> f-7 (stud (name Grisha) (where Greek) (when IV) (truth No))
FIRE 4 is_phinic: f-1
<== f-1 (stud (name Alesha) (where Greek) (when V) (truth No))
==> f-8 (stud (name Alesha) (where Phinic) (when V) (truth Yes))
FIRE 5 stop: f-8,f-7,f-7
[PRCCODE4] Execution halted during the actions of defrule stop.
CLIPS>

Выводим результат (рис. 10.90).

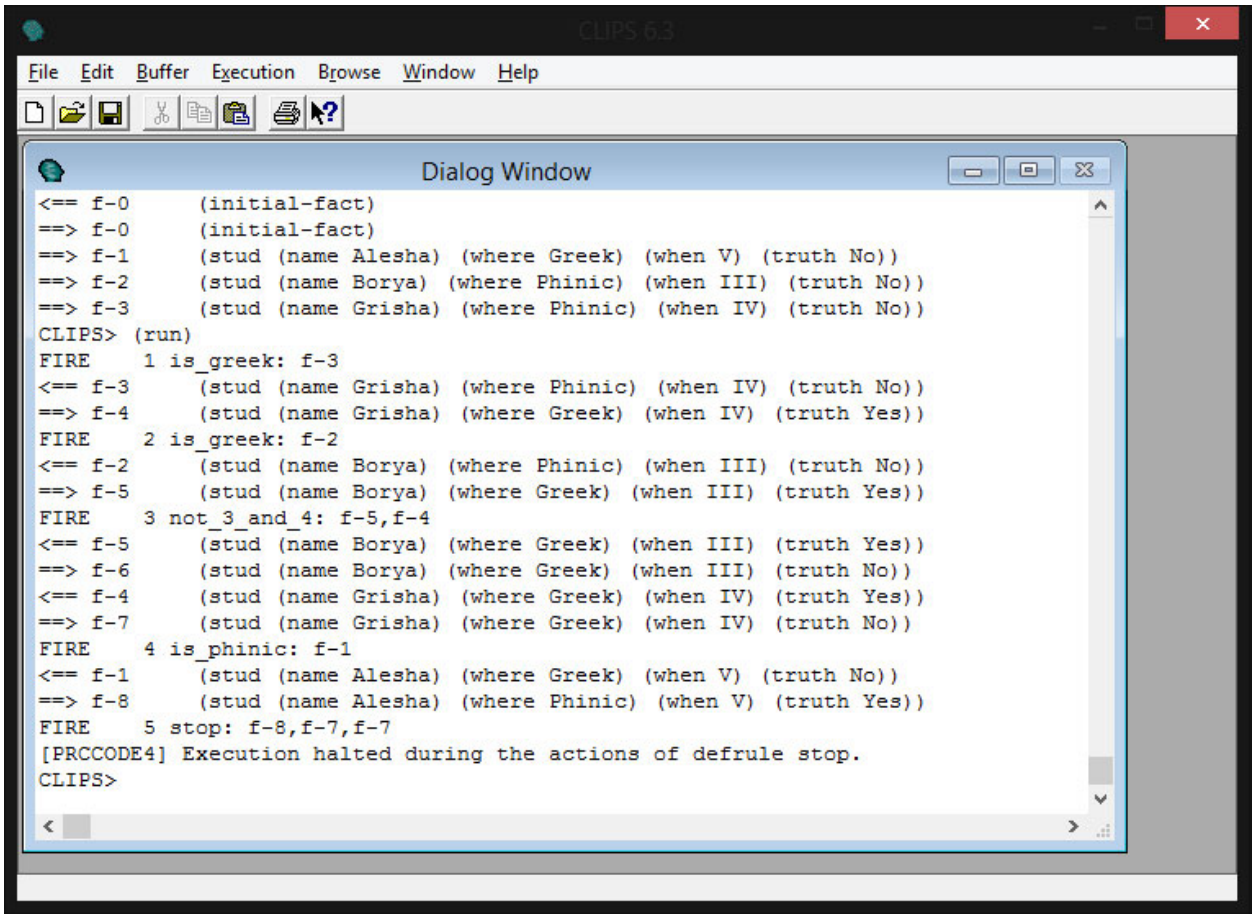

Рис. 10.90. Результат работы программы Logical.clp

2. Четыре подруги Аня, Маша, Настя, Вика пришли в магазин. Продавец сказал, что осталось только четыре платья: красное, розовое, оранжевое, синее. Отсюда можно утверждать, что:

а) красное платье купила Аня, а розовое – Маша;

b) Аня взяла розовое платье, а Вика купила оранжевое;

c) Настя забрала розовое, а Вика – синее платье.

Кто купил синее платье и какое платье выбрала Вика, если известно, что половина каждого утверждения истинна, а половина – ложь.

Исходя из условия, что каждое высказывание содержит и истину, и ложь, для каждого из них можно составить по два правила:

A: – если Аня купила красное платье, то Маша – не розовое;

– если Аня купила не красное платье, то Маша – розовое;

B: – если Аня купила розовое платье, то Вика – не оранжевое;

– если Аня купила не розовое платье, то Вика – оранжевое;

C: – если Настя купила розовое платье, то Вика – не синее;

– если Настя купила не розовое платье, то Вика – синее.

Листинг программы:

```
(deftemplate girl
(slot name1 (type SYMBOL)(default Unk))
(slot name2 (type SYMBOL)(default Unk))
(slot color1 (type SYMBOL) (default Unk))
(slot color2 (type SYMBOL) (default Unk))
(slot truthful (type SYMBOL)) )

(deffacts situation
(girl (name1 Anya)(color1 Red)(name2 Masha)(color2 Pink)(truthful No))
(girl (name1 Anya)(color1 Pink)(name2 Vika)(color2 Orange)(truthful No))
(girl (name1 Nastya)(color1 Pink)(name2 Vika)(color2 Blue)(truthful No))
)

(defrule stop
(declare (salience -1))
(girl (name1 ?Anya)(truthful Yes))
(girl (name2 ?Masha)(truthful Yes))
(girl (name1 ?Nastya)(truthful Yes))
(girl (name2 ?Vika)(truthful Yes))
=>
(halt) )
```

```
(defrule Anya
(girl (name2 Masha)(color2 Pink)(truthful No))
?Anya <- (girl (name1 Anya) (color1 Pink))=>
(modify ?Anya (color1 Red)(truthful Yes)) )

(defrule Vika
(girl (name1 Anya)(color1 Pink)(truthful No))
?Vika <- (girl (name2 Vika) (color2 Blue))=>
(modify ?Vika (color2 Orange)(truthful Yes)) )

(defrule Nastya
(girl (name2 Vika)(color2 Blue)(truthful No))
?Nastya <- (girl (name1 Nastya) (color1 Blue))=>
(modify ?Nastya (color1 Pink)(truthful Yes)) )

(defrule Masha
(girl (name1 Anya)(color1 Pink)(truthful No))
?Masha <- (girl (name2 Masha) (color2 Pink))=>
(modify ?Masha (color2 Blue)(truthful Yes)) )
```

Результат работы программы представлен на рис. 10.91.

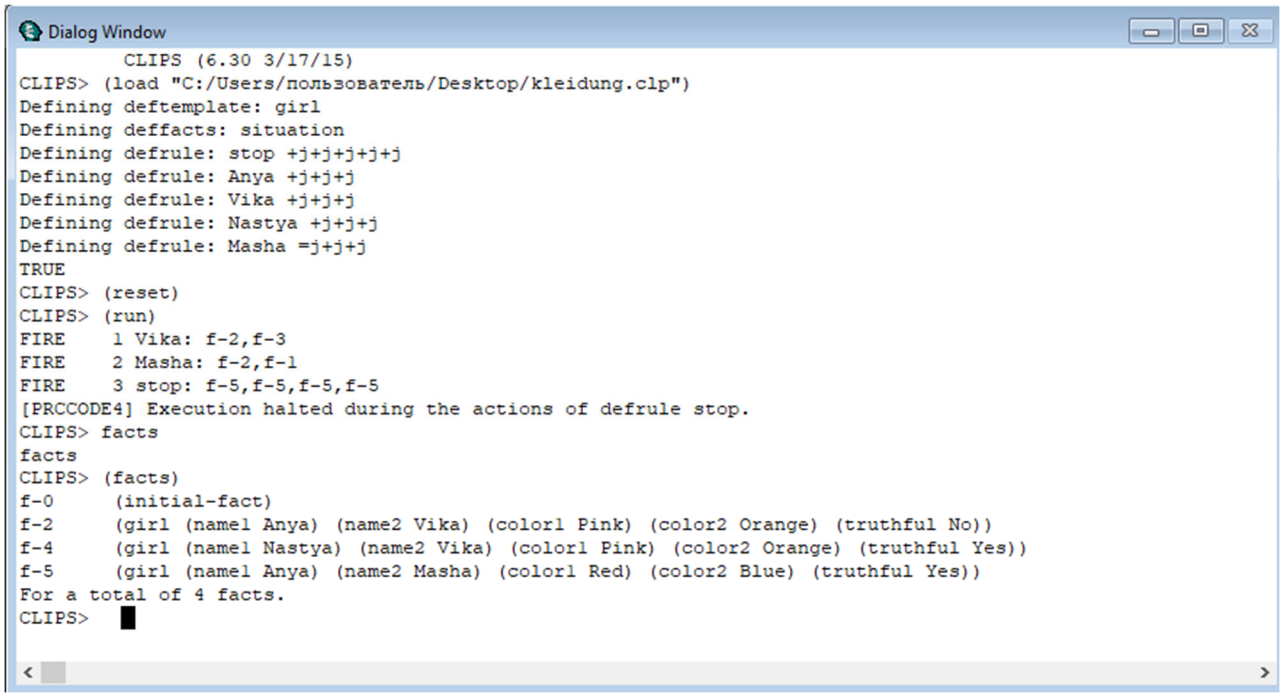

Рис. 10.91. Результат работы программы

3. Определите, кто из подозреваемых участвовал в преступлении. Известно, что:

a) если Иванов не участвовал или Петров участвовал, то Сидоров участвовал;

b) если Иванов не участвовал, то Сидоров не участвовал.

В задаче для программы с тремя переменными возможны восемь условий:

– GuiltyP – вина Петрова;

– GuiltyS – вина Сидорова;

– GuiltyI – вина Иванова.

(gangsters (GuiltyI 0) (GuiltyP 0) (GuiltyS 0))
(gangsters (GuiltyI 0) (GuiltyP 0) (GuiltyS 1))
(gangsters (GuiltyI 0) (GuiltyP 1) (GuiltyS 0))
(gangsters (GuiltyI 0) (GuiltyP 1) (GuiltyS 1))
(gangsters (GuiltyI 1) (GuiltyP 0) (GuiltyS 0))
(gangsters (GuiltyI 1) (GuiltyP 0) (GuiltyS 1))
(gangsters (GuiltyI 1) (GuiltyP 1) (GuiltyS 0))
(gangsters (GuiltyI 1) (GuiltyP 1) (GuiltyS 1))

По условиям задачи составим основные формулы:

– если Иванов не участвовал или Петров участвовал, то Сидоров участвовал:

(~Ivanov V Petrov => Sidorov);

– если Иванов не участвовал, то Сидоров не участвовал:

(~Ivanov => ~Sidorov).

Результат работы программы представлен на рис. 10.92.

Для проверки корректности работы программы составим таблицу истинности и вычислим значения при помощи алгебры логики (табл. 10.15).

Если сравнить данные работы программы с данными из таблицы, мы увидим, что исходя из заданных условий, программа произвела верный вывод.

При виновности Иванова по заданным условиям формула выдаст 1.

При виновности Иванова и Сидорова программа также выдаст 1.

При виновности всех участников формула также даст результат 1.

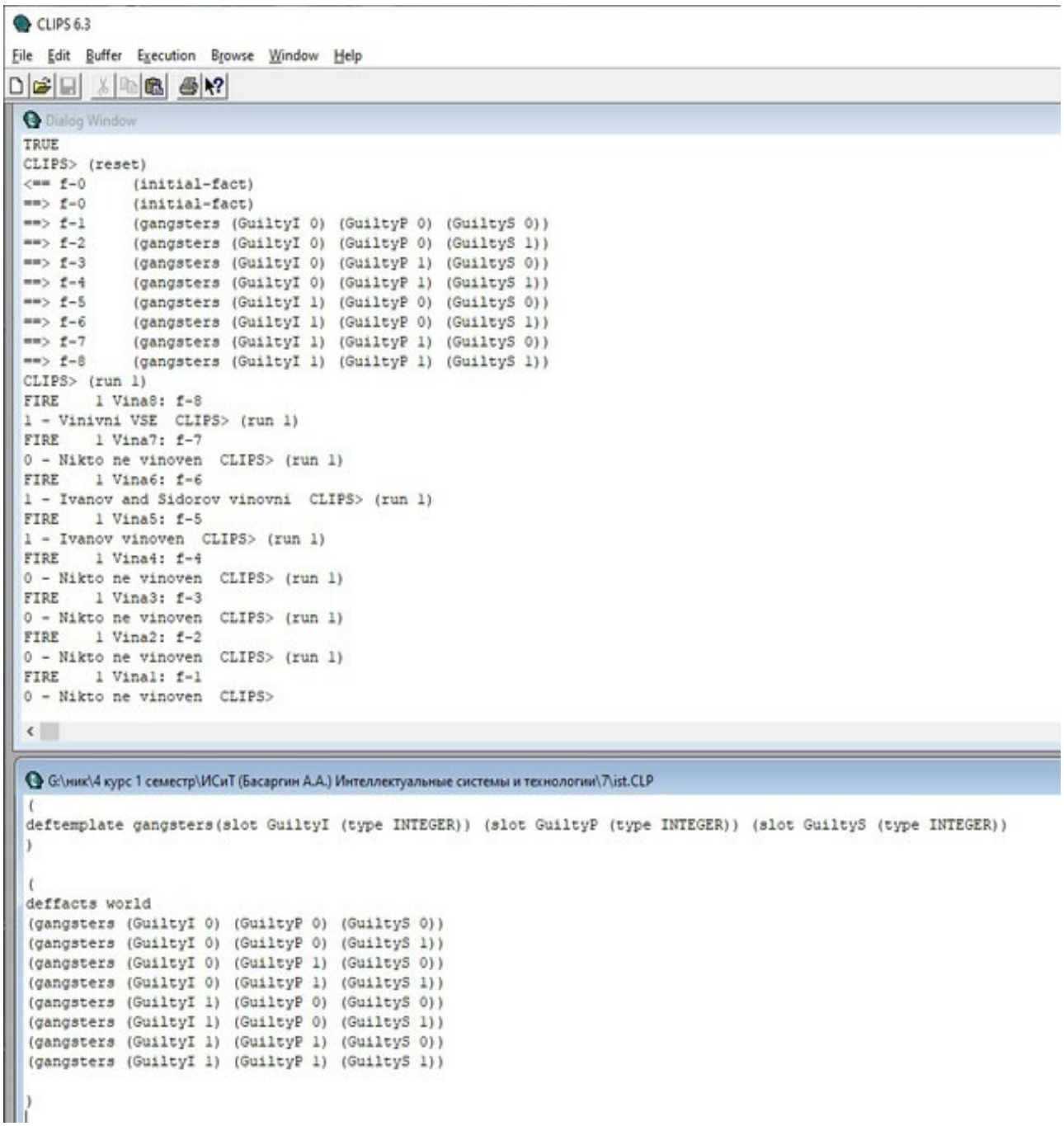

Рис. 10.92. Результат работы программы

Это означает, что возможны три исхода.

Нам требуется найти всего одного виновного, значит, по результатам выполнения условий в программе и составленной таблицы истинности виновен Иванов.

Таблица 10.15

Результат работы программы

Ivanov	Petrov	Sidorov	~Ivanov V Petrov	~Ivanov V Petrov => Sidorov	~Ivanov=> ~Sidorov	(~Ivanov V Petrov => Sidorov) and (~Ivanov=> ~Sidorov)
0	0	0	1	0	1	0
0	0	1	1	1	0	0
0	1	0	1	0	1	0
0	1	1	1	1	0	0
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	0	1	0
1	1	1	1	1	1	1

Варианты заданий

1. В школе было разбито стекло. В содеянном были заподозрены четыре ученика: Миша, Леня, Толя и Дима. Вот что они сказали в кабинете директора.

Леня: «Я не виноват. Я даже не подходил к окну. Миша знает, кто это сделал».

Дима: «Стекло разбил не я. С Мишей я не был знаком до поступления в школу. Это сделал Толя».

Толя: «Я не виновен. Это сделал Миша. Дима говорит неправду, утверждая, что я разбил стекло».

Миша: «Я не виноват. Стекло разбил Леня. Дима может поручиться за меня, так как знает меня очень давно».

При дальнейших расспросах ребята сознались, что две фразы из сказанных ими истинны, а одна, соответственно, ложна. Требуется выяснить, кто разбил стекло.

2. Аня, Вика и Сергей решили пойти в кино. Учитель, хорошо знавший ребят, высказал предположения:

– Аня пойдет в кино только тогда, когда пойдут Вика и Сергей;

– Аня и Сергей пойдут в кино вместе или же оба останутся дома;

– чтобы Сергей пошел в кино, необходимо, чтобы пошла Вика.

Когда ребята пошли в кино, оказалось, что учитель немного ошибся: из трех его утверждений истинными оказались только два. Кто из ребят пошел в кино?

3. Виктор, Роман, Леонид и Сергей заняли на олимпиаде по физике четыре первых места. Когда их спросили о распределении мест, они дали три ответа:

– Сергей первый, Роман второй;

– Сергей второй, Виктор третий;

– Леонид второй, Виктор четвертый.

Известно, что в каждом ответе только одно утверждение истинно. Как распределились места?

4. В нарушении правил обмена валюты подозреваются четыре работника банка – *A*, *B*, *C*, *D*. Известно, что:

– если *A* нарушил, то и *B* нарушил правила обмена валюты;

– если *B* нарушил, то и *C* нарушил или *A* не нарушал;

– если *D* не нарушил, то *A* нарушил, а *C* не нарушал;

– если *D* нарушил, то и *A* нарушил.

Кто из подозреваемых нарушил правила обмена валюты?

5. Богданову, Демидову и Смирнову предъявлено обвинение в ограблении ювелирного магазина. С места преступления они скрылись на автомобиле. На следствии Богданов сказал, что они уехали на синем «Москвиче», Демидов сказал, что это была черная «Волга», Смирнов утверждал, что это были «Жигули», но не синие. Известно, что каждый из них правильно назвал либо марку автомобиля, либо цвет. Определите марку автомобиля и его цвет.

6. В олимпиаде по биологии участвовало пять девушек: Алла, Нина, Вика, Рита, Соня. Об итогах олимпиады имеется пять высказываний:

– первое место заняла Алла, а Рита оказалась третьей;

– пятой была Вика, а Нина поднялась на первое место;

– первое место заняла Соня, а Вика была второй;

– Рита на последнем, пятом, месте, а Нина была предпоследней;

– Нина была четвертой, а первой – Алла.

Кто занял первое место и на каком месте была Алла, если известно, что в каждом высказывании одно утверждение правильное, а другое – нет.

7. Перед началом «Турнира четырех» болельщики высказали следующие предположения по поводу своих кумиров:

a) Макс победит, Билл – второй;

b) Билл третий, Ник первый;

c) Макс последний, а первый Джон.

Когда соревнования закончились, оказалось, что каждый из болельщиков был прав только в одном из своих прогнозов.

8. Классный руководитель пожаловался директору, что у него в классе появилась компания из трех учеников, один из которых всегда говорит правду, другой всегда лжет, а третий говорит через раз то ложь, то правду. Директор знает, что их зовут Коля, Саша и Миша, но не знает, кто из них правдив, а кто нет. Однажды все трое прогуляли урок астрономии. Директор знает, что никогда раньше никто из них не прогуливал астрономию. Он вызвал всех троих в кабинет и поговорил с мальчиками. Коля: «Я всегда прогуливаю астрономию. Не верьте тому, что скажет Саша». Саша: «Это был мой первый прогул этого предмета». Миша: «Все, что говорит Коля, – правда». Определите, кто из них всегда правдив, кто лжив всегда и кто – через раз.

9. Однажды Алиса повстречала Льва и Единорога, отдыхавших под деревом. Странные это были существа. Лев лгал по понедельникам, вторникам и средам и говорил правду во все остальные дни недели, Единорог же вел себя иначе: он лгал по четвергам, пятницам и субботам говорил правду во все остальные дни недели. Они высказали следующие утверждения:

Лев: «Вчера был один из дней, когда я лгу».

Единорог: «Вчера был один из дней, когда я тоже лгу».

Из этих двух высказываний Алиса сумела вывести, какой день недели был вчера. Что это был за день?

10. Покупатель в каждом из магазинов *A*, *B*, *C* и *D* сделал по одной покупке и приобрел джойстик, дискеты, бумагу и картридж. Известно, что:

– джойстик и картридж купил не в *A*;

– в *C* зашел, когда уже купил дискеты и бумагу;

– в *D* не было ни картриджа, ни дискет;

– в *B* приехал, когда джойстик уже был куплен;

– из *D* уходил без джойстика.

Необходимо определить, в каком магазине были куплены дискеты.

11. Три подразделения *A*, *B*, *C* компании хотят получить по итогам квартала максимальную прибыль. Были сделаны следующие прогнозы:

a) если *A* получит максимальную прибыль, то максимальную прибыл получат также *B* и *C*;

b) либо *A* и *C* получат максимальную прибыль одновременно, либо одновременно не получат;

c) для того, чтобы *C* получило максимальную прибыль, необходимо, чтобы и *B* получило максимальную прибыль.

По завершении квартала оказалось, что один из трех прогнозов ложен. Какие из названных подразделений получили максимальную прибыль?

12. Есть две комнаты. В каждой из комнат будет находиться либо принцесса, либо тигр, хотя вполне может статься, что сразу в обеих комнатах обнаружится по тигру или там окажутся только принцессы. Узник должен угадать, кто находится в каждой комнате. Если он угадает, то женится на принцессе, если нет, то его растерзает тигр. На табличках, прикрепленных к дверям каждой из комнат, написано: «В этой комнате находится принцесса, а в другой комнате сидит тигр» и «В одной из этих комнат находится принцесса; кроме того, в одной из этих комнат сидит тигр», причем известно, что на одной табличке написана правда, а на другой – ложь.

Какую дверь на месте узника открыли бы вы?

13. В школе-новостройке в каждой из двух аудиторий может находиться либо кабинет информатики, либо кабинет физики. На дверях аудиторий повесили шутливые таблички. На первой аудитории повесили табличку «По крайней мере, в одной из этих аудиторий размещается кабинет информатики», а на второй – табличку с надписью «Кабинет физики находится в другой аудитории». Проверяющему, который пришел в школу, известно только, что надписи либо истинны, либо обе ложны. Помогите проверяющему найти кабинет информатики.

14. Фамилии командира взвода, заместителя командира взвода и командиров трех отделений – Антипов, Борисов, Васильев, Григорьев, Дмитриев. Фамилии командиров первого и второго отделений начинаются

на «А» и «Д», и они решили подружиться. Дмитриев недавно узнал, что командир взвода и командир второго отделения – двоюродные братья. Григорьев дружит с командиром и его заместителем. У Васильева нет двоюродных братьев. Определите фамилии командира, заместителя командира и командиров отделений.

Контрольные вопросы

1. Что такое высказывание?
2. Что такое логические операции?
3. Что такое отрицание?
4. Что такое конъюнкция?
5. Что такое дизъюнкция?
6. Что такое импликация?
7. Что такое эквивалентность?

10.9. Лабораторная работа № 9. Объектное программирование в CLIPS

Время выполнения – 4 часа.

Цель работы: научиться использовать объектно-ориентированные средства CLIPS для составления правил.

Задачи работы

1. Изучить объектно-ориентированные средства в CLIPS.
2. Выполнить задания в лабораторной работе.
3. Реализовать задание из лабораторной работы № 8 с помощью объектно-ориентированных средств программирования в CLIPS.
4. Ответить на контрольные вопросы.

Перечень обеспечивающих средств

Для выполнения работы необходимы:

– конспект лекций;

– персональный компьютер с установленными на нем приложениями Microsoft Office Word и CLIPS.

Общие теоретические сведения

Объектно-ориентированные средства в CLIPS позволяют значительно упростить программирование правил. Для обновления данных можно применять механизм передачи и обработки сообщений методами классов.

Общий синтаксис определения класса в CLIPS выглядит следующим образом, представленным на рис. 10.93.

```
(defclass <name> [<comment>] (is-a <superclass-name>+) [<role>]
[<pattern-match-role>] <slot>*
<handler-documentation>*)
<role> ::= (role concrete | abstract)
<pattern-match-role>
::= (pattern-match reactive | non-reactive)
<slot> ::= (slot <name> <facet>*) |
(single-slot <name> <facet>*) |
(multislot <name> <facet>*)
<facet> ::= <default-facet> | <storage-facet> |
<access-facet> | <propagation-facet> |
<source-facet> | <pattern-match-facet> [
<visibility-facet> | <create-accessor-facet>
<override-message-facet> | <constraint-attributes>
<default-facet> ::=
(default "DERIVE | ?NONE | <expression>*) |
(default-dynamic <expression>*) <storage-facet> ::= (storage local | shared)
<access-facet>
::= (access read-write \ read-only | initialize-only)
<propagation-facet> ::= (propagation inherit \ no-inherit)
<source-facet> ::= (source exclusive \ composite)
<pattern-match-facet>
::= (pattem-match reactive | non-reactive)
 <visibility-facet> ::= (visibility private | public)
<visibility-facet> ::= (visibility private | public)
<create-accessor-facet>
::= (create-accessor ?NONE \ read | write | read-write) <override-message-facet>
::= (override-message ?DEFAULT\ <message-name>) <handler-
documentation> :: = (message-handler <name> [<handler-type>]) <handler-
type> ::= primary | around | before | after
```

Рис. 10.93. Общий синтаксис определения класса

Если класс *B* наследуется от *A*, то *B* является потомком *A*, а *A* – родителем *B*. Каждый создаваемый пользователем класс должен быть напрямую унаследован хотя бы от одного класса (обычно от OBJECT). Случай прямого наследования от более чем одного класса называется *множественным наследованием*. Новый класс наследует слоты и сообщения от своих родителей. При множественном наследовании необходимо учитывать следующие правила:

1) класс имеет более высокое старшинство, чем его родители;

2) в списке классов-родителей классы должны располагаться по старшинству. При конфликте имен слотов или сообщений выбирается слот или сообщение более старшего класса.

Например, класс A является прямым наследником класса USER. Список старшинства классов для A: A USER OBJECT. Синтаксис будет выглядеть следующим образом: (defclass A (is-a USER)).

Класс B является прямым наследником класса USER. Список старшинства классов для B: B USER OBJECT. Синтаксис будет выглядеть следующим образом: (defclass B (is-a USER)).

Класс C является прямым наследником классов A и B. Список старшинства классов для C: C A B USER OBJECT. Синтаксис будет выглядеть следующим образом: (defclass C (is-a A B)).

Абстрактный класс предназначен только для наследования, на его основе не могут создаваться экземпляры. На основе конкретного класса могут создаваться его экземпляры [2].

Слот – это место для хранения значений поля класса. Каждый экземпляр класса содержит копию всех слотов своего родителя. Количество слотов класса ограничено только размером свободной памяти, имя слота – любой набор символов, за исключением зарезервированных слов. Потомок класса содержит слоты родителя. В случае конфликта имен слотов он разрешается в соответствии с правилом старшинства. Пример классов показан на рис. 10.94.

```
(defclass A (is-a USER) (slot fooA) (slot barA))

(defclass B (is-a A) (slot fooB) (slot barB))
```

Рис. 10.94. Слоты класса

Список старшинства для A: A USER OBJECT. Экземпляр класса A будет иметь два слота: fooA и barA. Список старшинства для B: B A USER OBJECT. Экземпляр класса B будет иметь четыре слота: fooB, barB, fooA, barA. Для того чтобы создать экземпляр класса, необходимо выполнить команду (make-instance a of A). В результате создается экземпляр с именем a класса A.

Другой вариант создания массива экземпляра классов: (defmstances my_inst (a of A) (b of A) (c of A)).

Для каждого слота может быть определен набор фасетов. Фасеты описывают различные свойства слотов: значения по умолчанию, вид хранения, видимость и т. п.

Слот может содержать как одно, так и несколько значений. По умолчанию слот содержит только одно значение. Ключевое слово multistat необходимо для установки типа слота. Он позволяет хранить несколько значений, a slot или singleslot устанавливает тип слота, который может содержать только одно значение [12].

Многозначные слоты хранятся как значения с несколькими полями. Манипуляции с ними производятся посредством стандартных функций nth$ и lengths. Для установки значения слота используется функция slotinsertS. Слоты с одним значением хранятся в CLIPS как обычные переменные стандартных типов. Пример слота показан на рис. 10.95.

```
CLIPS> (clear)

CLIPS>

(defclass A (is-a USER) (role concrete)

(multislot foo (create-accessor read) (default abc def ghi))) CLIPS> (make-
instance a of A) [a]

CLIPS> (nth$ 2 (send [a] get-foo)) def

CLIPS>
```

Рис. 10.95. Слоты с одним значением

Фасеты используются для задания значений слота по умолчанию при создании экземпляра класса. Фасет default используется для задания статических значений слота. Фасет default-dynamic используется для заданий значения слота, которое задается всякий раз при создании нового экземпляра класса. Пример фасета показан на рис. 10.96.

```
CLIPS> (clear) CLIPS> (setgen 1) 1

CL1PS>

(defclass A (is-a USER) (role concrete)

(slot foo (default-dynamic (gensym)) (create-accessor read))) CLIPS> (make-
instance al of A) [al]

CLIPS> (make-instance a2 of A) [a2]

CLIPS> (send [al] get-foo) genl

CL1PS> (send [a2] get-foo)

gen2

CLIPS>
```

Рис. 10.96. Фасет *default*

Фасет определяет, будет ли значение слота храниться локально в экземпляре класса (*local*) либо это значение будет одним для всех экземпляров класса (*shared*). Пример фасета *local* показан на рис. 10.97.

Фасеты для доступа к слоту бывают трех типов:

– Read-write;

– Read-only;

– Initialize-only.

Пример работы с разными типами фасетов показан на рис. 10.98.

```
CLIPS> (clear) CLIPS>
(defclass A (is-a USER) (role concrete)
(slot foo (create-accessor write) (storage shared) (default 1))
(slot bar (create-accessor write)
(storage shared)
(default-dynamic 2))
(slot woz (create-accessor write)
(storage local)))
CLIPS> (make-instance a of A)
[a]
CLIPS> (send [a] print)
[a] of A
(foo 1)
(bar 2)
(woz nil)
CLIPS> (send [a] put-foo 56)
  CLIPS> (send [a] put-bar 104) 104
  CLIPS> (make-instance b of A) [b]
  CL1PS> (send [b] print)
  [b] of A
  (foo 56)
  (bar 2)
  (woz nil)
  CLIPS> (send [b] put-foo 34) 34
  CLIPS> (send [b] put-woz 68) 68
  CLIPS> (send [a] print)
[a]    of A (foo 34) (bar 2) (woz nil)
CLIPS> (send [b] print)
[b]    of A (foo 34) (bar 2) (woz 68) CLIPS>
```

Рис. 10.97. Фасет *local*

CLIPS> (clear) CLIPS>

(defclass A (is-a USER) (role concrete)

(slot foo (create-accessor write) (access read-write)) (slot bar (access read-only) (default abc))
(slot woz (create-accessor write) (access initialize-only))) CLIPS>

(definessage-handler A put-bar (?value) (dynamic-put (sym-cat bar) ?value)) CLIPS> (make-instance a of A (bar 34)) [MSGFUN3] bar slot in [a] of A: write access denied.

[PRCCODE4] Execution halted during the actions of message-handler put-bar primary in class A FALSE

CLIPS> (make-instance a of A (foo 34) (woz 65)) ta]

CLIPS> (send [a] put-bar 1)

[MSGFUN3] bar slot in [a] of A: write access denied.

[PRCCODE4] Execution halted during the actions of message-handler put-bar primary in class A FALSE

CLIPS> (send [a] put-woz 1)

[MSGFUN3] woz slot in [a] of A: write access denied.

[PRCCODE4] Execution halted during the actions of message-handler put-bar primary in class A FALSE

CLIPS> (send [a] print)

[a] of A

(foo 34)

(bar abc)

(woz 65)

CLIPS>

Рис. 10.98. Работа с разными типами фасетов

Для изменения значений свойств объектов по правилам объектно-ориентированного программирования производится самими объектами. Для этого в языке CLIPS реализован обработчик сообщений.

Общий синтаксис команды создания обработчика сообщений: (defmessage-handler <class-name> <message-name> [<handler-type>] [<comment>] (<parameter>* [<wildcard-parameter>]) <action>*).

Для вызова обработчика сообщений экземпляра класса служит команда (send [имя экземпляра] имя метода параметры) [13].

Обработчик сообщений можно уникально идентифицировать с помощью наименования класса и типа. Для класса обработчик сообщений может задаваться как при создании определения класса, так и после. Заметим, что при создании определения класса создается только заголовок обработчика сообщений. Собственно программный код обработчика создается позже при помощи команды defmessage-handler.

Методика выполнения работы

1. Продажа билетов. Имеется определенное количество билетов на различные мероприятия. Необходимо по запросу и заданному количеству монет выдавать билеты. Автомат должен оповещать об окончании билетов на запрашиваемое мероприятие. Для решения данной задачи создадим код программы в текстовом редакторе. В *defrule* записываем переменную *money*. Она определяет, сколько рублей посетитель положит в автомат. Выводим программу мероприятий, которые имеются, и их цену. Это показано на рис. 10.99.

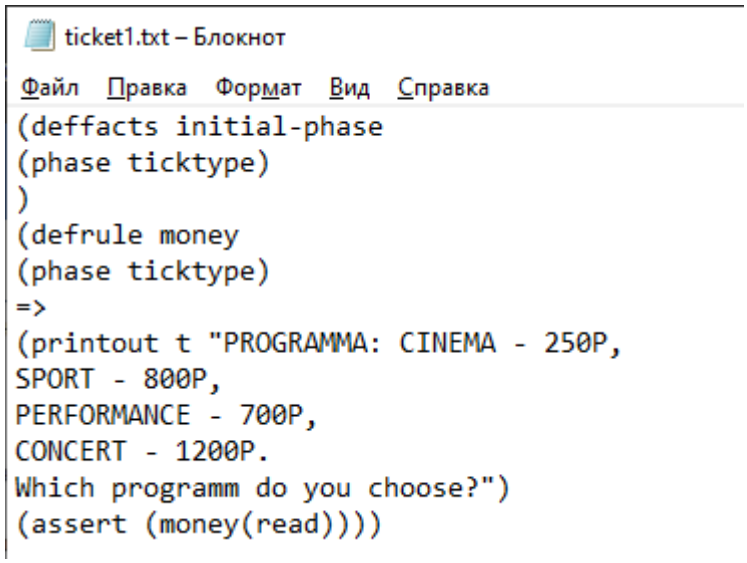

Рис. 10.99. Запись в правило переменной *money*

Для реализации автомата по продаже билетов создаем класс *ticket-selling*, родительским классом которого является *USER*, класс не абстрактный (*role concrete*). Чтобы класс мог быть использован при сопоставлении образцов во время выполнения правил pattern-match, он должен быть выставлен как reactive.

Класс имеет четыре слота: *slot ticket-cinema*, *slot ticket-sport*, *slot ticket-performance*, *slot ticket-concert*. Они доступны как для чтения, так и для записи. Слоты будут использоваться для хранения количества билетов, их тип задан как *INTEGER* (рис. 10.100).

```
(defclass ticket-selling
(is-a USER)
(role concrete)
(pattern-match reactive)
(slot ticket-cinema (type INTEGER) (create-accessor read-write))
(slot ticket-sport (type INTEGER) (create-accessor read-write))
(slot ticket-performance (type INTEGER) (create-accessor read-write))
(slot ticket-concert (type INTEGER) (create-accessor read-write))
(message-handler getticket)
)
```

Рис. 10.100. Создание класса

Далее определяем экземпляр данного класса *tickets1* с объектом *our-ticket*. Задаем билеты в кино – 30 штук, билеты на спорт – 40, билеты на спектакль – 45, билеты на концерт – 50 штук (рис. 10.101).

```
(definstances tickets1
(our-ticket of ticket-selling
(ticket-cinema 30)
(ticket-sport 40)
(ticket-performance 45)
(ticket-concert 50)
))
```

Рис. 10.101. Экземпляр класса

Выдачу билетов реализуем в обработчике сообщений *getticket*, в который в качестве параметра будем передавать номинал купюры.

Задаем следующие условия (рис. 10.102).

1) посетитель ввел 250: если количество билетов в кино больше 0, то выдается билет в кино, иначе выходит сообщение «Билетов в кино нет»;

2) посетитель ввел 800: если количество билетов на спорт больше 0, то выдается билет на спорт, иначе выходит сообщение «Билетов на спорт нет»;

3) посетитель ввел 700: если количество билетов на спектакль больше 0, то выдается билет на спектакль, иначе выходит сообщение «Билетов на спектакль нет»;

4) посетитель ввел 1 200: если количество билетов на концерт больше 0, то выдается билет на концерт, иначе выходит сообщение «Билетов на концерт нет».

Иначе появляется сообщение, что нет никаких билетов.

```
(defmessage-handler ticket-selling getticket (?money)
(if (= ?money 250) then
(if (> (dynamic-get ticket-cinema) 0) then
(dynamic-put ticket-cinema (- (dynamic-get ticket-cinema) 1))
(printout t "Ticket purchased on cinema" crlf)
else
(printout t "No tickets on cinema" crlf)
)
else
(if (= ?money 800) then
(if (> (dynamic-get ticket-sport) 0) then
(dynamic-put ticket-sport (- (dynamic-get ticket-sport) 1))
(printout t "Ticket purchased on sport" crlf)
else
(printout t "No tickets on sport" crlf)
)
else
(if (= ?money 700) then
(if (> (dynamic-get ticket-performance) 0) then
(dynamic-put ticket-performance (- (dynamic-get ticket-performance) 1))
(printout t "Ticket purchased on performance" crlf)
else
(printout t "No tickets on performance" crlf)
)
else
(if (= ?money 1200) then
(if (> (dynamic-get ticket-concert) 0) then
(dynamic-put ticket-concert (- (dynamic-get ticket-concert) 1))
(printout t "Ticket purchased on concert" crlf)
else
(printout t "No tickets on concert" crlf)
)
else
(printout t "No tickets" crlf)
)))))
```

Рис. 10.102. Задание условий

Запускаем и добавляем код в программу (рис. 10.103).

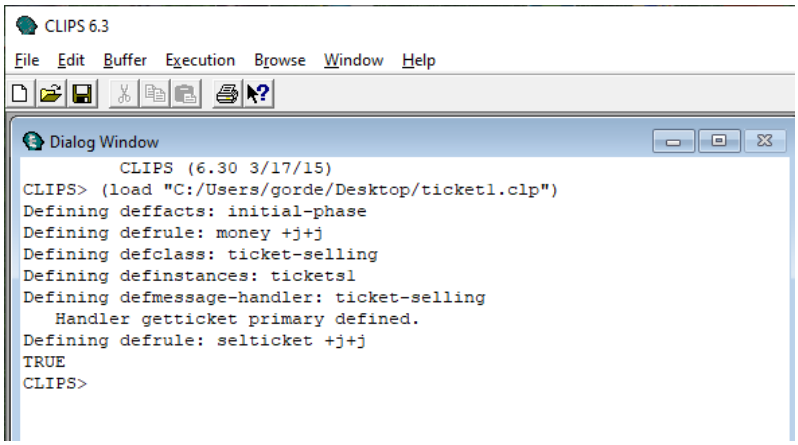

Рис. 10.103. Запуск программы

После ввода команды *Run* запускается интерпретатор и программа выполняется.

Выводится программа мероприятий на выбор с указанными ценами в рублях (рис. 10.104).

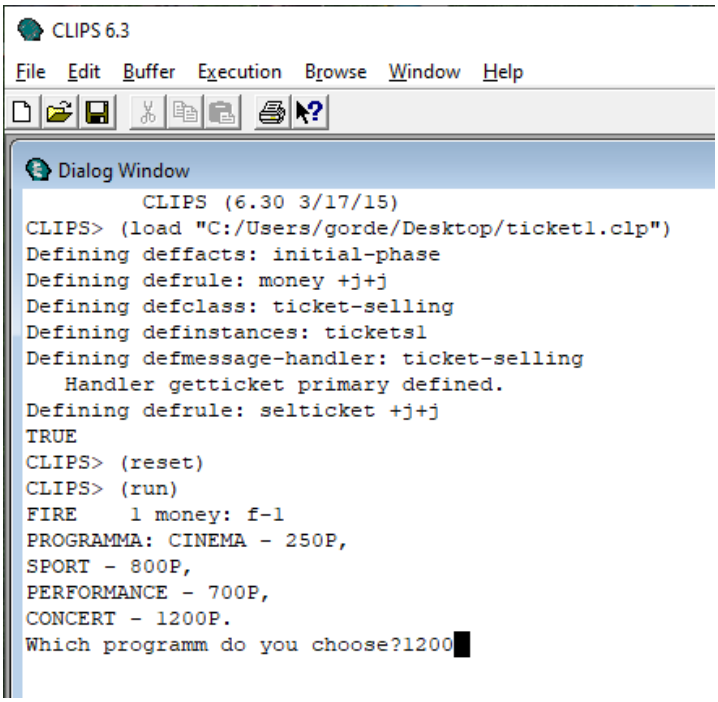

Рис. 10.104. Выполнение программы

Также задается вопрос *Какую из программ вы выберете?* Набираем *1200P*. И видим, что автомат выдает сообщение *Билет приобретен на концерт*. Или же вводим *250P*, после чего выводится сообщение *Билет приобретен в кино* (рис. 10.105).

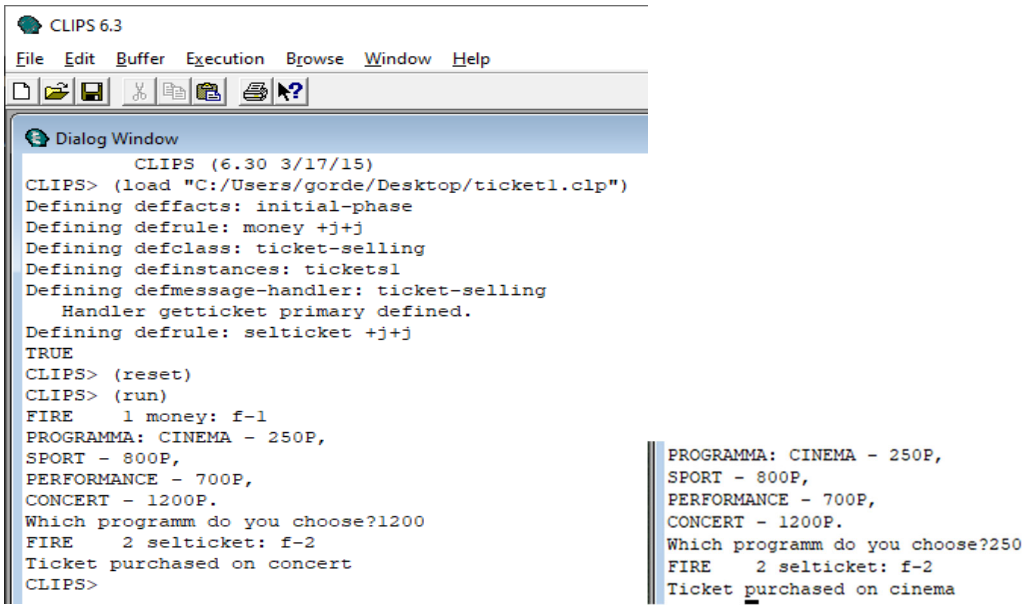

Рис. 10.105. Работа программы

Если же вводим количество денег, отличное от стоимости билетов на мероприятия, например, введем *100P*, то получаем сообщение «Нет билетов на указанную сумму» (рис. 10.106).

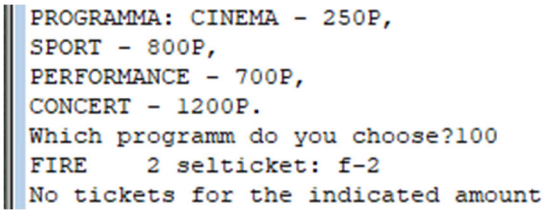

Рис. 10.106. Работа программы

2. Разработка предметной области, связанной с управлением грузами лифтом. Лифт может перемещаться между этажами, согласно вызовам перевозить грузы, учитывая их вес.

Для решения данной задачи создается два класса: *lift-automate* и *cargo*.
Класс *lift-automate*. Слоты:
– *location* – этаж, на котором находится лифт;
– *amount* – количество груза в лифте.
Сообщения:
– *load* – погрузить груз в лифт;
– *carry* – перевезти груз на другой этаж;
– *unload* – выгрузить груз на этаже;
– *move* – переместиться на другой этаж.
Класс *cargo.* Слоты:
– *location* – этаж, на котором находится груз;
– *amount* – количество груза на этаже.
Сообщения:
– *enough-cargo* – проверяет, достаточно ли груза на этаже для перевозки.

Определенные объекты:
– *my-lift* класса *lift-automate* на этаже Floor1 колчество груза – 0 т;
– *cargo1* класса *cargo* на этаже *Floor1*, количество груза – 225 т;
– *cargo2* класса *cargo* на этаже *Floor2*, количество груза – 40 т;
– *cargo3* класса *cargo* на этаже *Floor3*, количество груза – 25 т;
– *cargo4* класса *cargo* на этаже *Floor4*, количество груза – 15 т;
– *cargo5* класса *cargo* на этаже *Floor5*, количество груза – 22 т.
Факты: цель – перевезти 12 т груза с этажа *Floor2* на этаж *Floor5*.
Листинг программы указан ниже.

```
(defclass lift-automate (is-a USER)
(role concrete)
(pattern-match reactive)
(slot location (type SYMBOL)(create-accessor read-write))
(slot amount (type INTEGER)(create-accessor read-write))
(message-handler load)
(message-handler carry)
(message-handler unload)
(message-handler move)
```

```
)
(defclass cargo (is-a USER)
(role concrete)
(pattern-match reactive)
(slot location (type SYMBOL)(create-accessor read-write))
(slot amount (type INTEGER)(create-accessor read-write))
(message-handler enough-cargo)
)

(defmessage-handler lift-automate load(?from ?amt)
(if (eq ?self:location (send ?from get-location)) then
(if (> ?amt 10) then (printout t "Lift over limit!" crlf)
else
(dynamic-put amount ?amt)
(send ?from put-amount (- (send ?from get-amount) ?amt))
(printout t "Loading " ?amt "t of cargo..." crlf)
)
else
(printout t "Wrong floor. Moving..." crlf)
(send ?self move ?from)
) )

(defmessage-handler lift-automate unload(?to ?amt)
(if (eq ?self:location (send ?to get-location)) then
(if (> ?amt 10) then (printout t "Lift over limit!" crlf)
else
(dynamic-put amount 0)
(send ?to put-amount (+ (send ?to get-amount) ?amt))
(printout t "Unloading " ?amt "t of cargo..." crlf)
)
else
(printout t "Wrong floor. Moving..." crlf)
(send ?self move ?to)
) )
```

```
(defmessage-handler lift-automate carry(?from-loc ?to-loc ?amt)
(bind ?multifind (find-instance ((?from cargo) (?to cargo)) (and (eq ?from:location ?from-loc)(eq ?to:location ?to-loc))))
(bind ?x ?amt)
(while (> ?x 0)
(if (and (eq ?self:location ?from-loc)(send (nth$ 1 ?multifind) enough-cargo ?x)) then
(if (>= ?x 10) then
(send ?self load (nth$ 1 ?multifind) 10)
(dynamic-put location ?to-loc)
(printout t "Carrying " 10 "t of cargo..." crlf)
(send ?self unload (nth$ 2 ?multifind) 10)
(bind ?x (- ?x 10))
else
(send ?self load (nth$ 1 ?multifind) ?x)
(dynamic-put location ?to-loc)
(printout t "Carrying " ?x "t of cargo..." crlf)
(send ?self unload (nth$ 2 ?multifind) ?x)
(bind ?x 0) )
else
(if (not (send (nth$ 1 ?multifind) enough-cargo ?x)) then
(printout t "Not enough cargo on " (send (nth$ 1 ?multifind) get-location) crlf)
(break)
else
(printout t "Wrong floor. Moving..." crlf)
(send ?self move (nth$ 1 ?multifind)))
) )
(printout t "Done." crlf)
)

(defmessage-handler lift-automate move(?to)
(dynamic-put location (send ?to get-location))
)
(defmessage-handler cargo enough-cargo(?amt)
(return (>= ?self:amount ?amt))
)
```

```
(definstances instan
(my-lift of lift-automate (location Floor1)(amount 0))
(cargo1 of cargo (location Floor1)(amount 225))
(cargo2 of cargo (location Floor2)(amount 40))
(cargo3 of cargo (location Floor3)(amount 25))
(cargo4 of cargo (location Floor4)(amount 15))
(cargo5 of cargo (location Floor5)(amount 22))
)

(deftemplate goal
(slot amount (type INTEGER)) (slot from (type SYMBOL)) (slot to (type SYMBOL)) )

(deffacts world
(goal (amount 12) (from Floor2) (to Floor5) ) )

(defrule move-cargo
?f <- (goal(amount ?X)(from ?Y)(to ?Z)) =>
(send [my-lift] carry ?Y ?Z ?X)
(retract ?f)
).
```

Вывод результата работы программы в консоль:

```
CLIPS> (load "C:/Users/DNSPC/Desktop/В_Группу/Intel/lr8/lift.clp")
Defining defclass: lift-automate
Defining defclass: cargo
Defining defmessage-handler: lift-automate
Handler load primary defined.
Defining defmessage-handler: lift-automate
Handler unload primary defined.
Defining defmessage-handler: lift-automate
Handler carry primary defined.
Defining defmessage-handler: lift-automate
Handler move primary defined.
Defining defmessage-handler: cargo
```

```
Handler enough-cargo primary defined.
Defining definstances: instan
Defining deftemplate: goal
Defining deffacts: world
Defining defrule: move-cargo +j+j
TRUE
CLIPS> (reset)
<== f-0   (initial-fact)
::= local slot location in instance my-lift <- Floor1
::= local slot amount in instance my-lift <- 0
::= local slot location in instance cargo1 <- Floor1
::= local slot amount in instance cargo1 <- 225
::= local slot location in instance cargo2 <- Floor2
::= local slot amount in instance cargo2 <- 40
::= local slot location in instance cargo3 <- Floor3
::= local slot amount in instance cargo3 <- 25
::= local slot location in instance cargo4 <- Floor4
::= local slot amount in instance cargo4 <- 15
::= local slot location in instance cargo5 <- Floor5
::= local slot amount in instance cargo5 <- 22
==> f-0   (initial-fact)
==> f-1   (goal (amount 12) (from Floor2) (to Floor5))
CLIPS> (run)
FIRE  1 move-cargo: f-1
Wrong floor. Moving...
::= local slot location in instance my-lift <- Floor2
::= local slot amount in instance my-lift <- 10
::= local slot amount in instance cargo2 <- 30
Loading 10t of cargo...
::= local slot location in instance my-lift <- Floor5
Carrying 10t of cargo...
::= local slot amount in instance my-lift <- 0
::= local slot amount in instance cargo5 <- 32
Unloading 10t of cargo...
```

```
Wrong floor. Moving...
::= local slot location in instance my-lift <- Floor2
::= local slot amount in instance my-lift <- 2
::= local slot amount in instance cargo2 <- 28
Loading 2t of cargo...
::= local slot location in instance my-lift <- Floor5
Carrying 2t of cargo...
::= local slot amount in instance my-lift <- 0
::= local slot amount in instance cargo5 <- 34
Unloading 2t of cargo...
Done.
<== f-1    (goal (amount 12) (from Floor2) (to Floor5))
CLIPS>.
```

Результат работы программы показан на рис. 10.107.

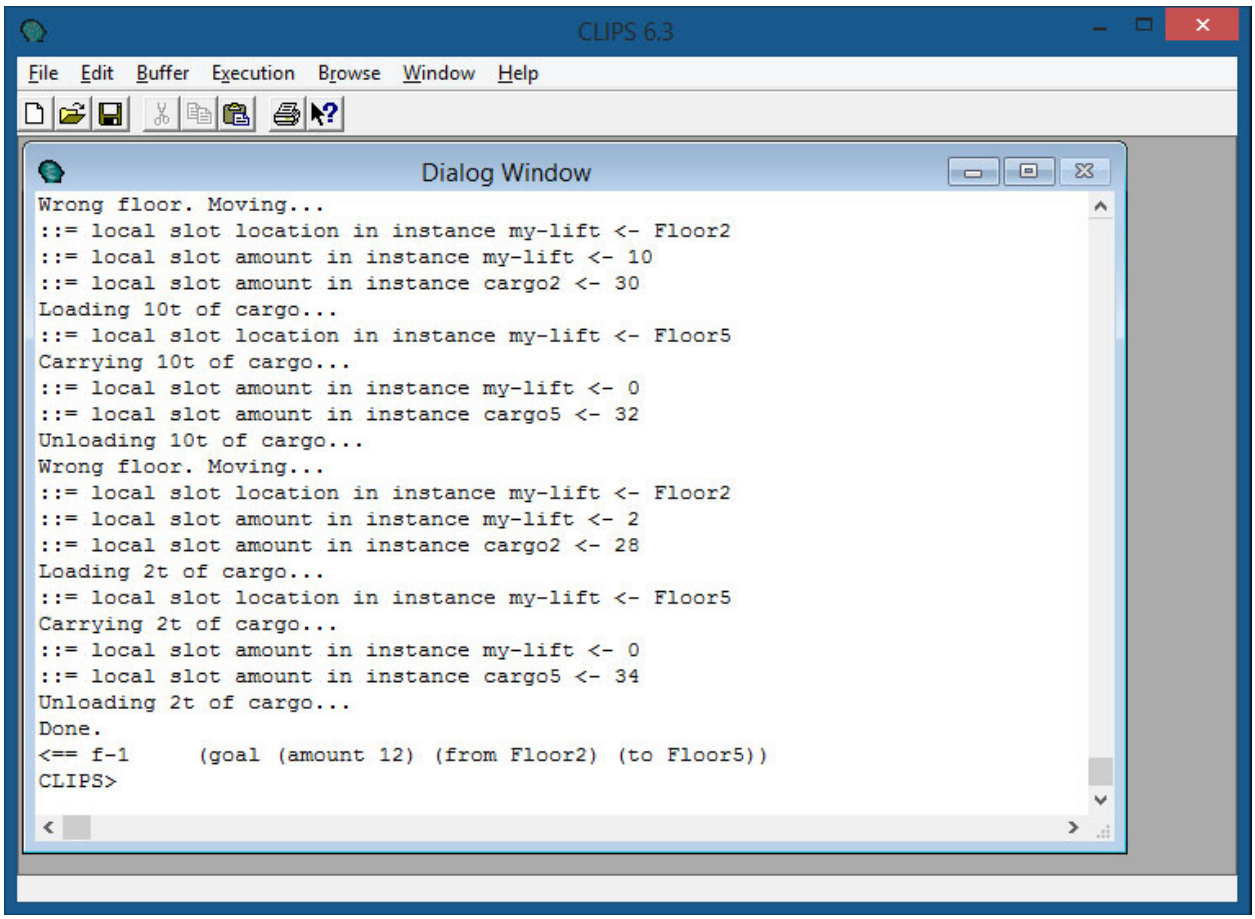

Рис. 10.107. Результат работы программы *lift.clp*

3. Разработка программы «Шиномонтажная мастерская» при помощи средств объектно-ориентированного программирования в CLIPS на планирование действий технического объекта.

Робот должен проводить диагностику и смену колеса с пробитой покрышкой. Для каждой марки автомашины определен свой класс колес.

Колес ограниченное количество, если какие-либо колеса отсутствуют, программа должна известить об этом.

Листинг программы указан ниже:

```
deffacts world
(car (type-w-c 1) (w1 1) (w2 1) (w3 1) (w4 0))
(car (type-w-c 2) (w1 1) (w2 1) (w3 1) (w4 1))
(car (type-w-c 3) (w1 0) (w2 1) (w3 1) (w4 1))
(car (type-w-c 4) (w1 1) (w2 0) (w3 0) (w4 1))
(car (type-w-c 5) (w1 0) (w2 1) (w3 1) (w4 0))
(
defclass auto-m
(is-a USER)
(role concrete)
(pattern-match reactive)
(slot count-type-w1 (type INTEGER) (create-accessor read-write))
(slot count-type-w2 (type INTEGER) (create-accessor read-write))
(slot count-type-w3 (type INTEGER) (create-accessor read-write))
(slot count-type-w4 (type INTEGER) (create-accessor read-write))
(slot count-type-w5 (type INTEGER) (create-accessor read-write))
)
(
definstances robot
(
ourwheel of auto-m
(count-type-w1 4)
(count-type-w2 2)
(count-type-w3 0)
(count-type-w4 2)
(count-type-w5 0)
)
)
```

```
(
deftemplate car
(slot type-w-c)
(slot w1)
(slot w2)
(slot w3)
(slot w4)
)
(
deffacts world
(car (type-w-c 1) (w1 1) (w2 1) (w3 1) (w4 0))
(car (type-w-c 2) (w1 1) (w2 1) (w3 1) (w4 1))
(car (type-w-c 3) (w1 0) (w2 1) (w3 1) (w4 1))
(car (type-w-c 4) (w1 1) (w2 0) (w3 0) (w4 1))
(car (type-w-c 5) (w1 0) (w2 1) (w3 1) (w4 0))
)
(defmessage-handler auto-m change-wheel1 (?w1 ?w2 ?w3 ?w4)
(if (= (dynamic-get count-type-w1) 0) then
(printout t "Not wheel 1 class for car" crlf)
)
)
(defmessage-handler auto-m change-wheel2 (?w1 ?w2 ?w3 ?w4)
(if (= (dynamic-get count-type-w2) 0) then
(printout t "Not wheel 2 class for car" crlf)
)
)
(defmessage-handler auto-m change-wheel3 (?w1 ?w2 ?w3 ?w4)
(if (= (dynamic-get count-type-w3) 0) then
(printout t "Not wheel 3 class for car" crlf)
)
)
(defmessage-handler auto-m change-wheel4 (?w1 ?w2 ?w3 ?w4)
(if (= (dynamic-get count-type-w3) 0) then
(printout t "Not wheel 4 class for car" crlf)
)
)
```

```
(defmessage-handler auto-m change-wheel5 (?w1 ?w2 ?w3 ?w4)
(if (= (dynamic-get count-type-w3) 0) then
(printout t "Not wheel 5 class for car" crlf)
)
)
(defrule change-w1
(car (type-w-c ?type-w-c) (w1 ?w1) (w2 ?w2) (w3 ?w3) (w4 ?w4))
(test (= ?type-w-c 1))
=>
(send [ourwheel] change-wheel1 ?w1 ?w2 ?w3 ?w4)
)
(defrule change-w2
(car (type-w-c ?type-w-c) (w1 ?w1) (w2 ?w2) (w3 ?w3) (w4 ?w4))
(test (= ?type-w-c 1))
=>
(send [ourwheel] change-wheel2 ?w1 ?w2 ?w3 ?w4)
)
(defrule change-w3
(car (type-w-c ?type-w-c) (w1 ?w1) (w2 ?w2) (w3 ?w3) (w4 ?w4))
(test (= ?type-w-c 1))
=>
(send [ourwheel] change-wheel3 ?w1 ?w2 ?w3 ?w4)
)
(defrule change-w4
(car (type-w-c ?type-w-c) (w1 ?w1) (w2 ?w2) (w3 ?w3) (w4 ?w4))
(test (= ?type-w-c 1))
=>
(send [ourwheel] change-wheel4 ?w1 ?w2 ?w3 ?w4)
)
(defrule change-w5
(car (type-w-c ?type-w-c) (w1 ?w1) (w2 ?w2) (w3 ?w3) (w4 ?w4))
(test (= ?type-w-c 1))
=>
(send [ourwheel] change-wheel5 ?w1 ?w2 ?w3 ?w4)
).
```

Запустим программу и проверим ее работоспособность (рис. 10.108).

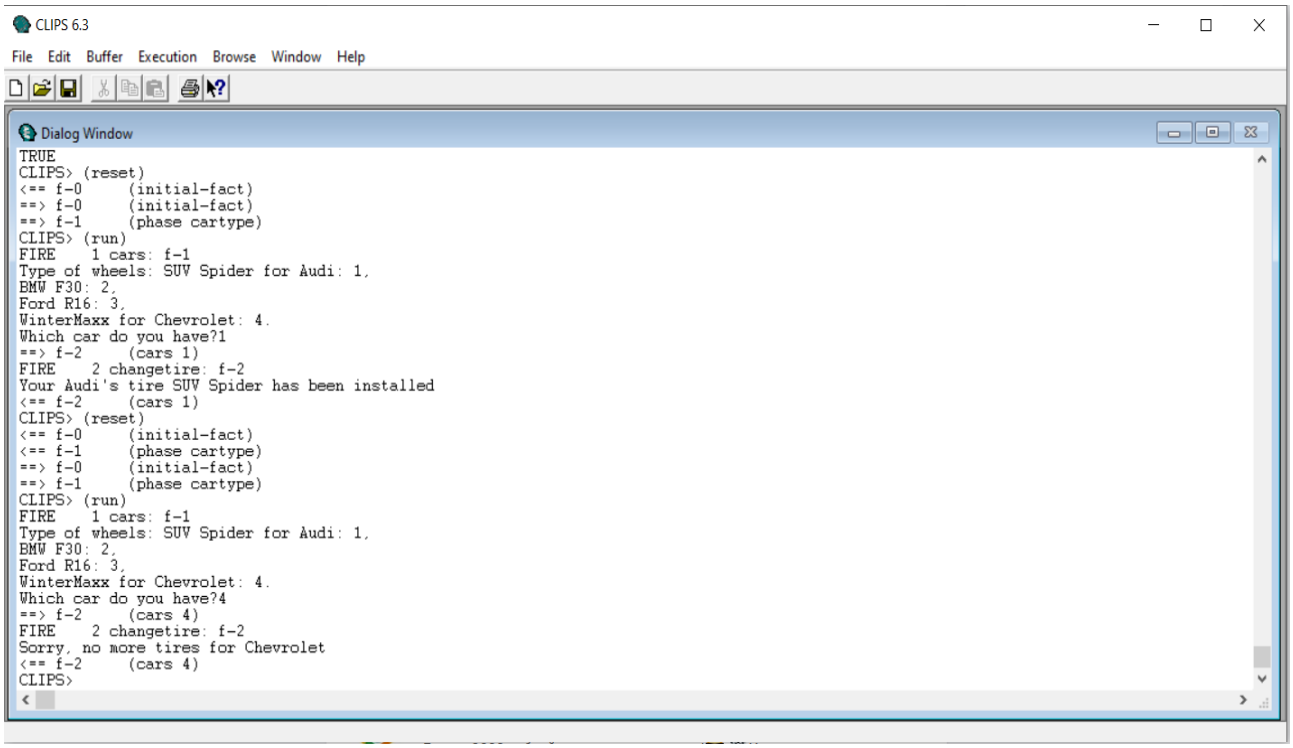

Рис. 10.108. Работа программы

В случае выбора типа автомобиля Audi колесо успешно заменяется, в случае же выбора типа автомобиля Chevrolet программа сообщает, что колес для Chevrolet не осталось.

4. Выполните вариант задания, выданного преподавателем из лабораторной работы № 8 с помощью объектно-ориентированных средств в CLIPS. Выполните полученную программу с различными начальными данными.

Контрольные вопросы

1. Что такое класс в языке CLIPS?
2. Что такое слот?
3. Что такое фасеты?
4. Что такое обработчик сообщений?
5. Какие объектно-ориентированные средства в CLIPS существуют?

ЗАКЛЮЧЕНИЕ

В наши дни интеллектуальные системы получили широкое распространение в различных сферах человеческой деятельности. Стремительный рост сложности информационных процессов в различных предметных областях требует использования новых интеллектуальных подходов, основанных на обработке знаний. Одновременно был осуществлен прорыв в системе программирования путем создания новых декларативных языков, программных продуктов и приложений. Это относится прежде всего к экспертным системам

При всем многообразии видов интеллектуального программного обеспечения, имеющегося в настоящее время, мы достаточно подробно рассмотрели среду CLIPS. Это объясняется ее доступностью и тем, что язык и среда CLIPS предоставляют пользователям возможность быстро создавать эффективные, компактные и легко управляемые экспертные системы. Несмотря на то, что CLIPS распространяется бесплатно, он весьма успешно конкурирует даже с самыми известными коммерческими проектами.

Учебное пособие является кратким руководством для проведения учебных занятий по специальности 09.03.02 Методы искусственного интеллекта.

Учебное пособие включает контрольные вопросы, что даст возможность студенту проверить качество усвоения материала.

Автор надеется, что данный учебник будет способствовать повышению качества подготовки бакалавров, а также будет полезен всем читателям, интересующимся современным состоянием и перспективами развития информационных систем и технологий.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Андрейчиков А. В. Интеллектуальные информационные системы : учеб. для вузов. – М. : Финансы и статистика, 2004. – 424 с.

2. Гаврилова Т. А., Хорошевский В. Ф. Базы знаний интеллектуальных систем. – СПб. : Питер, 2001. – 384 с.

3. Гаскаров Д. В. Интеллектуальные информационные системы : учеб. для вузов. – М. : Высшая школа, 2003. – 431 с.

4. Генетические алгоритмы, искусственные нейронные сети и проблемы виртуальной реальности / Г. К. Вороновский и [др.]. – Харьков : ОСНОВА, 1997. – 112 с.

5. Джарратано Д., Райлт Г. Экспертные системы: принципы разработки и программирование ; пер. с англ. – 4-е изд. – М. : ООО «И. Д. Вильямс», 2007. – 1152 с.

6. Круглов В. В., Дли М. И., Голунов Р. Ю. Нечеткая логика и искусственные нейронные сети. – М. : ФИЗМАТЛИТ, 2001. – 224 с.

7. Любарский Ю. Я. Интеллектуальные информационные системы. – М. : Наука, 1990. – 227 с.

8. Макаров И. М., Топчеев Ю. И. Робототехника: история и перспективы. – М. : Наука : изд-во МАИ, 2003. – 349 с.

9. Методы и модели анализа данных: OLAP и Data Mining / А. А. Барсегян и [др.]. – СПб. : БХВ-Петербург, 2004. – 336 с.

10. Миркес Е. М. Нейроинформатика : учеб. пособие для студентов. – Красноярск : ИПЦ КГТУ, 2002. – 347 с.

11. Нейроинформатика / А. Н. Горбань и [др.]. – Новосибирск : Наука : Сибирское предприятие РАН, 1998. – 296 с.

12. Нечеткие гибридные системы / И. З. Батыршин и [др.]. ; под ред. Н. Г. Ярушкиной. – М. : ФИЗМАТЛИТ, 2007. – 208 с.

13. Прикладные нечеткие системы ; пер. с яп. / К. Асаи и др. ; под ред. Т. Тэрано, К. Асаи, М. Сугэно. – М. : Мир, 1993. – 368 с.

14. Рассел С., Норвиг П. Искусственный интеллект: современный подход. – 2-е изд. – М. : Вильямс, 2006. – 1408 с.

15. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы ; пер. с польск. И. Д. Рудинского. – М. : Горячая линия – Телеком, 2008. – 452 с.

16. Ручкин В. Н., Фулин В. А. Универсальный искусственный интеллект и экспертные системы. – СПб. : БХВ-Петербург, 2009. – 240 с.

17. Уоссермен Ф. Нейрокомпьютерная техника : теория и практика ; пер. с англ. – 1992. – 118 с.

18. Уэно Х. Представление и использование знаний. – М. : Мир, 1989. – 220 с.

19. Фогель Л., Оуэнс А., Уолш М. Искусственный интеллект и эволюционное моделирование. – М. : Мир, 1969. – 230 с.

20. Частиков А. П., Гаврилова Т. А., Белов Д. Л. Разработка экспертных систем. Среда CLIPS. – СПб. : БХВ-Петербург, 2003. – 608 с.

21. Шеховцов О. И. Представление знаний : учеб. пособие. – СПб. : ГЭТУ, 1997. – 56 с.

22. Ярушкина Н. Г. Основы теории нечетких и гибридных систем : учеб. пособие. – М. : Финансы и статистика, 2004. – 320 с.

Учебное издание

Басаргин Андрей Александрович

МЕТОДЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Редактор *О. В. Георгиевская*

Компьютерная верстка *Н. Ю. Леоновой*

Изд. лиц. ЛР № 020461 от 04.03.1997.
Подписано в печать 27.05.2022. Формат 60 × 84 1/16.
Усл. печ. л. 9,53. Тираж 50 экз. Заказ 78.
Гигиеническое заключение
№ 54.НК.05.953.П.000147.12.02. от 10.12.2002.

Редакционно-издательский отдел СГУГиТ
630108, Новосибирск, ул. Плахотного, 10.

Отпечатано в картопечатной лаборатории СГУГиТ
630108, Новосибирск, ул. Плахотного, 8.