

**Отзыв на бакалаврскую работу**  
**«Искусственный интеллект на базе нейронной сети,**  
**реализующий перемещение модели сложного объекта в пространстве»**  
**студентки института электроники и светотехники**  
**Мордовского государственного университета им. Н. П. Огарева**  
**направления подготовки «Информатика и вычислительная техника»**  
**Немчиновой Елены Андреевны**

Бакалаврская работа выполнена в полном соответствии с заданием и представлена к заданному сроку.

За время написания работы Немчинова Е. А. проявила себя как грамотный исследователь, инженер и программист. На достаточно высоком уровне она овладела такими инструментами и технологиями, как MS Visual Studio, MS Visual C++ CLR, OpenGL, MS Windows Forms, ООП и другие.

Немчинова Е. А. детально изучила теоретические основы построения и применения искусственных нейронных сетей, особенности моделирования перемещения объектов в пространстве под действием внешних сил, построения прототипов моделей машинного обучения.

В результате Немчиновой Е. А. было разработано приложение для создания упрощенных моделей сложных объектов и построения и применения искусственной нейронной сети для управления перемещением созданных объектов в двумерном пространстве. С помощью данного приложения был проведен ряд исследований с целью выявления наиболее удачных параметров моделей объекта и искусственной нейронной сети.

Проверка пояснительной записки к бакалаврской работе на системе «Антиплагиат» показала процент оригинальности 94,73%.

Считаю, что бакалаврская работа отвечает всем необходимым требованиям, предъявляемым к выпускным квалификационным работам, а Немчинова Е. А. заслуживает присвоения ей квалификации бакалавра по направлению подготовки «Информатика и вычислительная техника».

Руководитель

к. т. н.,

старший преподаватель кафедры АСОИУ



Н. П. Плотникова

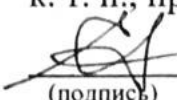
Федеральное государственное бюджетное образовательное  
учреждение высшего образования  
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ МОРДОВСКИЙ  
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ им. Н. П. ОГАРЁВА»

Институт электроники и светотехники

Кафедра автоматизированных систем обработки информации и управления

УТВЕРЖДАЮ

Зав. кафедрой  
к. т. н., проф.

 С. А. Федосин  
(подпись)


«20» 06 2018 г.

**БАКАЛАВРСКАЯ РАБОТА**  
**ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ НА БАЗЕ НЕЙРОННОЙ**  
**СЕТИ, РЕАЛИЗУЮЩИЙ ПЕРЕМЕЩЕНИЕ МОДЕЛИ**  
**СЛОЖНОГО ОБЪЕКТА В ПРОСТРАНСТВЕ**

Автор бакалаврской работы 20.06.2018г.  Е. А. Немчинова

Обозначение бакалаврской работы БР-02069964-090301-16-18

Направление 09.03.01 Информатика и вычислительная техника

Руководитель работы 20.06.2018г.  Н. П. Плотникова  
канд. техн. наук, ст. преподаватель

Нормоконтролер 20.06.2018г.  С. А. Ямашкин  
канд. техн. наук, ст. преподаватель


Саранск 2018

Федеральное государственное бюджетное образовательное  
учреждение высшего образования  
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ МОРДОВСКИЙ  
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ им. Н. П. ОГАРЁВА»

Институт электроники и светотехники  
Кафедра автоматизированных систем обработки  
информации и управления

УТВЕРЖДАЮ

Зав. кафедрой  
к. т. н., проф.

 С. А. Федосин  
(подпись)

«11» 12 2017 г.

**ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ**

(в форме бакалаврской работы)

Студент Немчинова Елена Андреевна

1 Тема «Искусственный интеллект на базе нейронной сети, реализующий  
перемещение модели сложного объекта в пространстве»

Утверждена по МордГУ № 9856-с \_\_\_\_\_ от 08.12.2017 г.

2 Срок представления работы к защите 27.06.2018

3 Исходные данные для выпускной квалификационной работы задание на  
выпускную квалификационную работу

4 Содержание выпускной квалификационной работы

4.1 Основные положения теории нейронных сетей

4.2 Разработка модели сложного объекта

4.3 Разработка искусственной нейронной сети

4.4 Реализация приложения для моделирования процесса перемещения сложного объекта в пространстве

4.5 Анализ результатов

5 Перечень графического материала модель сложного объекта, процесс создания модели объекта, структура искусственной нейронной сети, формулы расчета вознаграждения, диаграмма классов искусственной нейронной сети и модели объекта, схема взаимодействия модулей приложения, анализ эффективности, анализ эффективности перемещения различных моделей

6 Приложения программный код, графический материал

Руководитель работы

 11.12.17  
подпись, дата

Н. П. Плотникова

Задание принял к исполнению

 11.12.2017  
подпись, дата

Е. А. Немчинова

№ строки	Формат	Обозначение	Наименование	Кол. листов	Примечание
1					
2			Документация текстовая		
3					
4	A4	БР-02069964-090301-16-18	Пояснительная записка	63	
5					
6			Документация графическая		
7					
8	A1	БР-02069964-090301-16-18	Модель сложного объекта	1	
9			Процесс создания модели		
10			объекта		
11	A1	БР-02069964-090301-16-18	Структура нейронной сети	1	
12			Формулы расчета		
13			вознаграждения		
14	A1	БР-02069964-090301-16-18	Диаграмма классов		
15			искусственной нейронной сети		
16			и модели объекта		
17			Схема взаимодействия модулей		
18			приложения		
19	A1	БР-02069964-090301-16-18	Анализ эффективности	1	
20	A1	БР-02069964-090301-16-18	Анализ эффективности	1	
21			перемещения различных моделей		
22					
23			Документация прочая		
24					
25	A4	БР-02069964-090301-16-18	Программный код	29	Прил. А
26					
27	A4	БР-02069964-090301-16-18	Графический материал	6	Прил. Б

Подп и дата  
 Инв № дудл  
 Взам инв №  
 Подп и дата  
 Инв № подл

Изм	Лист	№ докум.	Подп	Дата
Разраб		Немчинова		20.06.18
Проб		Плотникова		20.06.18
Н контр		Ямашкин		20.06.18
Утв		Федосин		20.07

БР-02069964-090301-16-18

Искусственный интеллект на базе нейронной сети, реализующий перемещение модели сложного объекта в пространстве

Лит	Лист	Листов
	4	63

МГУ им Н.П. Огарева,  
ИЭС, АСОИЧ, ИВТ 441

## Реферат

Пояснительная записка содержит 63 листа, 33 рисунка, 6 таблиц, 9 использованных источников, 2 приложения.

**НЕЙРОН, ИСКУССТВЕННАЯ НЕЙРОННАЯ СЕТЬ, ОБУЧЕНИЕ СЕТИ, МОДЕЛЬ СЛОЖНОГО ОБЪЕКТА, ПЕРЕМЕЩЕНИЕ В ПРОСТРАНСТВЕ.**

Объектом разработки является искусственный интеллект на базе нейронной сети, реализующий перемещение модели сложного объекта в пространстве.

Целью данной работы является создание и обучение искусственной нейронной сети, управляющей перемещением модели сложного объекта в двумерном пространстве. Основная задача модели – переместиться как можно дальше влево или вправо от своей начальной координаты. Также необходимо реализовать приложение для создания нейронных сетей путем установки их параметров и конструирования моделей объектов.

В качестве средств разработки использовалась среда Microsoft Visual Studio 2015 Community.

В результате проведенной работы была реализована искусственная нейронная сеть, а также приложение, предоставляющее возможности по моделированию процесса перемещения сложного объекта в пространстве.

В процессе работы проводились исследования предметной области, теоретическое проектирование и разработка отдельных модулей системы.

Подп. и дата	
Инв № дубл.	
Взам. инв. №	
Подп. и дата	
Инв № подл.	

БР-02069964-090301-16-18				
Изм.	Лист	№ докум.	Подп.	Дата
Разраб.		Немчинова	<i>[Подпись]</i>	20.06.18
Проб.		Плотникова	<i>[Подпись]</i>	20.06.18
Н.контр.		Ямашкин	<i>[Подпись]</i>	20.06.18
Утв.		Федосин	<i>[Подпись]</i>	20.06.18

Лит.	Лист	Листов		
	5	63		
МГУ им. Н.П. Огарева, ИЭС, ИВТ, 441				

Искусственный интеллект на базе нейронной сети, реализующий перемещение модели сложного объекта в пространстве

## Содержание

Введение.....	8
1 Основные положения теории нейронных сетей.....	10
1.1 Строение и основные принципы функционирования нервной системы.....	10
1.2 Искусственный нейрон и искусственные нейронные сети.....	11
1.3 Основные принципы обучения искусственных нейронных сетей.....	17
2 Разработка модели сложного объекта.....	19
2.1 Понятие модели объекта.....	19
2.2 Способы организации движения модели.....	20
2.2.1 Неупругие перемещения конечностей.....	21
2.2.2 Упругие перемещения конечностей.....	22
2.3 Особенности применения нейронной сети.....	23
3 Разработка искусственной нейронной сети.....	27
3.1 Постановка задачи .....	27
3.2 Архитектура искусственной нейронной сети.....	27
3.3 Выбор среды разработки.....	30
3.4 Система классов искусственной нейронной сети и модели объекта.....	31
3.5 Алгоритм обучения.....	37
4 Реализация приложения для моделирования процесса перемещения сложного объекта в пространстве.....	43
4.1 Общая информация и цели использования.....	43
4.2 Архитектура приложения.....	43
4.3 Функциональные возможности приложения.....	45
5 Анализ результатов.....	55
5.1 Условия и критерии проведения экспериментов .....	55
5.2 Сравнение эффективности обучения искусственных нейронных сетей с различным количеством нейронов в скрытом слое.....	56

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

						<b>БР-02069964-090301-16-18</b>	Лист
Изм	Лист	№ докум.	Подпись	Дата			6

5.3 Сравнение эффективности обучения искусственных нейронных сетей с различными комбинациями параметров вознаграждения.....	57
5.4 Эффективность работы нейронных сетей, управляющих различными миделями объектов.....	58
Заключение.....	62
Список использованных источников.....	63
Приложение А – Программный код.....	64
Приложение Б – Графический материал.....	93

Инв.№ подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата	<b>БР-02069964-090301-16-18</b>	Лист
						7



## Введение

Искусственный интеллект является одним из передовых направлений современной науки. Достижения данной дисциплины захватывают все сферы человеческой деятельности. Основная цель изучения искусственного интеллекта — научить машины думать подобно тому, как это делает человеческий мозг.

Нейронные сети решают широкий круг задач, каждая из которых непосредственно связана с обучением. К основным областям применения нейронных сетей относят распознавание образов и их классификацию, принятие решений и управление, кластеризацию, прогнозирование, аппроксимацию, оптимизацию, сжатие и анализ данных. В настоящий момент нейронные сети являются ключевым элементом современных систем навигации, используются для распознавания и синтеза речи, обработки изображений, а также для защиты информационных систем от несанкционированного доступа и другого вида атак.

В процессе изучения возможностей искусственного интеллекта одной из сложнейших задач стало обучение нейронной сети осознанному перемещению в пространстве. Когда человек идет по улице, он с легкостью переставляет ноги, поддерживает равновесие с помощью рук, не задумываясь, обходит препятствия. Все это происходит, как нам кажется, автоматически. Однако, это не так. Перемещение человека — это сложный процесс, хорошо просчитанный и четко спланированный нашим мозгом. Нейронные сети решают подобные проблемы пока лишь в упрощенном варианте, но в то же время они могут управлять моделями любых сложных объектов. Ярким примером является разработка компанией Google DeepMind искусственного интеллекта, которому удалось научиться ходить, бегать, прыгать и преодолевать препятствия без каких-либо предварительных рекомендаций [6].

Целью данной работы является создание искусственной нейронной сети,

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата	<b>БР-02069964-090301-16-18</b>	Лист
						8

управляющей перемещением модели сложного объекта в двумерном пространстве. Основная задача модели — переместиться как можно дальше влево или вправо от своей начальной координаты.

В качестве вспомогательных ресурсов необходимо реализовать приложение для создания двумерной модели объекта, а также для редактирования и настройки параметров нейронной сети.

Задачи работы:

- анализ предметной области;
- разработка архитектуры нейронной сети;
- реализация нейронной сети;
- разработка приложения для моделирования процесса перемещения сложного объекта в пространстве.

Инв.№ подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата	<b>БР-02069964-090301-16-18</b>	Лист
						9

# 1 Основные положения теории нейронных сетей

Искусственные нейронные сети представляют собой сложные вычислительные системы с множеством связей между параллельно выполняющимися простыми процессами. По своей структуре они зачастую напоминают строение человеческого мозга, являются его упрощенной моделью. Их основу составляют элементы, функционально похожие на биологический нейрон. Эти элементы выступают в роли преобразователей поступающей на них информации. Взаимодействие преобразователей также имеет сходство с организацией связей между биологическими аналогами, а принципы познания и способность к адаптивному обучению и развитию позволяют провести параллель между работой нейронной сети и мозговой активностью.

## 1.1 Строение и основные принципы функционирования нервной системы

Центральная нервная система человека имеет клеточное строение. Структурной единицей системы является нервная клетка — нейрон (рисунок 1). Основу нейрона, как и любой другой клетки, составляет тело, называемое сомой. Нейроны связаны между собой нервными волокнами, по которым передаются сигналы. От тела нервной клетки отходят два типа отростков нервных волокон — дендриты, которых может быть множество, и единственный аксон. Дендриты — короткие разветвленные отростки — предназначены для приема поступающих с других нейронов импульсов. Аксон представляет собой длинный утолщенный отросток с разветвлением на конце. Каждое ответвление заканчивается синапсом, с помощью которого аксон передает выходные сигналы нейрона на дендриты других нейронов.

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата	БР-02069964-090301-16-18	Лист
						10

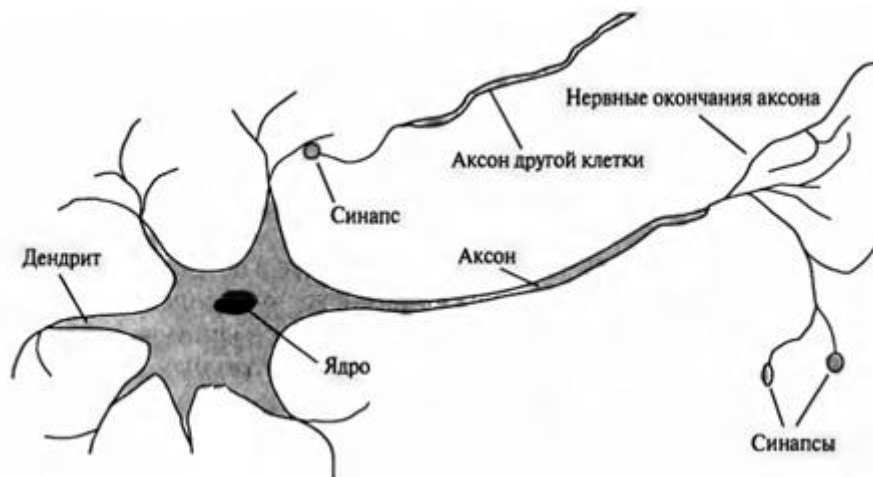


Рисунок 1 — Строение биологического нейрона

Основная функция нейрона состоит в передаче возбуждения из дендритов в аксон. На сигнал аксона оказывают влияние все сигналы, поступающие в сому по дендритам с синапсов, которые могут быть как возбуждающими, так и тормозными. Возбуждающие синапсы способствуют увеличению возбуждения нейрона, тормозные, наоборот, препятствуют дальнейшему распространению импульса. Нейрон генерирует сигнал лишь в том случае, когда суммарное возбуждение окажется выше некоторого предельного значения, иначе реакция на возбуждение будет отсутствовать. Данный процесс в действительности имеет более сложную реализацию, однако большинство нейронных сетей строятся на основе этого свойства [3, 4].

## 1.2 Искусственный нейрон и искусственные нейронные сети

Моделирование человеческого мозга начинается с создания информационной модели отдельной нервной клетки — *искусственного нейрона*. Следующим этапом становится объединение полученных нейронов в единую структуру путем организации связей, характеризующихся определенными параметрами. Результатом этих действий является *искусственная нейронная сеть*, готовая к дальнейшему обучению.

Искусственный нейрон реализует модель поведения своего

Инв. № подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

биологического прототипа (рисунок 2).

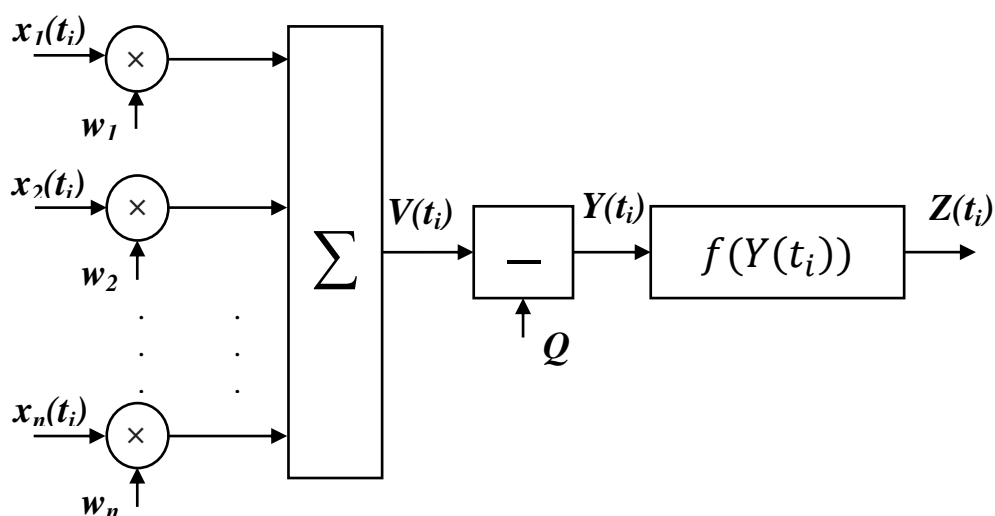


Рисунок 2 — Схема искусственного нейрона

На вход он принимает некоторое множество сигналов, поступающих от других нейронов сети или из внешней среды (в этом случае вход нейрона является входом всей сети). Совокупность всех входных сигналов в момент времени  $t_i$  представляется в виде вектора:

$$X(t_i) = [x_1(t_i), x_2(t_i), \dots, x_n(t_i)]. \quad (1)$$

Значение сигнала каждого входа умножается на соответствующий вес из множества  $W = [w_1, w_2, \dots, w_n]$ , аналогичный синаптической силе. Полученные произведения  $w_j \cdot x_j(t_i)$  складываются, образуя взвешенную сумму сигналов  $V(t_i)$ :

$$V(t_i) = \sum_{j=1}^n w_j x_j(t_i). \quad (2)$$

Аналогично нервным клеткам нейрон может обладать порогом возбуждения  $Q$ . В этом случае уровнем активации нейрона  $Y(t_i)$  будет

Инв. № подл.	
Подп. и дата	
Взам. инв. №	
Инв. № дубл.	
Подп. и дата	

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

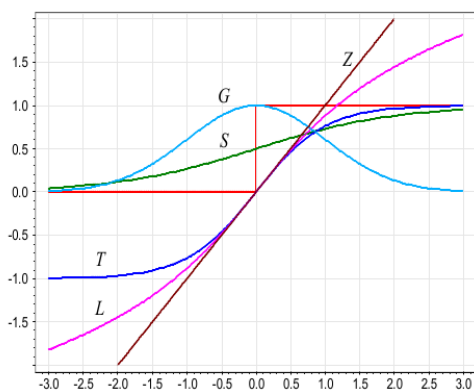
считаться разность между взвешенной суммой входных сигналов  $V(t_i)$  и значением величины порога  $Q$ :

$$Y(t_i) = V(t_i) - Q . \quad (3)$$

Если уровень активации принимает положительное значение ( $Y(t_i) > 0$ ), то нейрон возбуждается, генерируя выходной сигнал  $Z(t_i)$  и передавая его в аксон. Генерация выходного сигнала производится путем преобразования уровня активации  $Y(t_i)$  активационной функцией  $f$ :

$$Z(t_i) = f(Y(t_i)) . \quad (4)$$

Активационная функция  $f$  может представлять собой различные линейные и нелинейные зависимости, каждая из которых по-своему оказывает влияние на поведение нейронной сети (рисунок 3).



$\theta(z) = [z \geq 0]$  ступенчатая функция Хэвисайда;  
 $\sigma(z) = (1 + e^{-z})^{-1}$  сигмоидная функция (S);  
 $\text{th}(z) = 2\sigma(2z) - 1$  гиперболический тангенс (T);  
 $\ln(z + \sqrt{z^2 + 1})$  логарифмическая функция (L);  
 $\exp(-z^2/2)$  гауссовская функция (G);  
 $z$  линейная функция (Z);

Рисунок 3 — Графики активационных функций

Наиболее часто применяются следующие функции:

1 Пороговая функция — простая кусочно-линейная функция. Принимает минимально допустимое значение, если взвешенная сумма входных сигналов меньше порогового возбуждения. В остальных случаях равна максимально

Инв. № подл.	Подп. и дата
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

допустимому значению.

$$f(x) = \begin{cases} 1, & x \geq c, \\ 0, & x < c. \end{cases} \quad (5)$$

2 Линейный порог или гистерезис – кусочно-линейная функция. Она разделена на три участка: два из них соответствуют максимальному и минимальному значениям, на оставшемся участке функция монотонно возрастает.

$$f(x) = \begin{cases} 0, & x \leq -c, \\ 1, & x \geq c, \\ x, & -c < x < c. \end{cases} \quad (6)$$

3 Сигмоидальная функция — монотонно возрастающая дифференцируемая S-образная нелинейная функция с насыщением. Способствует усилению слабых сигналов. Существует две основных разновидности данной функции:

### 3.1 Логистическая функция

$$f(x) = \frac{1}{1+e^{-\alpha x}}, \quad (7)$$

где  $\alpha$  — параметр, влияющий на наклон сигмоидальной функции.

### 3.2 Гиперболический тангенс

$$f(x) = th(x) = \frac{e^{2x}-1}{e^{2x}+1}. \quad (8)$$

Несмотря на то, что искусственный нейрон отражает далеко не все свойства нервной клетки, упуская из вида временные задержки при обработке сигналов и много другое, структуры, сконструированные из таких элементов,

Инв. № подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	Подп. и дата

обладают свойствами, напоминающими биологическую систему [3, 4].

Простейшей математической моделью восприятия информации человеческим мозгом является персептрон. Эта модель была предложена Фрэнком Розенблаттом в 1957 году.

Персептрон представляет собой искусственную нейронную сеть, основанную на генераторах трех типов:

- сенсорных элементов (чувствительные элементы, вырабатывающие сигнал);
- ассоциативных элементов (логические решающие элементы, предназначенные для обработки поступающей информации);
- реагирующих элементов (формируют выходной сигнал в зависимости от значения суммы поступающих на них сигналов).

Рассмотрим основные типы структур искусственных нейронных сетей, основанных на модели персептрона.

### Однослойная нейронная сеть

Упрощенный вариант нейронной сети состоит из совокупности нейронов, организованных в слой. Структура искусственной нейронной сети с одним рабочем слоем представлена на рисунке 4.

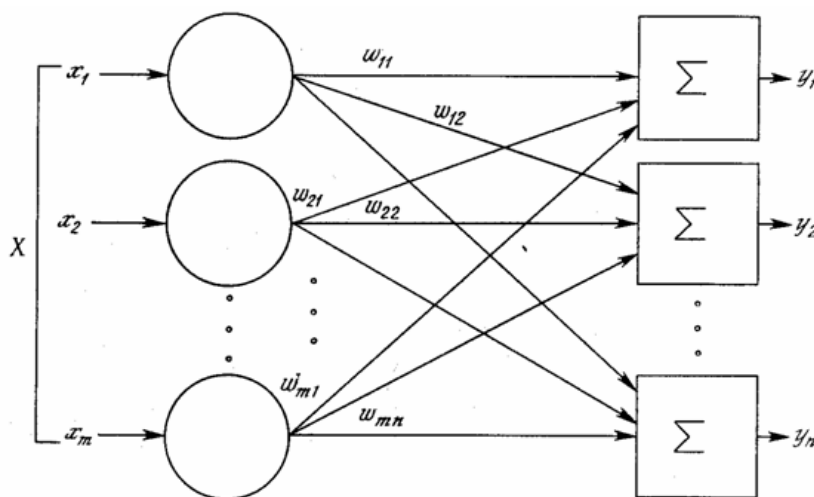


Рисунок 4 – Структура однослойной нейронной сети

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18



В сеть из внешней среды поступает вектор входных данных  $X = [x_1, x_2, \dots, x_n]$ .

Первый слой, обозначенный кругами, не выполняет каких-либо вычислений, а служит лишь для распространения входных сигналов, поэтому данный слой не учитывается при подсчете количества слоев сети.

С распределительного слоя входные сигналы поступают на нейроны скрытого (рабочего) слоя, обозначенного прямоугольниками. Каждый элемент входного вектора соединен с каждым нейроном рабочего слоя ребром, обладающим отдельным весом, характеризующим интенсивность сигнала. Скрытый слой выполняет все необходимые вычисления и генерирует выход нейронной сети в виде вектора  $Y = [y_1, y_2, \dots, y_m]$ .

Реализация однослойных нейронных сетей является наиболее простой, однако функциональные возможности таких сетей ограничены.

### Многослойная нейронная сеть

Структура искусственной нейронной сети с двумя рабочими слоями представлена на рисунке 5.

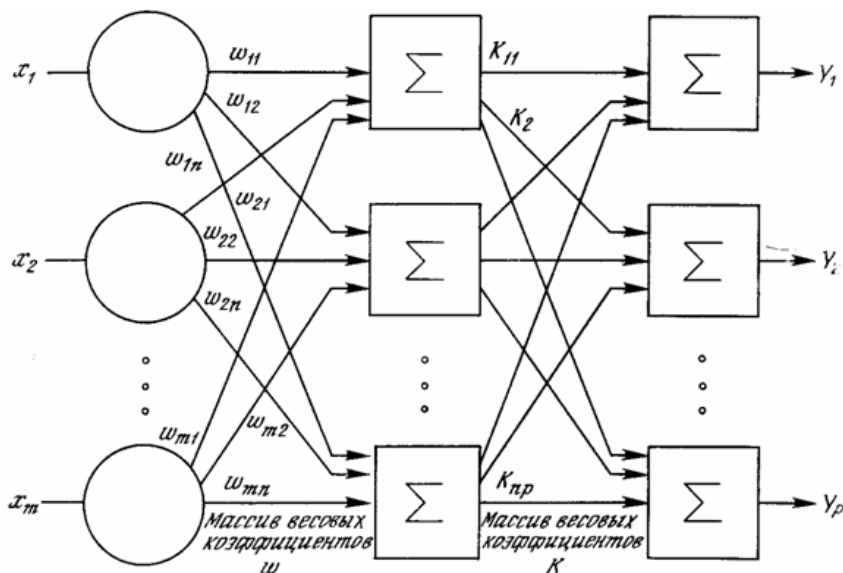


Рисунок 5 — Структура многослойной нейронной сети

Многослойные нейронные сети расширяют функциональные

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	Подп. и дата

возможности однослойных сетей путем копирования слоистых структур определенных отделов мозга. При таком строении выход одного слоя служит входом для другого, передавая ему обработанную информацию.

Слои сети могут отличаться по количеству нейронов и иметь различные активационные функции. Если функции между слоями окажутся линейными, то многослойная нейронная сеть будет эквивалентна однослойной, и прироста вычислительной мощности не произойдет.

### 1.3 Основные принципы обучения искусственных нейронных сетей

Одним из главных свойств нейронной сети, которое позволяет сравнить их деятельность с активностью биологического мозга, является способность к обучению. Именно оно отличает нейронную сеть от классических средств прогнозирования и классификации.

Под термином «обучение» подразумевается корректировка исходных параметров сети таким образом, чтобы она могла решать поставленную задачу. Настраиваемыми параметрами сети выступают веса и пороговые значения возбуждения нейронов (смещения). Целью обучения искусственной нейронной сети является достижение желаемой реакции сети на некоторое множество входных данных.

Корректировка параметром сети производится в соответствии с некоторым алгоритмом. Все алгоритмы обучения можно разделить на две категории: обучение с учителем и без учителя.

При обучении с учителем необходимо для каждого входного вектора наличие целевого вектора (четко установленного желаемого результата). Такие пары векторов носят название *обучающей пары*. Обучение сети производится на некотором множестве таких пар.

Обучение без учителя больше похоже на процесс познания биологического мозга. При таком подходе отсутствуют целевые векторы, а

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата	<b>БР-02069964-090301-16-18</b>	Лист
						17

корректность работы сети оценивается по ее реакции на схожие между собой входные данные. Правильно обученная сеть должна давать одинаковый выход на близкие входные векторы. Процесс обучения распределяет обучающее множество по классам на основе его статических свойств. Выходы нейронной сети при этом преобразуются в некоторую понятную, обусловленную процессом обучения форму.

Инв.№ подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата	<i>БР-02069964-090301-16-18</i>	<i>Лист</i>
						18

## 2 Разработка модели сложного объекта

### 2.1 Понятие модели объекта

Под моделью объекта будем понимать совокупность отрезков, концы которых соединены между собой. В ней должны быть отражены основные части объекта, с помощью которых он перемещается. Соединения отрезков назовем «суставами» модели объекта. Так как перемещение происходит в двумерном пространстве, для однозначного определения положения некоторого отрезка в каждый момент времени зададим его в виде координат точек, являющихся концами этого отрезка. Телом модели объекта будет являться набор жестко связанных отрезков, то есть неподвижных друг относительно друга. Тогда отрезки, способные менять свое положение относительно тела, будут представлять собой конечности модели. Рассмотрим конкретную модель объекта, представленную на рисунке 6.

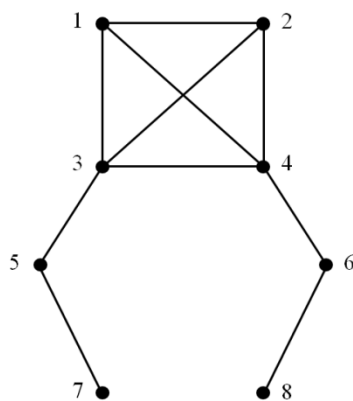


Рисунок 6 — Модель объекта

Исходя из рисунка 6, модель имеет тело, обозначенное четырехугольником 1-2-4-3, и две ярко выраженные конечности (левая конечность — отрезки 3-5 и 5-7, правая — отрезки 4-6, 6-8), с помощью которых объект будет перемещаться.

Следующим шагом необходимо определить, в каких «суставах» и каким

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

Лист

19

образом будут двигаться отрезки, из которых состоит модель. Очевидно, что движения будут совершать отрезки конечностей:

- отрезок 3-5 будет совершать поворот на некоторый угол в «суставе», обозначенном точкой 3;
- отрезок 5-7 поворачивается в точке 5;
- отрезок 4-6 совершает поворот в «суставе» 4;
- отрезок 6-8 — в точке 6.

Обозначим на рисунке 7 углы, на которые смогут поворачиваться конечности объекта.

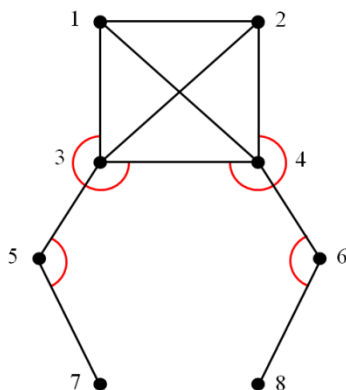


Рисунок 7 — Углы поворота конечностей объекта

## 2.2 Способы организации движения модели

В широком смысле под перемещение понимается изменение положения тела в пространстве с течением времени относительно выбранной системы отсчета. Однако многое зависит от того, как именно будет двигаться объект: будет ли он катиться или упругими прыжками стремиться к цели. Поэтому необходимо определить способ передвижения объекта, который напрямую зависит от свойств его конечностей.

Обозначим два способа организации движения конечностей модели:

- неупругое перемещение, при котором конечности фиксируют свое

Инв. № подл.	
Подп. и дата	
Взам. инв. №	
Инв. № дубл.	
Подп. и дата	

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

положение после совершённого действия;

- упругое перемещение, использующее пружинящий эффект мышц.

Рассмотрим каждый из них более детально.

### 2.2.1 Неупругие перемещения конечностей

Данный способ характеризуется фиксацией отрезка конечности в некотором положении, в которое он пришел после совершения очередного действия. Изначально необходимо определить для исходной модели, на какие углы смогут поворачиваться отрезки в «суставах», то есть требуется указать крайние положения отрезков. Для того чтобы ограничения были корректны в любом положении объекта относительно системы координат, их следует привязать к неподвижным отрезкам модели, а именно к отрезкам, формирующим тело. На рисунке 8 пунктирными линиями указаны предельные положения для каждого из отрезков конечностей объекта.

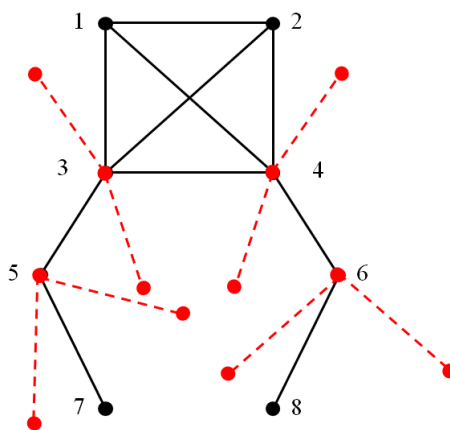


Рисунок 8 — Диапазоны углов поворота конечностей

После поворота конкретного отрезка на определенный угол в доступном ему диапазоне, его положение фиксируется в ожидании дальнейшего действия или реакции окружающей среды. Части конечности, связанные «суставами» с

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

тем отрезком, что совершил поворот в данный момент, также изменяют свое положение, если это необходимо. На текущем шаге эти части считаются жестко связанными с ним.

### 2.2.2 Упругие перемещения конечностей

Этот способ в качестве основных двигателей использует «мышцы», представленные в виде пружин [6]. Для подвижных отрезков, как и в предыдущем случае, устанавливается угол поворота. Затем для каждого «сустава», в котором производится некоторый поворот, определяется соответствующая ему «мышца», обозначаемая на рисунке в виде отрезка, лежащего против угла поворота с вершиной в этом «суставе». Крепление «мышц» осуществляется в точки, лежащие на отрезках. На рисунке 9 представлена модель с указанием углов поворота отрезков. Рисунок 10 демонстрирует «мышцы» (голубым цветом) полученной модели объекта.

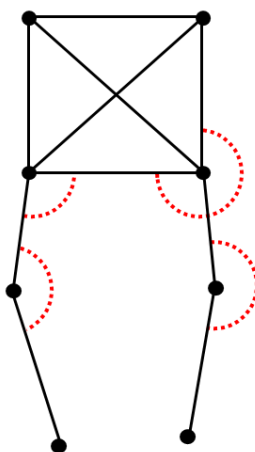


Рисунок 9 — Модель объекта с указанием углов поворота отрезков конечностей

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

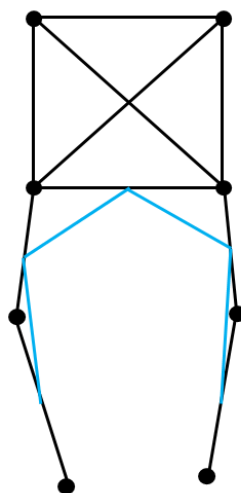


Рисунок 10 — Расположение «мышц» объекта

Модель использует два вида «мышц»: «умные» и «немые». Первые «мышцы» инициируют действие модели. Они изменяют свою длину в соответствии с данными, полученными от нейронной сети. Вторые всего лишь реагируют на произошедшие изменения, влияющие на их состояние. При этом все «мышцы» не остаются в четко фиксированном положении, а реагируют упругими сжатиями и растяжениями подобно пружине.

### 2.3 Особенности применения нейронной сети

В рассмотренных способах перемещения для управления действиями модели объекта будут использоваться нейронные сети, основным отличием которых станут данные, подаваемые на входы сетей, а также результаты, полученные после обработки этих данных.

Применительно к неупругой модели движения целесообразно предоставить нейронной сети информацию об углах поворота конечностей объекта и координатах концов отрезков в текущий момент времени. Однако нейронная сеть, которая будет управлять движением конечностей модели, не сможет повернуть некоторый отрезок на произвольный угол. Ей необходим ряд

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18



состояний, для каждого из которых она будет давать оценку выгоды этого состояния в текущий момент времени. Для удовлетворения потребностей сети разделим диапазон возможных углов поворота для некоторого отрезка на фиксированное количество частей. Определим как действия модели повороты отрезков по часовой стрелке и против часовой стрелки на угол, равный полученной при делении части диапазона. Состояниями при этом будут все возможные положения конечностей модели, количество которых равно произведению чисел частей, на которые разделили каждый из диапазонов. На рисунке 11 приведена схема разбиения диапазонов допустимых углов поворота отрезков на конечное число частей. Красными пунктирными линиями изображены границы частей основного диапазона. При достаточно большом количестве таких частей объект будет двигаться более плавно.

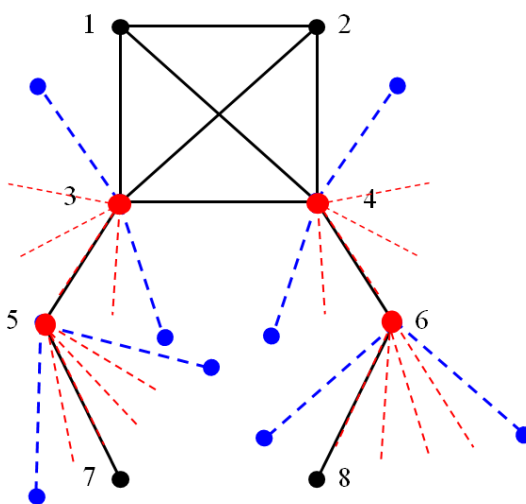


Рисунок 11 — Разбиение диапазонов углов поворота отрезков на конечное число одинаковых частей

Каким же образом будет перемещаться полученная модель? В каждый момент времени нейронная сеть, оценив предыдущие действия и состояние окружающей среды, делает предположение о дальнейшем действии. Модель выполняет полученное от нейронной сети указание, то есть поворачивает один

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

из отрезков конечностей на некоторый определенный заранее угол. После чего производятся получение данных о новом состоянии объекта. Также производится проверка на устойчивость модели в новом состоянии. Если модель начинает падать, то рассчитывается угол поворота тела объекта в сторону падения и соответствующее ее смещение на часть этого угла, определенную для единичного промежутка времени. Когда вычислено итоговое положение модели в пространстве после всех преобразований, данные могут быть переданы на вход нейронной сети для генерации следующего действия.

В случае модели упругого перемещения за счет сжатий и растяжений «мышц» движение происходит путем отталкивания объекта от поверхности, то есть прыжками, поэтому в некоторые моменты времени он находится в состоянии полета. Перемещение модели представлено на рисунке 12.

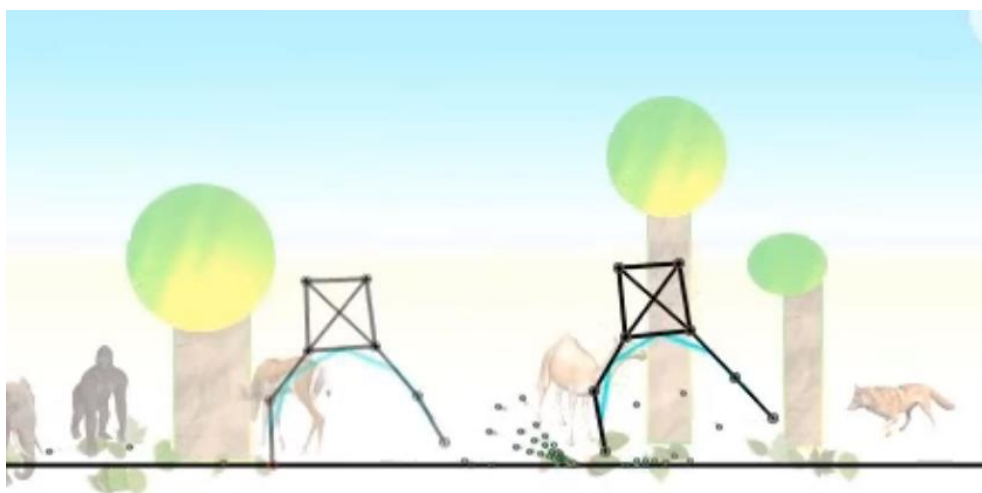


Рисунок 12 — Перемещение модели объекта

Для описания текущего состояния модели необходимо учитывать ряд дополнительных параметров, передаваемых на входы нейронной сети, таких как скорость перемещения, давление касания точек, скорость вращения объектов. Эти данные однозначно характеризуют модель в каждый момент времени. После обработки поданной информации нейронной сетью объект получает от нее значения длин «умных мышц» и изменяет их состояние,

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

сжимаемая или растягиваемая последние. В зависимости от того, как изменилось положение «умных мышц» и соответствующих отрезков, реагируют «немые мышцы». Таким образом производится упругое отталкивание модели объекта от поверхности, по которой он движется, или его приземление. Как и в первом случае, все данные о новом состоянии модели передаются на входы нейронной сети.

Инв.№ подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата	<b>БР-02069964-090301-16-18</b>	Лист
						26

### 3 Разработка искусственной нейронной сети

#### 3.1 Постановка задачи

Целью данной работы является создание искусственной нейронной сети, управляющей перемещением модели сложного объекта в двумерном пространстве. В данной работе будет рассмотрен способ перемещения модели с помощью неупругого движения конечностей.

Требуется обучить нейронную сеть таким образом, чтобы она могла производить оценку полезности совершения каждого допустимого действия. На основе полученных результатов модель должна совершать наиболее выгодное для текущего состояния действие. Под действием будем понимать поворот одной из частей конечностей модели объекта. За единицу времени модель может совершить поворот только одной части любой из конечностей. Задача модели — переместиться как можно дальше влево или вправо от своей начальной координаты.

Также необходимо реализовать приложение для создания двумерной модели объекта и редактирования и настройки параметров нейронной сети.

#### 3.2 Архитектура искусственной нейронной сети

Искусственная нейронная сеть будет представлять собой многослойный персептрон. Параметры архитектуры сети приведены в таблице 1.

Таблица 1 — Параметры архитектуры нейронной сети

Наименование параметра	Значение параметра
Количество нейронов входного слоя	Количество суставов модели объекта, умноженное на количество параметров, характеризующих каждый

Инв. № подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	

Изм	Лист	№ докум.	Подпись	Дата	<b>БР-02069964-090301-16-18</b>	Лист
						27

	сустав
--	--------

Окончание таблицы 1

	(используются координаты точек в двумерном пространстве)
Количество скрытых слоев	Два
Количество нейронов скрытых слоев	32 – 96
Функция активации скрытых слоев	Тангенс гиперболический
Количество нейронов выходного слоя	Суммарное количество частей подвижных конечностей, умноженное на количество действий, которые они могут совершать (каждая часть конечности может совершить два действия: поворот на единичный угол по часовой стрелке и против часовой стрелки)
Функция активации нейронов выходного слоя	Линейная

На входы нейронная сеть должна получить набор параметров, однозначно характеризующих положение модели объекта в двумерном пространстве. В качестве таких параметров были выбраны координаты точек, соответствующих суставам модели, в двумерном пространстве. Каждая точка описывается двумя координатами — значениями по осям абсцисс и ординат.

Результатом работы нейронной сети является оценка полезности совершения моделью каждого допустимого действия. На нейроны выходного слоя поступает информация, соответствующая элементарному действию (поворот на единичный угол по часовой стрелке или против часовой стрелки)

Инв. № подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	

частей подвижных конечностей.

Архитектура искусственной нейронной сети представлена на рисунке 13.

Рассмотрим параметры нейронной сети для конкретной модели объекта, представленной на рисунке 14. Черными кругами отмечены суставы модели, красным пронумерованы отрезки.

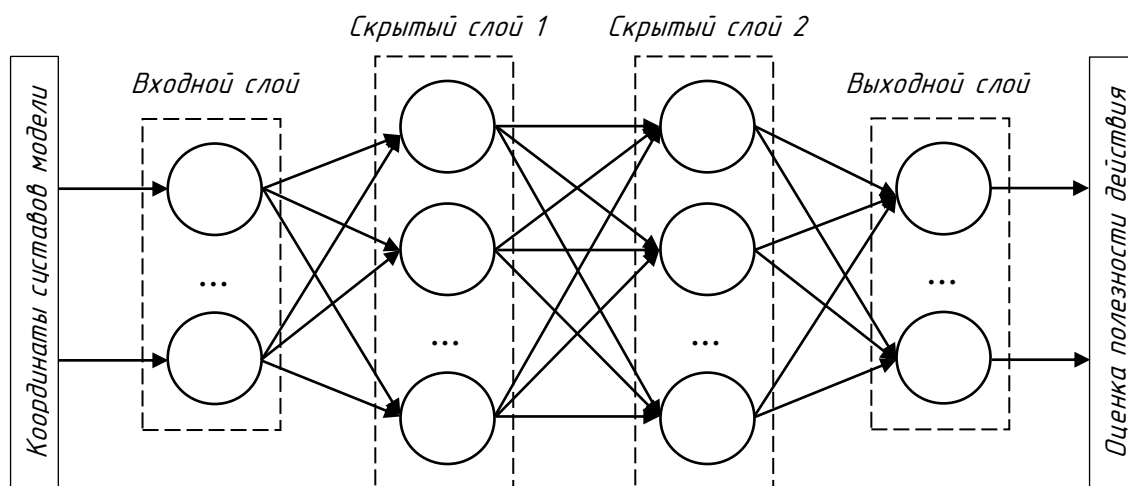


Рисунок 13 – Архитектура искусственной нейронной сети

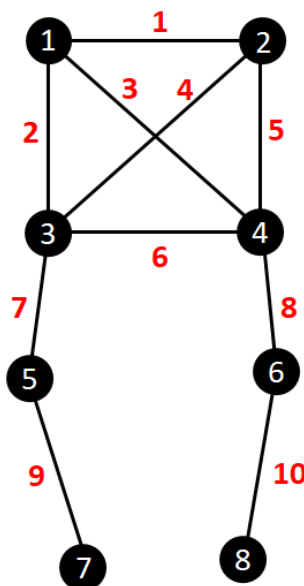


Рисунок 14 – Модель объекта

Модель имеет 8 суставов, каждому из которых соответствуют две

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

координаты. Следовательно, у нейронной сети, управляющей перемещением данной модели, будет 16 входов.

Пусть подвижными будут являться две конечности: первая состоит из отрезков 7 и 9, вторая — из отрезков 9 и 10. Таким образом, суммарное количество отрезков равно 4. С учетом возможных направлений поворота отрезков, нейронная сеть будет иметь 8 выходов.

Параметры архитектура нейронной сети для данной модели представлены в таблице 2.

Таблица 2 — Параметры архитектуры нейронной сети для конкретной модели объекта

Наименование параметра	Значение параметра
Количество нейронов входного слоя	8 суставов · 2 = 16
Количество скрытых слоев	2
Количество нейронов скрытых слоев	32
Функция активации скрытых слоев	Тангенс гиперболический
Количество нейронов выходного слоя	4 отрезка · 2 = 8
Функция активации нейронов выходного слоя	Линейная

### 3.3 Выбор среды разработки

Для реализации выбрана среда разработки Microsoft Visual Studio 2015 Community.

В качестве языка программирования использовался C++. Выбор данного языка обусловлен его высокой вычислительной производительностью. Гибкость и компактность языка способствуют созданию понятного, легко воспринимаемого кода. Язык обладает обширным кругом инструментов для программирования и отладки. Еще одним преимуществом C++ является

Инв. № подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	Подп. и дата

наличие богатой стандартной библиотеки STL (Standard Template Library), которая включает набор распространённых контейнеров и алгоритмов, средств ввода-вывода, регулярных выражений, поддержку многопоточности и других возможностей. C++ сочетает в себе свойства как высокоуровневых, так и низкоуровневых языков.

Пользовательский интерфейс создавался на основе CLR-проекта. CLR(Common Language Runtime) — исполняющая среда, в которой компилируются программы, написанные на .NET-совместимых языках программирования. Является спецификацией CLI (Common Language Infrastructure, общезыковая инфраструктура) [8].

Причиной для выбора Microsoft Visual Studio 2015 Community средой для разработки стало наличие широких возможностей по отладке программного продукта. Также Microsoft Visual Studio обладает удобным и понятным интерфейсом, а механизм подстановки определений элементов позволяет очень легко и быстро создавать многомодульные проекты.

Реализация отрисовки изображений будет производиться средствами графической библиотеки OpenGL. OpenGL (Open Graphics Library) — спецификация, определяющая платформонезависимый программный интерфейс для написания приложений, использующих двумерную и трёхмерную компьютерную графику. Библиотека включает более 300 функций для рисования сложных сцен из простых примитивов.

### 3.4 Система классов искусственной нейронной сети и модели объекта

Структура классов нейронной сети и модели объекта может быть представлена в виде диаграммы (рисунок 15) [1]. Программная реализация классов приведена в приложении А.

Класс **NeuroNet** реализует структуру искусственной нейронной сети, основанную на многослойном персептроне. Класс включает свойства и методы,

Инв.№ подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата	<b>БР-02069964-090301-16-18</b>	Лист
						31



необходимые для функционирования и обучения персептрона:

- num\_layers — количество слоев нейронной сети;
- layers — объекты класса Layers, характеризующие каждый слой нейронной сети, включая входной (распределительный);

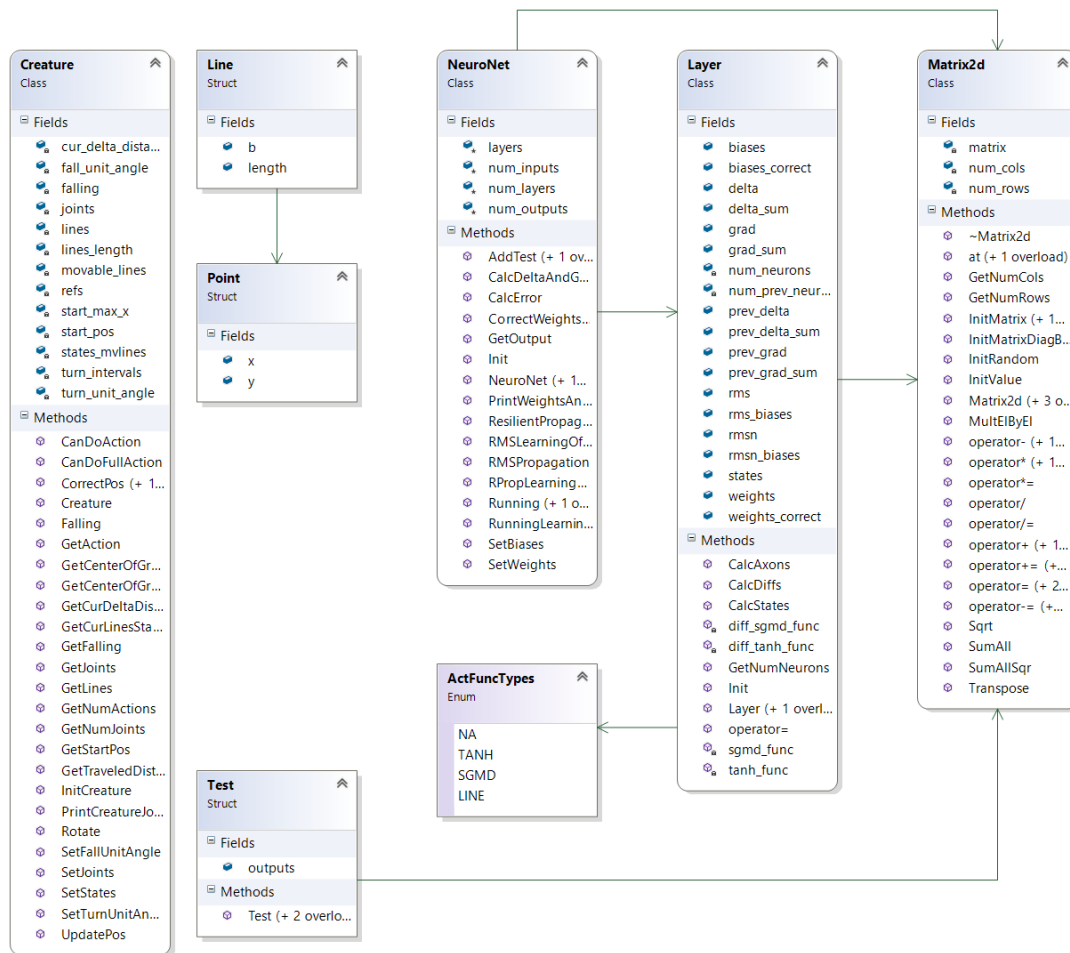


Рисунок 15 — Диаграмма классов нейронной сети и модели объекта

- num\_inputs — количество входов нейронной сети;
- num\_outputs — количество выходов нейронной сети;
- outputs — объект класса Matrix2d, содержащий значения выходов нейронной сети;
- NeuroNet() — конструктор по умолчанию класса NeuroNet;
- NeuroNet(int ninputs, int nlayers, vector<int> nlnurons, vector<ActFuncTypes> \_aft, int noutputs) — конструктор с параметрами;
- Init(int ninputs, int nlayers, vector<int> nlnurons, vector<ActFuncTypes>

Инв. № подл.	
Подп. и дата	
Взам. инв. №	
Инв. № дубл.	
Подп. и дата	

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

\_aft, int noutputs, vector<Matrix2d> \_biases, vector<Matrix2d> \_weights) — метод, производящий инициализацию объекта класса в соответствии с набором переданных параметров;

- SetWeights(vector<Matrix2d> \_w) — установка значений весовых коэффициентов в соответствии с переданными параметрами;

- SetBiases(vector<Matrix2d> \_b) — установка значений смещений в соответствии с переданными параметрами;

- AddTest — метод, добавляющий очередную обучающую пару к обучающей выборке;

- Running — метод, для реализации процесса прямого распространения сигнала по нейронной сети с заданным значением входных данных;

- RMSPropagation() — метод, осуществляющий корректировку весовых коэффициентов и значений смещений нейронов многослойного персептрона в соответствии с алгоритмом RMS Propagation;

- RMSLearningOffline — метод, осуществляющий обучение нейронной сети с помощью алгоритма RMS Propagation;

- CalcDeltaAndGrad(Test& test) — подсчет значений градиента каждого слоя нейронной сети;

- CalcError(Test& test) — расчет значения ошибки выхода нейронной сети;

- CorrectWeightsAndBiases(vector<Matrix2d>&\_gradient,vector<Matrix2d>&\_delta) — метод, осуществляющий корректировку весовых коэффициентов и значений смещений в соответствии с заданными значениями;

- GetOutput() — вывод значений выходов нейронной сети;

- PrintWeightsAndBiases(ostream& fout, bool print\_null) – метод сохранения значений весовых коэффициентов и смещений в файл;

- operator << — оператор вывода параметров нейронной сети.

Класс **Layer** описывает слой нейронной сети с заданным количеством нейронов и указанной функцией активации. Класс содержит свойства и методы

Инв.№ подл.	Взам. инв. №	Инв. № дубл.	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата	<b>БР-02069964-090301-16-18</b>	Лист
						33

для расчета состояний и аксонов нейронов слоя:

- num\_neurons — количество нейронов слоя;
- num\_prev\_neurons — количество нейронов предыдущего слоя нейронной сети;
- weights — матрица значений весовых коэффициентов;
- states — матрица значений состояний нейронов;
- biases — матрица значений смещений;
- axons — матрица значений аксонов нейронов;
- activation\_func — активационная функция нейронов слоя;
- sgmd\_func(Matrix2d& \_states) — расчет сигмоиды для элементов матрицы состояний;
- tanh\_func(Matrix2d& \_states) — расчет гиперболического тангенса для элементов матрицы состояний;
- diff\_sgmd\_func(Matrix2d& \_axons) — расчет производной сигмоиды для элементов матрицы;
- diff\_tanh\_func(Matrix2d& \_axons) — расчет производной гиперболического тангенса для элементов матрицы;
- void Init(int nneurons, int nprevneurons, ActFuncTypes act\_func, Matrix2d \_biases, Matrix2d \_weights) — инициализация объекта класса Layer в соответствии с набором параметров;
- CalcStates(Layer& prev\_layer) — метод расчета состояний нейронов слоя;
- CalcAxons() — метод расчета аксонов нейронов слоя.

Класс **Matrix2d** представляет собой описание двумерных матриц, содержит средства работы с ними. Свойства и методы класса:

- num\_rows — количество строк матрицы;
- num\_cols — количество столбцов матрицы;
- matrix — двумерная матрица;
- InitMatrix(int nrow, int ncol, double val) — инициализация объекта класса

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.

Matrix2d в соответствии с заданными параметрами;

- InitMatrixDiagByOne(int nrow, int ncol) — инициализация матрицы, главная диагональ которой заполнена единицами, а остальные элементы являются нулями;

- InitRandom(double minval, double maxval) — инициализация матрицы случайными значениями в заданном интервале;

- InitValue(double val) — инициализация элементов матрицы заданным значением;

- GetNumRows() — вывод количества строк матрицы;

- GetNumCols() — вывод количества столбцов матрицы;

- Transpose() — транспонирование матрицы;

- MultElByEl(Matrix2d \_m) – поэлементное умножение матриц;

- SumAll() — сложение всех элементов матрицы;

- SumAllSqr() — сложение квадратов всех элементов матрицы;

- operator +, operator -, operator \*, operator / — операторы выполнения арифметических действий.

Класс **Creature** описывает модель сложного объекта, отражая все необходимые характеристики. Класс снабжен функционалом для работы с параметрами модели:

- fall\_unit\_angle — единичный угол поворота при падении модели;

- turn\_unit\_angle — единичный угол поворота части конечности модели;

- double start\_pos — координата начального положения центра тяжести модели по оси абсцисс;

- joints — массив, содержащий описание суставов модели (координаты точек двумерного пространства);

- lines — массив, содержащий информацию об отрезках, из которых состоит модель (каждый отрезок описывается парой чисел – номер точек концов отрезка из массива joints);

- lines\_length — соответствующие длины отрезков массива lines;

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата	БР-02069964-090301-16-18	Лист
						35

- movable\_lines — массив, содержащий описание подвижных отрезков (содержит пары чисел: номер отрезка из массива lines и номер сустава, в котором он поворачивается, из массива joints);

- turn\_intervals — массив значений интервалов допустимых углов поворота каждого отрезка модели;

- states\_mvlines — массив состояний подвижных отрезков;

- refs — массив зависимостей отрезков;

- InitCreature(vector<pair<double, double>> \_joints, vector<pair<int, int>> \_lines, vector<pair<int, int>> \_mvlines, vector<pair<double, double>> \_turnint, vector<pair<int, int>> states, vector<vector<int>> \_refs, vector<int> \_head\_points)

— инициализация объекта класса Creature в соответствии с заданными значениями;

- GetNumActions() — вывод общего количества элементарных действий, которые может совершить модель;

- GetCenterOfGravityX() — вывод координаты центра тяжести модели по оси абсцисс;

- GetCenterOfGravityY() — вывод координаты центра тяжести модели по оси ординат;

- GetCurDeltaDistance() — вывод смещения центра тяжести модели по оси абсцисс от своего начально положения;

- GetAction(int action\_num) — определение действия на основе результатов работы нейронной сети;

- CanDoAction(int action) — проверка возможности выполнения указанного действия;

- Rotate(int line, int tdir) — поворот отрезка модели на единичный угол по часовой стрелке или против нее;

- Falling() — корректировка координат точек суставов модели с учетом ее падения;

- UpdatePos(int action\_num) — совершение указанного действия моделью;

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	Подп. и дата

- PrintCreatureJoints() — сохранение в файл текущих координат точек суставов модели.

### 3.5 Алгоритм обучения

Для обучения искусственной нейронной сети с выбранной архитектурой использовался алгоритм RMS Propagation совместно с алгоритмом Q-Learning [5, 9].

#### RMS Propagation

RMS Propagation (Root Mean Square Propagation) основан на классическом алгоритме обратного распространения ошибки. Его отличие состоит в том, что часто обновляемые веса корректируются меньше остальных. Это достигается за счет вычисления усредненных по истории значений градиента и квадрата градиента (экспоненциальное скользящее среднее).

Экспоненциальное скользящее среднее является частным случаем взвешенного скользящего среднего и рассчитывается по следующей формуле:

$$E[x]_t = \gamma \cdot E[x]_{t-1} + (1 - \gamma) \cdot x, \quad (9)$$

где  $\gamma$  — коэффициент сохранения ( $0 \leq \gamma \leq 1$ ).

Приближение значения  $\gamma$  к единице говорит о том, что история значений будет влиять на результат больше, чем текущее значение  $x$ .

С учетом формулы 9 расчет значений корректировки весовых коэффициентов искусственной нейронной сети будет производиться по формуле:

$$W_t = W_{t-1} - \frac{\eta}{\sqrt{E[g^2]_t - E[g]_t^2 + \varepsilon}} g_t, \quad (10)$$

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

Изм	Лист	№ докум.	Подпись	Дата	<b>БР-02069964-090301-16-18</b>	Лист
						37

где  $W_t$  — значение веса на текущем шаге;

$W_t$  — значение веса на предыдущем шаге;

$\eta$  — значение скорости обучения;

$g_t$  — значение градиента на текущем шаге;

$E[g]_t$  — скользящее среднее значения градиента на текущем шаге;

$E[g^2]_t$  — скользящее среднее значение квадрата градиента на текущем шаге,

$\varepsilon$  — константа, задающая значение погрешности. Данная формула представляет собой модификацию стандартного алгоритма RMS Propagation, заключающуюся в применении значения корректировки квадрата скользящего среднего градиента совместно с бегущим средним квадрата градиента.

Для вычисления значений градиентов используется метод обратного распространения ошибки (Backpropagation) [2]. Согласно алгоритму расчет значения градиента производится по следующей формуле:

$$g_t(W) = \frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} \cdot \frac{dy_j}{ds_j} \cdot \frac{\partial s_j}{\partial w_{ij}}, \quad (11)$$

где  $w_{ij}$  — весовой коэффициент синаптической связи, соединяющей  $i$ -ый нейрон слоя  $n-1$  с  $j$ -ым нейроном слоя  $n$ ;

$y_j$  — значение выходного сигнала  $j$ -ого нейрона;

$s_j$  — взвешенная сумма входных сигналов  $j$ -ого нейрона (аргумент активационной функции).

Множитель  $dy_j/ds_j$  является производной активационной функции по ее аргументу, тогда производная этой функция должна быть определена на всей оси абсцисс. Для гиперболического тангенса она принимает вид:

$$\frac{dy}{ds} = 1 - th^2(s). \quad (12)$$

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.

Множитель  $\partial s_j / \partial w_{ij}$  равен значению выходного сигнала предыдущего слоя  $y_i^{(n-1)}$ .

Первый множитель в формуле (11) можно разложить следующим образом:

$$\frac{\partial E}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} \cdot \frac{dy_k}{ds_k} \cdot \frac{\partial s_k}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} \cdot \frac{dy_k}{ds_k} \cdot w_{jk}^{(n+1)}. \quad (13)$$

Суммирование по  $k$  выполняется среди нейронов слоя  $n+1$ .

Введем новую переменную:

$$\delta_j^{(n)} = \frac{\partial E}{\partial y_j} \cdot \frac{dy_j}{ds_j}. \quad (14)$$

Тогда расчет величин  $\delta_j^{(n)}$  слоя  $n$  из величин  $\delta_k^{(n+1)}$  следующего за ним слоя  $n+1$  можно реализовать по следующей рекурсивной формуле:

$$\delta_j^{(n)} = \left[ \sum_k \delta_k^{(n+1)} \cdot w_{jk}^{(n+1)} \right] \cdot \frac{dy_j}{ds_j}. \quad (15)$$

Для выходного слоя эта формула примет вид:

$$\delta_l^{(N)} = (y_l^{(N)} - d_l) \cdot \frac{dy_l}{ds_l}, \quad (16)$$

где  $d_l$  — желаемое значение выходного сигнала нейрона  $l$ .

С учетом формул (15) и (16) по формуле (11) рассчитываются значения градиента.

### Q-Learning

Данный метод применяется в искусственном интеллекте при агентном подходе, когда нельзя четко определить желаемый результат. Он основан на

Инв. № подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	



введении функции  $Q$ , отражающей полезность каждого возможного действия агента для его текущего состояния. Значения функции формируются в зависимости от вознаграждения. Благодаря этим значениям агент не случайно выбирает стратегию поведения, а учитывает опыт предыдущих действий.

Перед началом обучения формируется множество возможных состояний  $S$  агента, а также набор допустимых для него действий  $A$ . Выполнение некоторого действия  $a \in A$  приводит к переходу агента из одного состояния в другое. В результате выполнения действия агенту начисляется вознаграждение. Задачей агента является максимизация суммарной награды.

Процесс обучения представляет собой итерационное уточнение значения функции  $Q$ . На каждом шаге рассчитывается значение оценки полезности совершения каждого допустимого действия для текущего состояния агента путем запуска нейронной сети. Затем производится корректировка оценки полезности последнего действия по следующей формуле:

$$Q(State_t, Action_{t-1}) = Reward_t + \gamma \cdot \max_{Action_t} Q(State_t, Action_t), \quad (17)$$

где  $Reward$  — вознаграждение, полученное агентом за предыдущее действие;

$\gamma$  — коэффициент, отражающий степень доверия новому действию ( $0 \leq \gamma \leq 1$ ), он показывает, что для агента имеет большую ценность: сиюминутные награды или будущие. Чем ближе значение коэффициента к единице, тем быстрее он забывает предыдущие действия.

После всех расчетов полученные значения сохраняются в тренировочный набор.

Вознаграждение за совершенное действие учитывает следующие параметры:

- 1) расстояние, пройденное моделью от ее начального положения:

$$AllDist_t = |CurrentPositionX_t - StartPositionX|, \quad (18)$$

Инв.№ подл.	Взам. инв. №	Инв. № дубл.	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата	<b>БР-02069964-090301-16-18</b>	Лист
						40

где  $CurrentPositionX_t$  — значение координаты центра тяжести модели по оси абсцисс в текущий момент времени;

$StartPositionX$  — значение координаты центра тяжести модели по оси абсцисс в начальный момент времени;

2) расстояние, пройденное моделью за последнюю единицу времени:

$$PrevStepDist_t = |AllDist_t - AllDist_{t-1}|; \quad (19)$$

3) штраф за приближение центра тяжести модели к «земле»:

$$CenterOfGravity_t = - \frac{k_1}{CenterOfGravityY_t}, \quad (20)$$

где  $CenterOfGravity_t$  — координата центра тяжести модели по оси ординат;

$k_1$  — коэффициент, показывающий силу влияния данного параметра на общее значение вознаграждения;

4) штраф за падение модели:

$$Falling_t = - \frac{k_2}{MinDistToSupportPoint_t}, \quad (21)$$

где  $MinDistToSupportPoint_t$  — минимальное значение среди расстояний от центра тяжести модели до точек опоры, рассчитанное по оси абсцисс;

$k_2$  — коэффициент, показывающий силу влияния данного параметра на общее значение вознаграждения;

5) штраф за приближение верхней точки («головы») модели к «земле»:

$$Head_t = - \frac{k_3}{MinHeadY_t}, \quad (22)$$

Инв. № подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	
Инв. № подл.	

где  $MinHeadY_t$  — минимальное значение среди координат точек «головы» модели по оси ординат;

$k_3$  — коэффициент, показывающий силу влияния данного параметра на общее значение вознаграждения.

Формула для расчета вознаграждения представляет собой комбинацию выше указанных параметров и может быть записана в следующем виде:

$$Reward_t = \alpha_1 * AllDist_t + \alpha_2 * PrevStepDist_t + \alpha_3 * CenterOfGravity_t + \alpha_4 * Falling_t + \alpha_5 * Head_t , \quad (23)$$

где  $\alpha_i (i = \overline{1,5})$  — коэффициент, принимающий значения 0 или 1 и показывающий, будет ли учитываться текущий параметр при расчете вознаграждения.

### Совмещенный алгоритм

С помощью алгоритма Q-Learning производится формирование желаемых выходных данных для искусственной нейронной сети (значения рассчитываются по формулам (17) – (24)). Полученные данные помещаются в тренировочный набор, после чего с помощью алгоритма RMS Propagation происходит обучение сети. На выходе нейронной сети формируется оценка целесообразности поворота отрезков модели. Ошибка выхода нейронной сети вычисляется по следующей формуле:

$$\frac{1}{2} [Reward + \gamma \cdot \max_{Action_t} Q(State_t, Action_t) - Q(State_{t-1}, Action_{t-1})]^2. \quad (24)$$

Обучение продолжается в течение нескольких эпох до тех пор или до тех пор, пока значение ошибки не станет меньше некоторой допустимой погрешности.

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата	<b>БР-02069964-090301-16-18</b>	Лист
						42

## 4 Реализация приложения для моделирования процесса перемещения сложного объекта в пространстве

### 4.1 Общая информация и цели использования

Приложение для моделирования процесса перемещения сложного объекта в пространстве — программный продукт, реализованный на языке C++. Он предоставляет средства для создания нейронных сетей путем установки их параметров, конструирования моделей сложных объектов, а также запуска процесса моделирования. Для отображения пользовательского интерфейса используется технология Windows Forms [8].

### 4.2 Архитектура приложения

Приложение представляет собой систему, состоящую из четырех логических модулей:

1 Model Creator — пользовательский интерфейс для создания моделей объектов и редактирования их параметров;

2 Object Model — модель сложного объекта, инициализированная параметрами, заданными с помощью интерфейса Model Creator;

3 NeuroNet Editor — пользовательский интерфейс, позволяющий задавать параметры нейронной сети;

4 Neural Network — модуль, реализующий нейронную сеть, управляющую поведением модели объекта.

Схема взаимодействия модулей приложения представлена на рисунке 16.

Взаимодействие модулей происходит следующим образом.

Средствами интерфейса Model Creator создается модель некоторого объекта, задаются значения ее характеристик. Результатом работы модуля является структура данных, полностью описывающая модель и набор ее

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

Изм	Лист	№ докум.	Подпись	Дата	<b>БР-02069964-090301-16-18</b>	Лист
						43

допустимы действий. Эти данные передаются в модуль Object Model, производя инициализацию объекта.

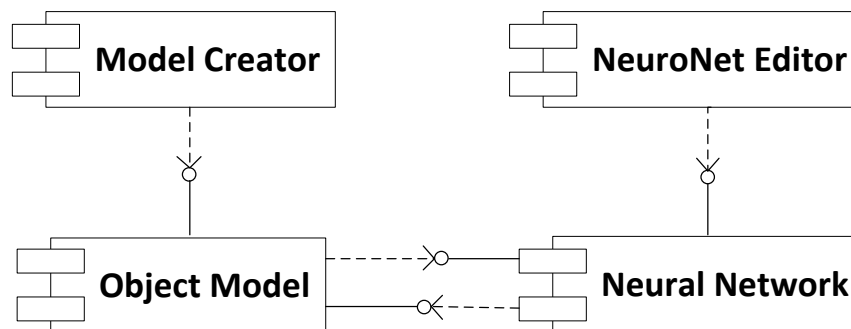


Рисунок 16 — Схема взаимодействия модулей приложения

В интерфейсе NeuroNet Editor происходит настройка параметров нейронной сети. После чего указанная информация отправляется в модуль Neural Network, реализующий нейронную сеть для управления моделью.

Нейронная сеть также должна получить исходные данные от модуля Object Model для формирования своей структуры.

Дальнейшее взаимодействие между модулями Object Model и Neural Network заключается в обмене данными о текущем состоянии модели. Neural Network получает однозначную характеристику модели объекта в некоторый момент времени, возвращая взамен указания на выполнения конкретного действия.

### 4.3 Функциональные возможности приложения

Приложение предоставляет ряд возможностей для моделирования перемещения сложного объекта, управляемого нейронной сетью, в двумерном пространстве. Основу функционала составляют следующие действия:

- 1 Создание модели сложного объекта;
- 2 Создание искусственной нейронной сети и настройка ее параметров;
- 3 Запуск искусственной нейронной сети на обучение;

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

#### 4 Запуск обученной искусственной нейронной сети.

При запуске приложение открывается главное окно, которое содержит список всех сохраненных нейронных сетей (рисунок 17).

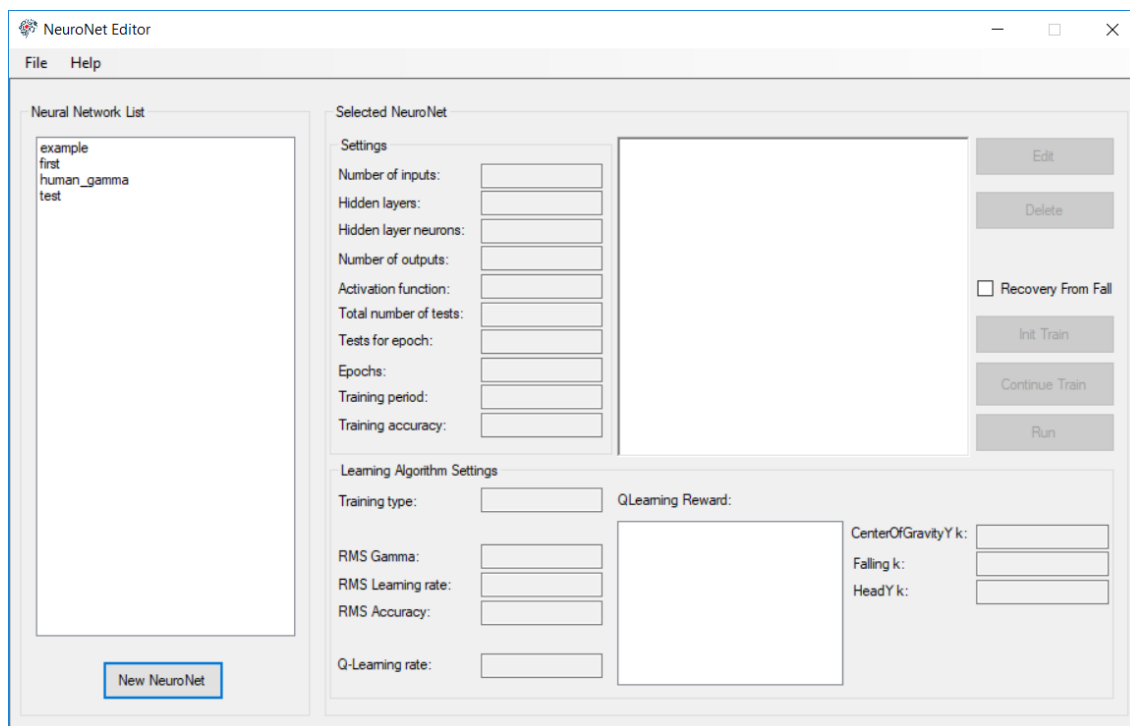


Рисунок 17 — Главное окно приложения

Приложение позволяет производить действия с уже существующими искусственными нейронными сетями или создать новую. После нажатия на кнопку создания нейронной сети появится интерфейс для настройки ее параметров (рисунок 18).

В открывшемся окне задается имя искусственной нейронной сети, ее характеристики, указанные в таблице 3, а также выбирается одна из существующих моделей объектов, которой сеть будет управлять. Количество входов и выходов нейронной сети устанавливаются в зависимости от выбранной модели.

Подп. и дата
Инв. № дубл.
Взам. инв. №
Подп. и дата
Инв. № подл.

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

**БР-02069964-090301-16-18**

Лист

45

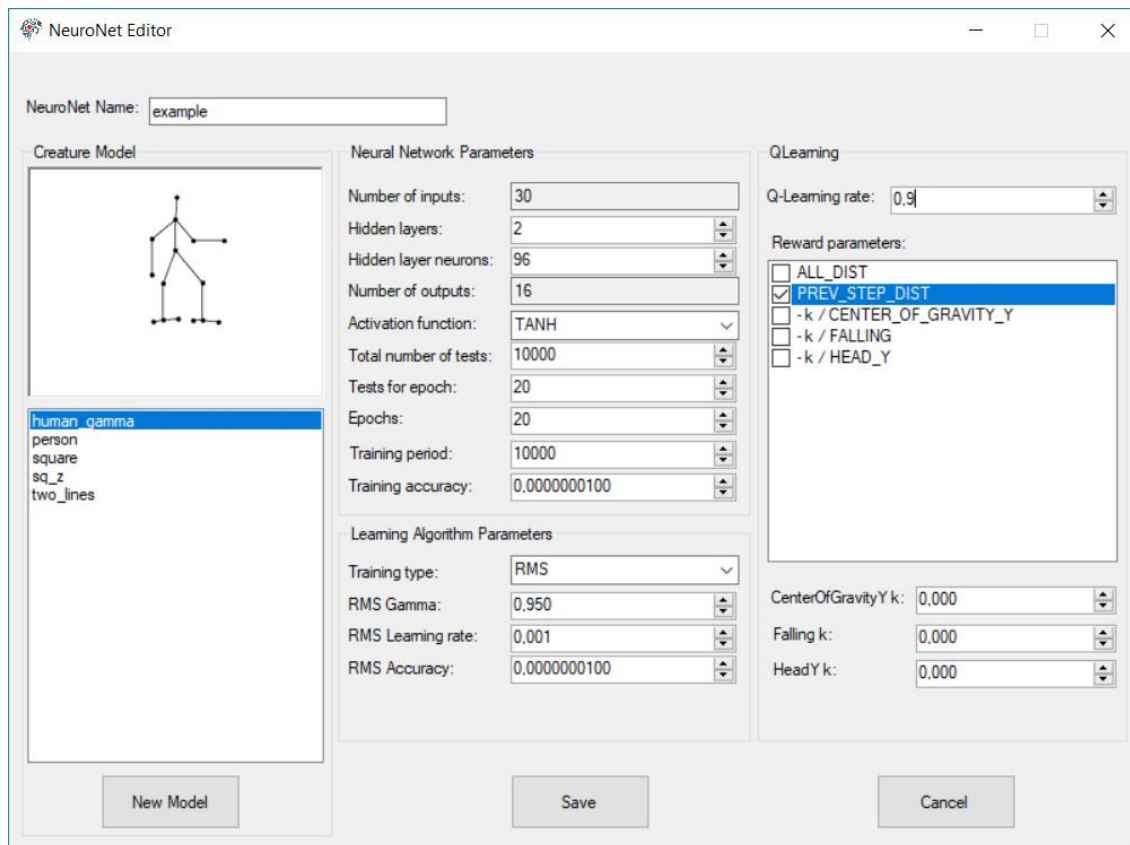


Рисунок 18 — Интерфейс настройки параметров нейронной сети

Таблица 3 — Параметры искусственной нейронной сети и алгоритма ее обучения

Наименование параметра	Описание	Допустимые значения
Creature Model	Модель, управляемая нейронной сетью	Модель выбирается из представленного списка
Number of inputs	Удвоенное количество суставов модели	Задается на основе выбранной модели
Hidden layers	Количество скрытых слоев сети	1 – 10
Hidden layers neurons	Количество нейронов в скрытом слое, каждый скрытый слой содержит	1 – 1000

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

Лист

46

Продолжение таблицы 3

	одинаковое количество нейронов	
Number of outputs	Количество допустимых действий модели (удвоенное количество подвижных отрезков модели)	Задается на основе выбранной модели
Activation function	Функция активации скрытых слоев, все скрытые слои имеют одинаковую функцию активации	Линейная (LINE), Сигмоида(SGMD), Гиперболический тангенс(TANH)
Total number of tests	Максимальный размер тестового набора, из которого делаются случайные выборки	1 – 100000
Tests for epoch	Размер случайной выборки из тестового набора, используемый для обучения	1 – 100000
Epochs	Количество эпох обучения	1 – 200
Training period	Интервал времени (в тиках), в течении которого будет продолжаться процесс обучения	1 – 500000
Training accuracy	Минимальное значение	[0,0000000001; 10]

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

Лист

47



Продолжение таблицы 3

	ошибки, при котором процесс обучения на данный момент времени будет остановлен	
Training type	Выбор алгоритма обучения	RMS Propagation
RMS Gamma	Значение $\gamma$ для алгоритма RMS Propagation	[0; 1]
RMS Learning rate	Значение $\eta$ для алгоритма RMS Propagation	[0; 1]
RMS Accuracy	Значение $\epsilon$ для алгоритма RMS Propagation	[0; 1]
Q-Learning rate	Коэффициент доверия новому действию в алгоритме Q-Learning	[0; 1]
Reward parameters	Параметры, учитываемые при расчете вознаграждения в алгоритме Q-Learning	ALL_DIST, PREV_STEP_DIST, CENTER_OF_GRAVITY_Y, FALLING, HEAD_Y

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	Подп. и дата

CenterOfGravityY k	Коэффициент значимости учета штрафа за приближение центра тяжести модели к «земле»	[0; 1000000]
Falling k	Коэффициент значимости учета штрафа за падение модели	[0; 1000000]

Окончание таблицы 3

HeadY k	Коэффициент значимости учета штрафа за приближение «ГОЛОВЫ» модели к «земле»	[0; 1000000]
---------	--	--------------

При необходимости может быть создана новая модель. Процесс создания модели состоит из нескольких этапов и реализуется с помощью специального интерфейса. Первоначально конструируется ее внешний вид (рисунок 19). Интерфейс предоставляет объекты двух типов: точки, соответствующие суставам модели, и отрезки, из которых модель состоит.

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	Подп. и дата

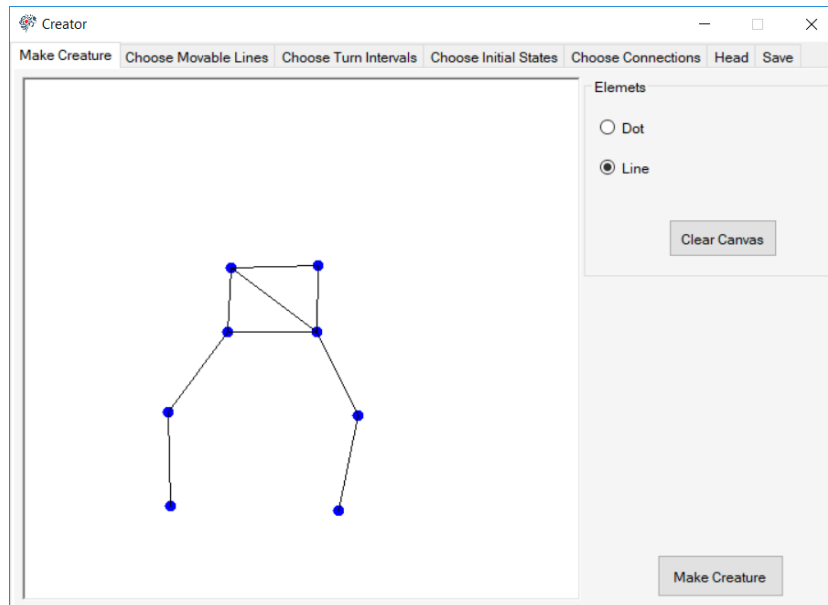


Рисунок 19 — Вкладка конструирования внешнего вида модели

Затем указываются подвижные отрезки. На вкладке отображается список всех отрезков с указанием сустава поворота (рисунок 20). Выбранные отрезки и соответствующие им суставы выделяются красным.

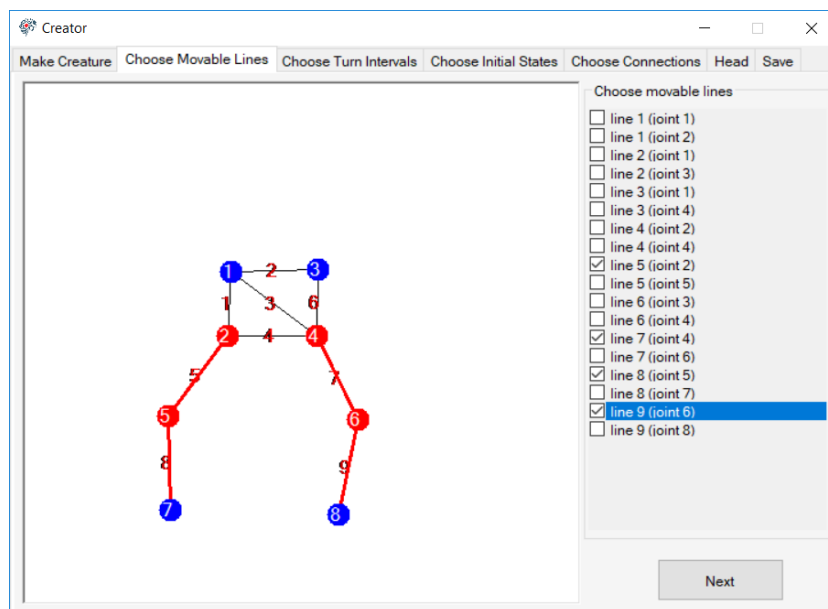


Рисунок 20 — Вкладка выбора подвижных отрезков

Следующим этапом задается общая скорость поворота отрезков, а для каждого подвижного отрезка определяется интервал угла поворота (рисунок

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Лист	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

21). После сохранения интервала для некоторого отрезка, границы интервала появляются на изображении модели.

На основе введенных значений углов поворота генерируются наборы возможных состояний отрезков, после чего можно выбрать начальное состояние для каждого отрезка (рисунок 22).

Следующий этап состоит в определении взаимосвязей между отрезками, то есть обозначаются те части модели, которые будут жестко связаны с текущим отрезком и в момент его поворота будут поворачиваться вместе с ним (рисунок 23).

На последнем этапе можно указать те суставы, которые будут считаться «головой» модели объекта (рисунок 24). Эти данные необходимы для расчета значения одного из параметров вознаграждения в алгоритме Q-Learning.

Результатом проделанных действий является модель с установленным набором параметров, которые можно записать в файл (рисунок 25).

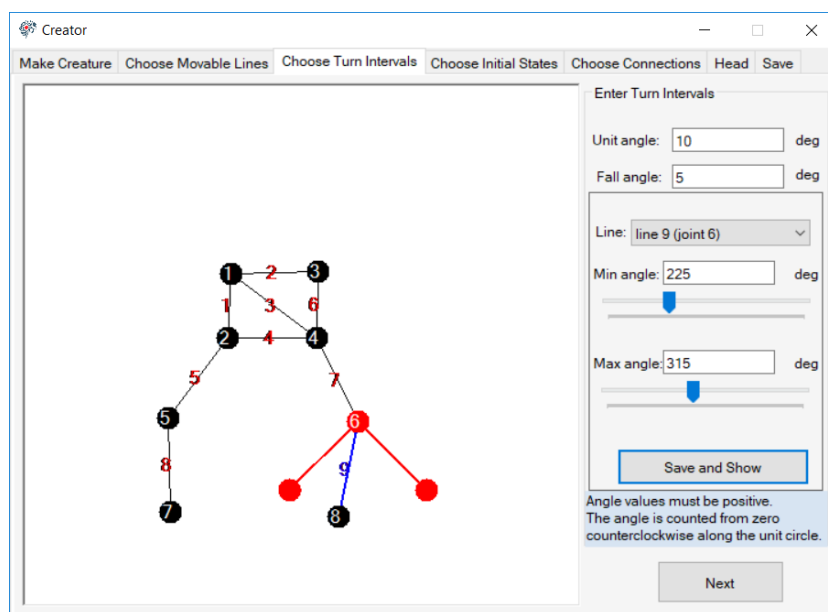


Рисунок 21 — Вкладка выбора интервалов углов поворота и скорости поворота подвижных отрезков

Подп. и дата
Инв. № дубл.
Взам. инв. №
Подп. и дата
Инв. № подл.

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

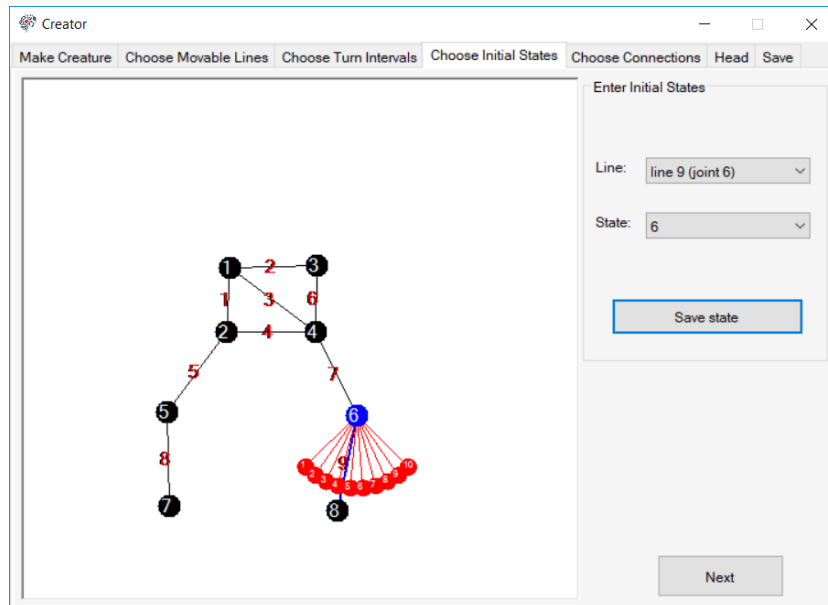


Рисунок 22 — Вкладка выбора начальных состояний подвижных отрезков

После сохранения модели, она станет доступна для выбора в списке моделей в интерфейсе редактирования параметров нейронной сети.

Для работы с некоторой нейронной сетью необходимо выбрать ее из списка, после чего ее параметры и управляемая ей модель отражаются на специальной панели окна, а также станет доступен ряд кнопок (рисунок 26).

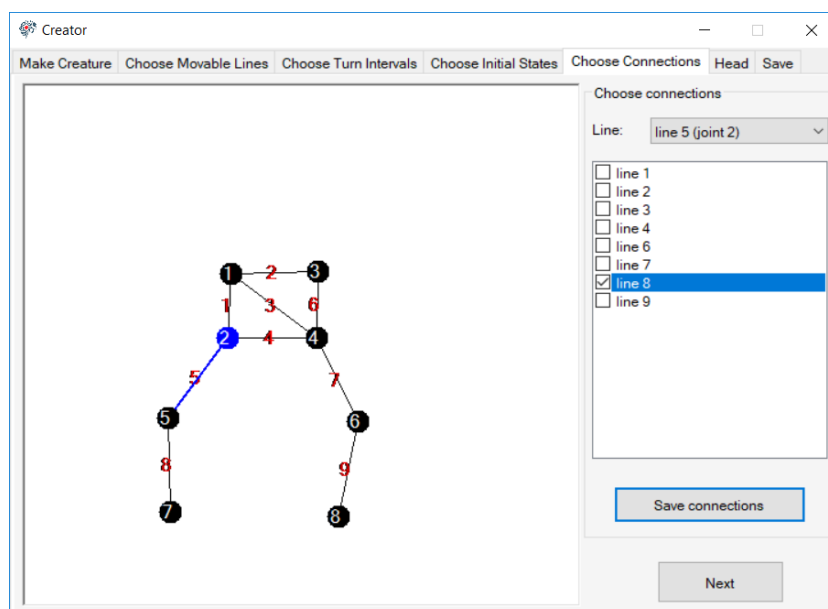


Рисунок 23 — Вкладка определения взаимосвязей между отрезками

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

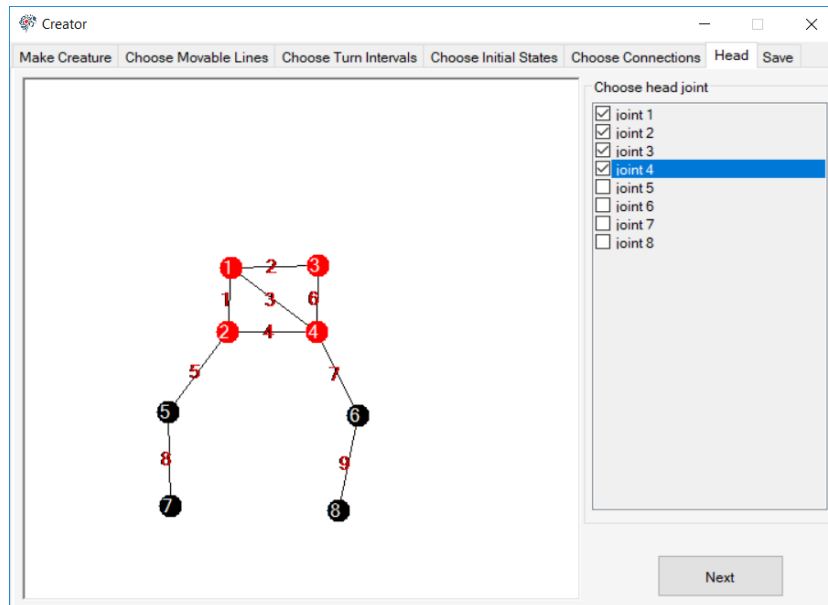


Рисунок 24 — Вкладка определения точек, принадлежащих «голове» модели объекта

С помощью этих кнопок можно открыть окно редактирования параметров нейронной сети, удалить сеть или запустить ее. Окно редактирования параметров сети будет иметь вид, показанный на рисунке 18 с тем лишь отличием, что поля формы будут заполнены текущими характеристиками выбранной нейронной сети.

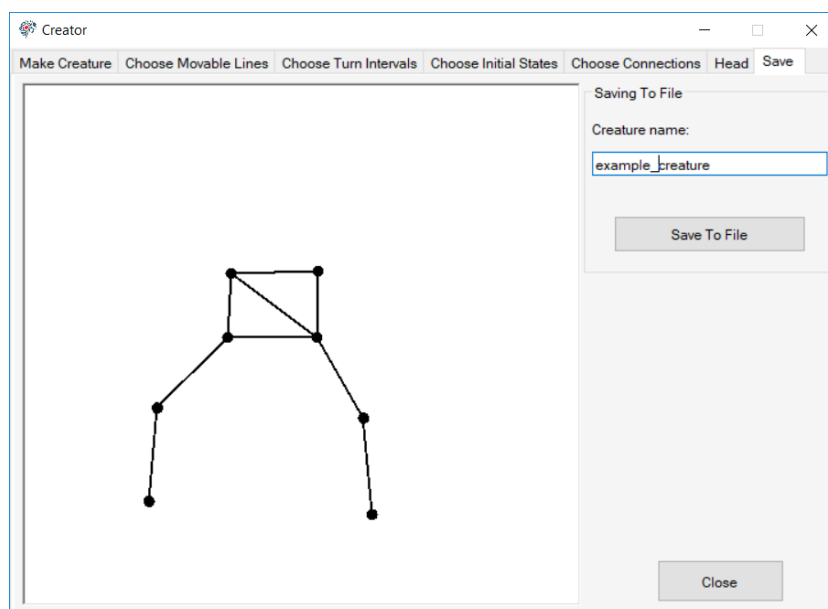


Рисунок 25 — Вкладка сохранения параметров модели объекта в файл

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

Изм	Лист	№ докум.	Подпись	Дата

БР-02069964-090301-16-18

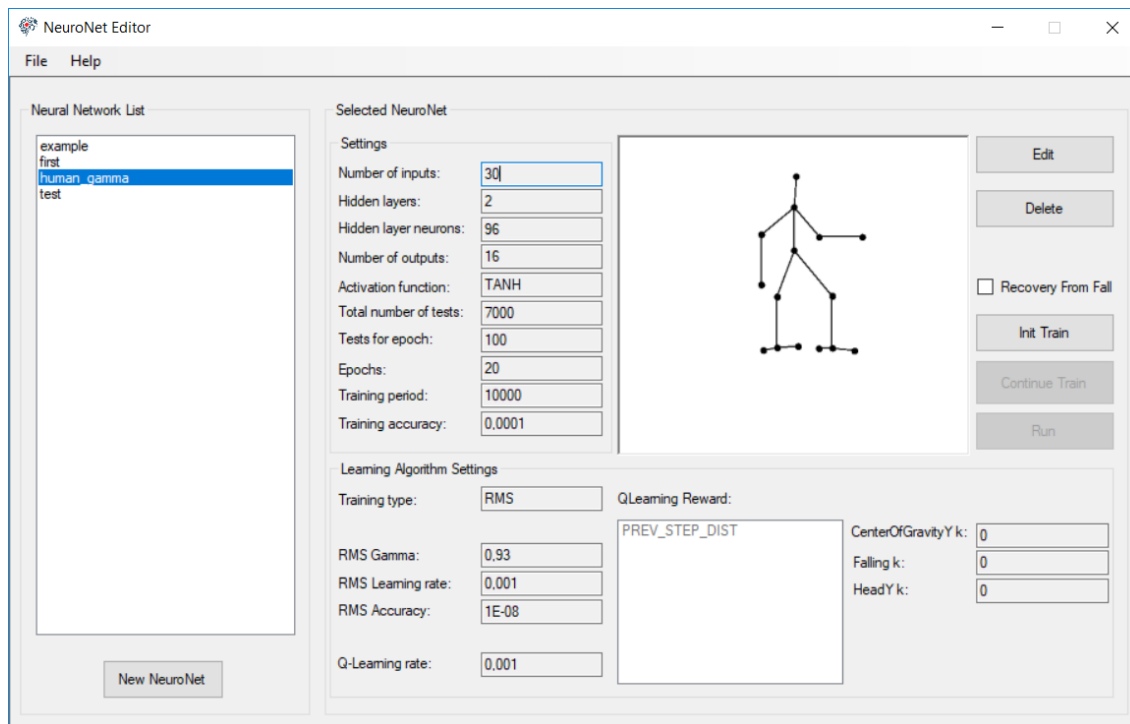


Рисунок 26 — Главное окно приложения

Запуск искусственной нейронной сети может производиться в одном из трех режимов:

- 1 Первый запуск на обучение, при котором веса и смещения нейронной сети задаются случайным образом, а отсчет пройденного времени начинается с нуля;
- 2 Продолжение обучения нейронной сети позволяет запустить ее с теми значениями весов и смещений и с тем положением модели объекта в пространстве, которые были в момент завершения последнего запуска сети на обучение;
- 3 Запуск обученной нейронной сети, при котором отключается режим обучения, и сеть не изменяет свои веса.

Также интерфейс позволяет разрешить или запретить восстановление модели в исходное состояние после падения без прерывания процесса обучения искусственной нейронной сети. Пройденное расстояние при этом сохраняется.

Пример окна запуска моделирования перемещения объекта представлен на рисунке 27.

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

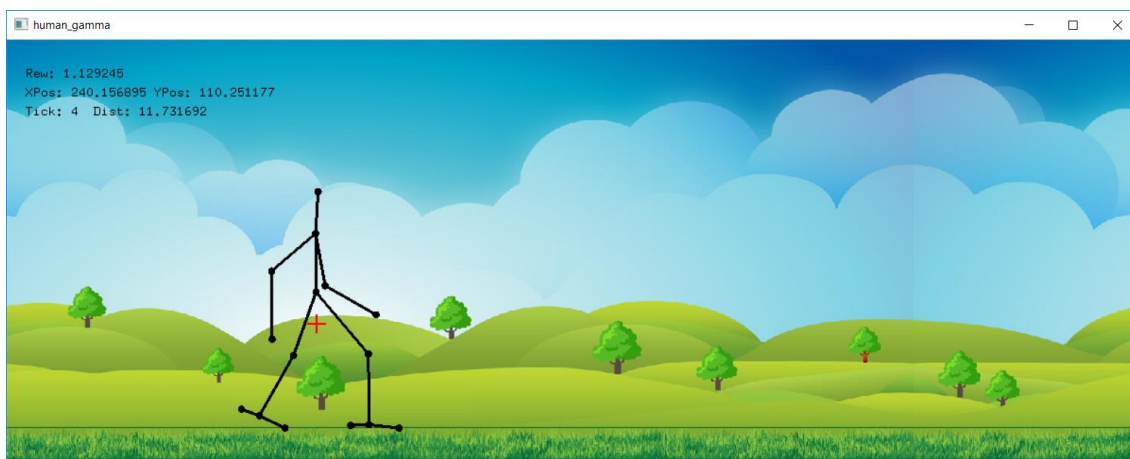


Рисунок 27 — Окно моделирования перемещения объекта

В окне отображается время, прошедшее с начала запуска, координаты центра тяжести модели и пройденное моделью расстояние. Красным цветом обозначен центр тяжести модели.

Инв.№ подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата

**БР-02069964-090301-16-18**

Лист

55



## 5 Анализ результатов

### 5.1 Условия и критерии проведения экспериментов

Для настройки и анализа эффективности работы нейронной сети был проведен ряд исследований. В качестве критерия для исследований была выбрана зависимость пройденного моделью за определенное количество времени расстояния (отсчет времени производился в тиках).

Производился подбор следующих характеристик нейронной сети и алгоритма обучения:

- количество нейронов скрытого слоя;
- комбинации параметров вознаграждения, используемого в алгоритме Q-Learning.

Часть параметров нейронной сети была зафиксирована (таблица 4).

Таблица 4 — Фиксированные параметры искусственной нейронной сети

Наименование параметра	Значение
Количество скрытых слоев	2
Функция активации	Гиперболический тангенс
Размер тестовой выборки	10000
Количество тестов в эпоху	100
Количество эпох	20
Время обучения	10000
Точность обучения	0,0001
Алгоритм обучения	RMS Propagation
RMS Гамма	0,95
RMS Скорость обучения	0,001
RMS Точность	0,00000001
Коэффициент доверия новому действию	0,001

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

Лист

56

Процесс тестирования заключался в запуске нейронной сети с заданными значениями параметров. Каждый вариант набора параметров запускался несколько раз, после чего полученные результаты усреднялись. Продолжительность запусков составляла 10000 тиков.

## 5.2 Сравнение эффективности обучения искусственных нейронных сетей с различным количеством нейронов в скрытом слое

Для исследования были использованы пять значений количества нейронов скрытого слоя. Остальные параметры нейронной сети были зафиксированы (таблица 5).

Таблица 5 — Фиксированные для текущего исследования параметры искусственной нейронной сети

Наименование параметра	Значение
Количество входов	16
Количество выходов	8
Параметры вознаграждения	PREV_STEP_DIST

Управляемая модель объекта изображена на рисунке 28.

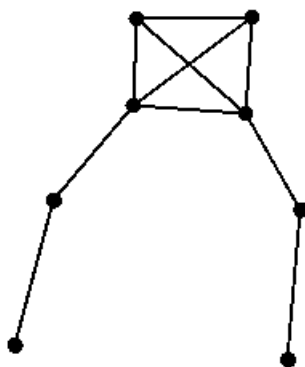


Рисунок 28 — Модель объекта

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

Результаты эксперимента представлены на рисунке 29.

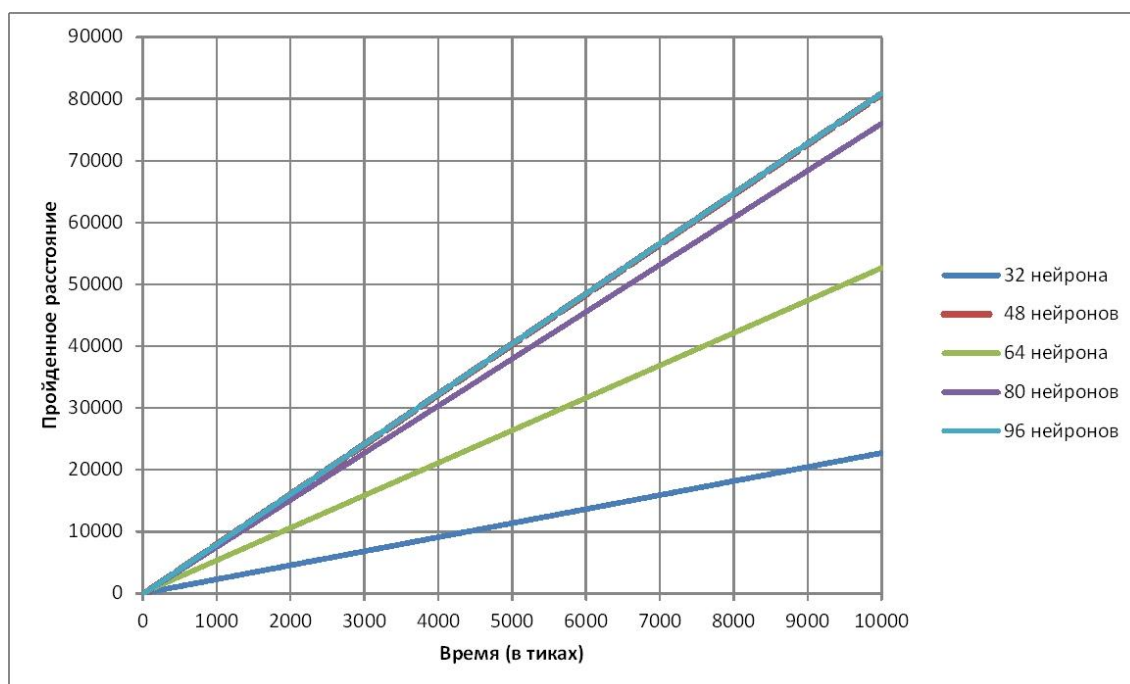


Рисунок 29 — Зависимость величины пройденного расстояния за определенный временной интервал от момента времени для нейронных сетей с разным количеством нейронов скрытого слоя

Судя по графику, наиболее эффективно нейронная сеть работает при значениях количества нейронов скрытого слоя, равных 48 и 96.

### 5.3 Сравнение эффективности обучения искусственных нейронных сетей с различными комбинациями параметров вознаграждения

Данное исследование проводилось на всех допустимых комбинациях параметров вознаграждений. Параметры подбирались таким образом, чтобы их значения не противоречили друг другу.

В процессе исследования была использована та же модель объекта и те же параметры нейронной сети, что и в предыдущем эксперименте. Количество нейронов в скрытых слоях зафиксировано и равно 96. Результаты эксперимента

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	Подп. и дата

представлены на рисунке 30.

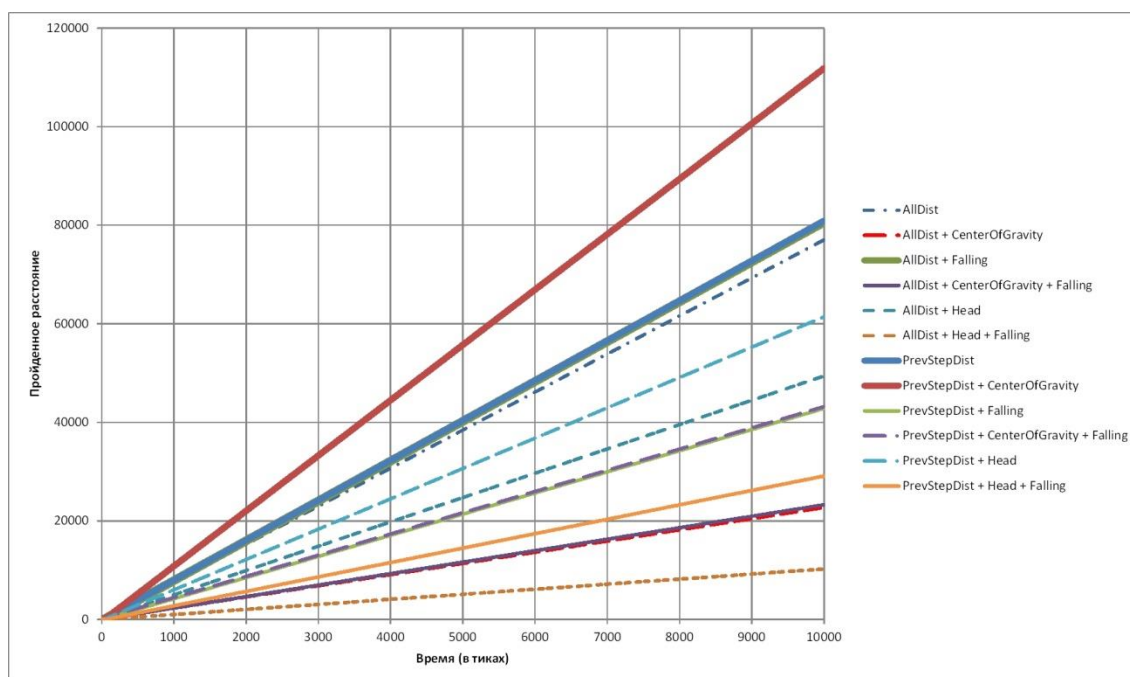


Рисунок 30 — Зависимость величины пройденного расстояния за определенный временной интервал от момента времени для нейронных сетей с различными комбинациями параметров вознаграждения

По полученным данным можно сделать вывод, что наилучшие результаты модель показывает при расчете вознаграждения на основе пройденного за последний шаг расстояния и комбинации этого параметра со штрафом за приближения центра тяжести модели к «земле».

#### 5.4 Эффективность работы нейронных сетей, управляющих различными моделями объектов

Данное исследование проводилось на трех различных моделях (рисунок 31). Количество входов и выходов искусственных нейронных сетей для каждой модели указано в таблице 6.

Количество нейронов в скрытом слое было зафиксировано равным 96.

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

Параметры для расчета вознаграждения были выбраны на основе результатов, предыдущего эксперимента. В первом случае в качестве вознаграждения использовалось расстояние, пройденное моделью за последний шаг, во втором — комбинация расстояния за последний шаг со штрафом за приближения центра тяжести модели к «земле». Результаты экспериментов представлены на рисунках 32 и 33 соответственно.

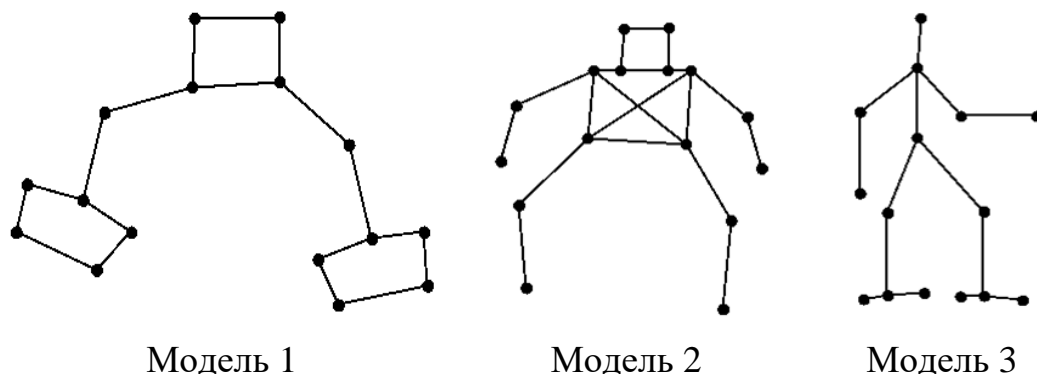


Рисунок 31 — Модели объектов, используемые при проведении исследования

Таблица 6 — Количество входов и выходов искусственных нейронных сетей для приведенных моделей

Наименование параметра	Модель 1	Модель 2	Модель 3
Количество входов	16	32	32
Количество выходов	8	16	16

По полученным данным в первом эксперименте при заданных параметрах нейронной сети и алгоритма обучения наилучший результат показала Модель 1, во втором — Модель 1 и Модель 3.

Инв.№ подл.	Взам. инв. №	Инв. № дубл.	Подп. и дата

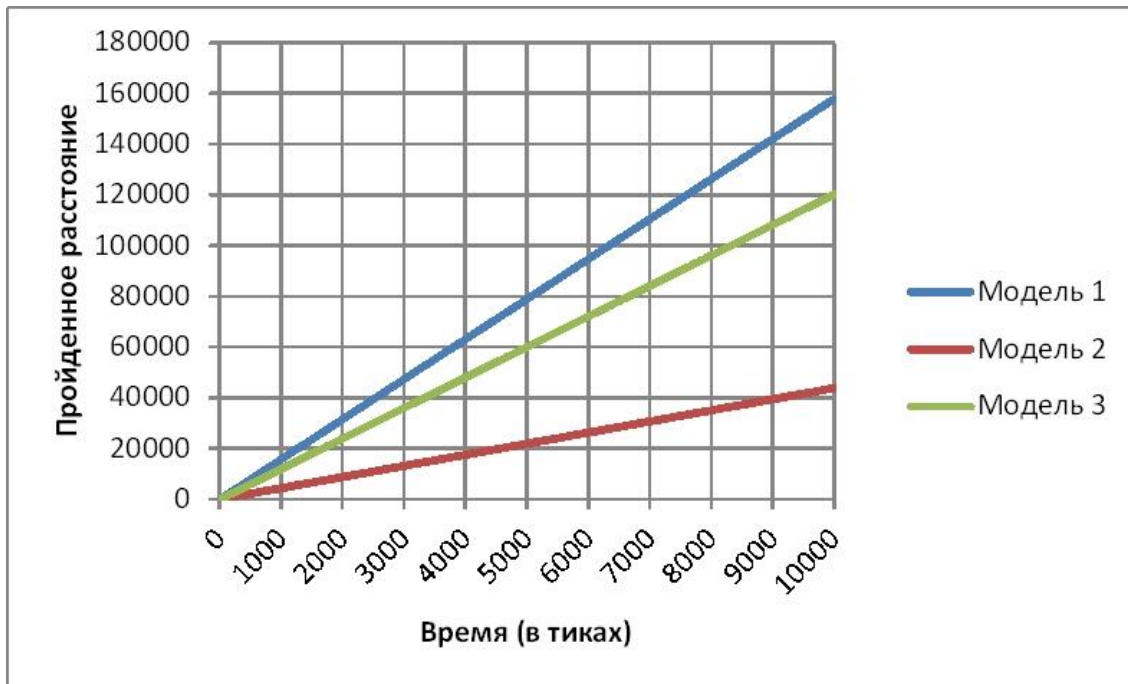


Рисунок 32 — Зависимость величины пройденного расстояния от момента времени для разных моделей с вознаграждением, равным расстоянию, пройденному моделью за последний шаг

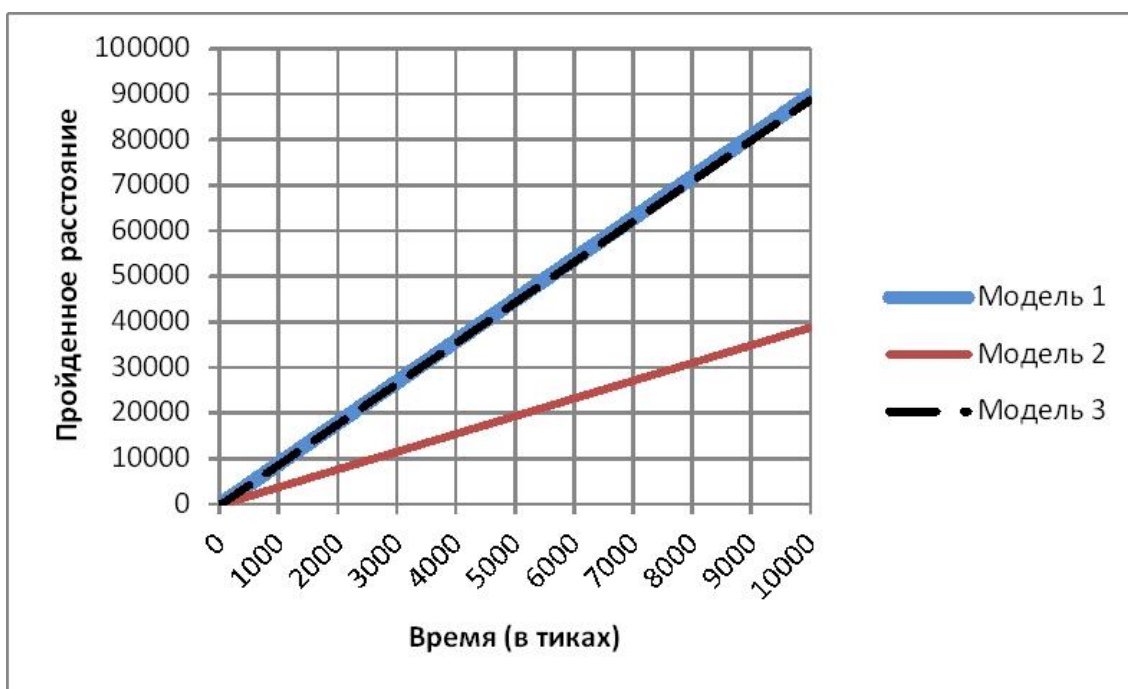


Рисунок 33 — Зависимость величины пройденного расстояния от момента времени для разных моделей с вознаграждением, равным сумме расстояния, пройденного за последний шаг, и штрафа за центр тяжести

Инв.№ подл.	Взам. инв. №	Инв. № дубл.	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата

БР-02069964-090301-16-18

Конструкция Модели 2 основана на конструкции модели, с помощью которой производилась настройка параметров нейронной сети и алгоритма ее обучения. Последняя была дополнена двумя конечностями и набором неподвижных отрезков. Расстояние, пройденное Моделью 2 за фиксированный промежуток времени, в обоих экспериментах меньше, чем у остальных моделей. Это связано с тем, что первая и третья модели имеют более устойчивую конструкцию, которая помогает избежать падения.

	Инв.№ подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата

*БР-02069964-090301-16-18*

## Заключение

В ходе данной работы была реализована искусственная нейронная сеть, предназначенная для управления моделью сложного объекта, создано приложение для моделирования процесса перемещения объекта в двумерном пространстве.

Разработанное приложение позволяет конструировать модели сложных объектов, создавать искусственные нейронные сети путем настройки их параметров, а также запускать сети на обучение.

В процессе реализации было изучено большое количество литературы, содержащей информацию по нейронным сетям, принципам их работы и обучению. Для описания многослойного персептрона и модели объекта на языке C++ была реализована система классов. Создание приложения производилось с помощью инструментов, предоставляемых языком C++ и интегрированной средой разработки Microsoft Visual Studio.

В ходе проведенных исследований были определены оптимальное количество нейронов скрытого слоя (32 нейрона) и параметры, используемые при расчете вознаграждения агента в алгоритме Q-Learning (расстояния, пройденное моделью за последний шаг за последний шаг и комбинация расстояния, пройденного за последний шаг, со штрафом за приближения центра тяжести модели к «земле»).

Данная программа позволит упростить процесс моделирования перемещения сложных объектов в пространстве.

Инв.№ подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата	<b>БР-02069964-090301-16-18</b>	Лист
						63



### Список использованных источников

1. Буч Г. Язык UML. Руководство пользователя / Рамбо Д., Якобсон И.; пер. с англ. Мухин Н. — 2-е изд., перераб. и доп. — М.: ДМК Пресс, 2006. — 496 с.: ил.
2. Короткий С. Нейронные сети: алгоритм обратного распространения; статья — 15 с.
3. Оссовский С. Нейронные сети для обработки информации / Пер. с польского И. Д. Рудинского. — М.: Финансы и статистика, 2002. — 344с.: ил.
4. Хайкин Саймон Нейронные сети: полный курс. — 2-е изд., перераб. и доп. — М.: Издательский дом «Вильямс», 2006 — 1104 с.: ил.
5. DEMYSTIFYING DEEP REINFORCEMENT LEARNING [Электронный ресурс]. — Режим доступа: <http://neuro.cs.ut.ee/demystifyingdeep-reinforcement-learning/>.— Загл. с экрана.
6. Google's DeepMind AI Just Taught Itself To Walk [Электронный ресурс]. — Режим доступа: <https://www.youtube.com/watch?v=gn4nRCC9TwQ>. — Загл. с экрана.
7. Learn to Walk (genetic algorithm & Neural Network) [Электронный ресурс]. — Режим доступа: <https://www.youtube.com/watch?v=h-89xjWpV4U>. — Загл. с экрана.
8. MSDN – сеть разработчиков Microsoft [Электронный ресурс]. — Режим доступа: <https://msdn.microsoft.com/ru-ru/default.aspx> .— Загл. с экрана
9. Neural Networks for Machine Learning [Электронный ресурс]. — Режим доступа: [http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture\\_slides\\_lec6.pdf](http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf) . — Загл. с экрана.

Инд. № подл.	Взам. инв. №	Инв. № дубл.	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата	<b>БР-02069964-090301-16-18</b>	Лист
						64

# Приложение А (обязательное) Программный код

## Листинг А.1 – Описание классов

```
extern int TOTAL_TESTS_NUMBER;
extern int CUR_TESTS_NUMBER;

enum TrainingTypes { RMS };
extern TrainingTypes TRAINING_TYPE;

struct Test {
    Matrix2d inputs;
    Matrix2d outputs;
    Test() {};
    Test(vector<vector<double>> _in, vector<vector<double>> _out) {
        inputs = _in;
        outputs = _out;
    }
    Test(Matrix2d _in, Matrix2d _out) {
        inputs = _in;
        outputs = _out;
    }
};

extern double LearningRate;

extern int NUM_HIDDEN_NEURONS;
extern int NUM_HIDDEN_LAYERS;
extern ActFuncTypes ACT_FUNC;

//For RMS
extern double RMS_GAMMA;
extern double RMS_LEARN_RATE;
extern double RMS_EPS;

class NeuroNet {
protected:
    int num_layers;
    vector<Layer> layers;
    int num_inputs;
    int num_outputs;
    Matrix2d outputs;
public:
    NeuroNet();
    NeuroNet(int ninputs, int nlayers, vector<int> nlneurons, vector<ActFuncTypes> _aft,
int noutputs);

    void Init(int ninputs, int nlayers, vector<int> nlneurons, vector<ActFuncTypes> _aft,
int noutputs, vector<Matrix2d> _biases, vector<Matrix2d> _weights);

    void AddTest(deque<Test>& ts, vector<vector<double>> _in, vector<vector<double>>
_out);
    void AddTest(deque<Test>& ts, Matrix2d _in, Matrix2d _out);
};
```

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

Изм	Лист	№ докум.	Подпись	Дата					

БР-02069964-090301-16-18

```

void Running(Test& test);
void Running(vector<vector<double>>& inps);
double RunningLearningOffline(deque<Test> & tests, vector<int>& tests_pos);
void ResilientPropagation();
double RPropLearningOffline(deque<Test> & tests, vector<int>& tests_pos);
void RMSPropagation();
double RMSLearningOffline(deque<Test> & tests, vector<int>& tests_pos);

void PrintWeightsAndBiases(ostream& fout, bool print_null);

Matrix2d GetOutput();
void CalcDeltaAndGrad(Test& test);
double CalcError(Test& test);
void CorrectWeightsAndBiases(vector<Matrix2d>& _gradient, vector<Matrix2d>& _delta);

void SetWeights(vector<Matrix2d> _w);
void SetBiases(vector<Matrix2d> _b);

friend ostream& operator << (ostream& out, NeuroNet& _nnet);
};

enum ActFuncTypes { NA, TANH, SGMD, LINE };

class Layer {
private:
    ActFuncTypes activation_func;
    int num_neurons;
    int num_prev_neurons;

    Matrix2d sgmd_func(Matrix2d& _states);           // Сигмоидальная функция
    Matrix2d tanh_func(Matrix2d& _states);          // Гиперболический тангенс
    Matrix2d diff_sgmd_func(Matrix2d& _axons);      // Сигмоидальная функция
    Matrix2d diff_tanh_func(Matrix2d& _axons);      // Гиперболический тангенс

public:
    Matrix2d weights;           // Веса нейронов
    Matrix2d states;           // Состояния нейронов
    Matrix2d biases;           // Смещения нейронов
    Matrix2d axons;           // Аксоны нейронов
    Matrix2d delta;           // Дельта
    Matrix2d prev_delta;       // Предыдущая дельта
    Matrix2d delta_sum;
    Matrix2d prev_delta_sum;
    Matrix2d grad;           // Градиент
    Matrix2d prev_grad;
    Matrix2d grad_sum;
    Matrix2d prev_grad_sum;

    Matrix2d weights_correct;
    Matrix2d biases_correct;

    //RMS
    Matrix2d rms;
    Matrix2d rms_biases;
    Matrix2d rmsn;
    Matrix2d rmsn_biases;

    Layer();
    Layer(int nneurons, int nprevneurons, ActFuncTypes act_func);

    Layer& operator = (Layer& _l);
};

```

Подп. и дата

Инв. № дубл.

Взам. инв. №

Подп. и дата

Инв. № подл.

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

Лист

66

```

    void Init(int nneurons, int nprevneurons, ActFuncTypes act_func, Matrix2d _biases,
Matrix2d _weights);

    int GetNumNeurons();

    void CalcStates(Layer& prev_layer);
    void CalcAxons();
    Matrix2d CalcDiffs();
};

extern random_device rnd_dev;
extern mt19937 eng;
extern uniform_int_distribution<> rnd_gen;

int SomeRand();

double GetRandVal(double min_val, double max_val);

class Matrix2d {
private:
    int num_rows;
    int num_cols;
    double* matrix;
public:
    Matrix2d() { num_rows = 0; num_cols = 0; matrix = nullptr; };
    ~Matrix2d();
    Matrix2d(int nrow, int ncol, double val = 0.0);
    Matrix2d(const Matrix2d& _m);
    Matrix2d(vector<double>& _m);

    void InitMatrix(double val);
    void InitMatrix(int nrow, int ncol, double val);
    void InitMatrixDiagByOne();
    void InitRandom(double minval, double maxval);
    void InitValue(double val);

    int GetNumRows() const;
    int GetNumCols() const;

    Matrix2d Transpose() const; // Транспонирование матрицы
    Matrix2d MultElByEl(const Matrix2d& _m) const; // Поэлементное умножение матриц
    double SumAll() const; // Сложение всех элементов матрицы
    double SumAllSqr() const; // Сложение квадратов всех элементов матрицы
    Matrix2d Sqrt();

    double& at(const int& i, const int& j);
    double at(const int& i, const int& j) const;

    Matrix2d& operator = (vector<double>& _m);
    Matrix2d& operator = (vector<vector<double>>& _m);
    Matrix2d& operator = (Matrix2d& _m);

    Matrix2d operator + (const double val); // Сложение матрицы с числом
    Matrix2d& operator += (const double val); // Сложение матрицы с числом
    Matrix2d operator + (Matrix2d& _m); // Сложение матриц
    Matrix2d& operator += (Matrix2d& _m); // Сложение матриц

    Matrix2d operator - (const double val); // Вычитание числа из матрицы
    Matrix2d& operator -= (const double val); // Вычитание числа из матрицы
    Matrix2d operator - (Matrix2d& _m); // Вычитание матриц
    Matrix2d& operator -= (Matrix2d& _m); // Вычитание матриц

```

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	Подп. и дата

```

Matrix2d operator * (const double val);           // Умножение матрицы на число
Matrix2d& operator *= (const double val);        // Умножение матрицы на число
Matrix2d operator * (Matrix2d& _m);             // Произведение матриц

Matrix2d operator / (const double val);         // Деление элементов матрицы на число
Matrix2d& operator /= (const double val);       // Деление элементов матрицы на число

friend Matrix2d operator / (const double val, const Matrix2d& _m);
};

class Creature {
private:
    double fall_unit_angle = M_PI / 30; // единичный угол падения
    double turn_unit_angle = M_PI / 30; // единичный угол поворота
    double start_pos = 0.0;
    double start_max_x = 0.0;
    double cur_delta_distance = 0.0;

    double falling = 0.0;

    vector<pair<double, double>> joints; // суставы (точки соединения)
    vector<pair<int, int>> lines; // отрезки, из которых состоит модель
    vector<double> lines_length;

    vector<pair<int, int>> movable_lines; //{номер отрезка; номер сустава, в котором он
поворачивается}

    vector<pair<double, double>> turn_intervals; // левая и правая границы интервалов
поворота отрезков
    vector<pair<int, int>> states_mvlines; // текущие состояния подвижных отрезков,
количество состояний отрезка
    vector<vector<int>> refs; //какие отрезки повернутся, если повернуть текущий
vector<int> head_points;

public:
    Creature() {};
    void InitCreature(vector<pair<double, double>> _joints,
                      vector<pair<int, int>> _lines,
                      vector<pair<int, int>> _mvlines,
                      vector<pair<double, double>> _turnint,
                      vector<pair<int, int>> states,
                      vector<vector<int>> _refs,
                      vector<int> _head_points);

    int GetNumActions() { return 2*movable_lines.size(); } //поворот каждого отрезка в
двух направлениях
    vector<pair<double, double>> GetJoints() { return joints; }
    double GetStartPos() { return start_pos; }
    vector<pair<int, int>> GetCurLinesStates() { return states_mvlines; }
    int GetNumJoints() { return joints.size(); }

    vector<Line> GetLines();
    double GetCenterOfGravityX();
    double GetCenterOfGravityY();
    double GetCurDeltaDistance();
    double GetTraveledDistance();

    double GetFalling() { return falling; }
    double GetHeadY();

```

Инв.№ подл.	Взам. инв. №	Инв. № дубл.	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата

БР-02069964-090301-16-18

Лист

68

```

void SetFallUnitAngle(double val) { fall_unit_angle = val; }
void SetTurnUnitAngle(double val) { turn_unit_angle = val; }

void SetJoints(vector<pair<double, double>> new_joints);
void SetStates(vector<int> new_states);

pair<int, int> GetAction(int action_num);
bool CanDoAction(int action);
bool CanDoFullAction(int action);
int Rotate(int line, int tdir);
void CorrectPos(int line, int tdir);
void CorrectPos(int line, int tdir, int point);
void Falling();
void UpdatePos(int action_num); // номер действия

void PrintCreatureJoints(ostream& fout) {
    fout << "-----Creature Joints-----"
" << endl;
    for (int i = 0; i < joints.size(); ++i) {
        fout << fixed <<setprecision(8) << joints[i].first << " " << fixed <<
setprecision(8) << joints[i].second << endl;
    }
    fout << "-----Creature States-----"
" << endl;
    for (int i = 0; i < states_mvlines.size(); ++i) {
        fout << states_mvlines[i].first << " ";
    }
    fout << endl << "-----"
-----" << endl;

}

};

```

## Листинг А.2 — Реализация методов классов

```

NeuroNet::NeuroNet() {
    num_inputs = 0;
    num_layers = 0;
    num_outputs = 0;
    layers.resize(0);
}
NeuroNet::NeuroNet(int ninputs, int nlayers, vector<int> nlnurons, vector<ActFuncTypes>
_aft, int noutputs) {
    num_inputs = ninputs;
    num_layers = nlayers;
    num_outputs = noutputs;
    layers.push_back(Layer(ninputs, 0, LINE));
    for (int i = 0; i < nlayers; ++i) {
        layers.push_back(Layer(nlnurons[i], layers.back().GetNumNeurons(), _aft[i]));
    }
}

void NeuroNet::Init(int ninputs, int nlayers, vector<int> nlnurons, vector<ActFuncTypes>
_aft, int noutputs, vector<Matrix2d> _biases, vector<Matrix2d> _weights) {
    num_inputs = ninputs;
    num_layers = nlayers;
    num_outputs = noutputs;
    outputs = Matrix2d(1, noutputs);
    Matrix2d _wm, _bm;
    _wm.InitMatrix(ninputs, 1, 1.0);
}

```

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

Изм	Лист	№ докум.	Подпись	Дата	БР-02069964-090301-16-18	Лист
						69

```

    _bm = Matrix2d(1, ninputs);
    layers.push_back(Layer());
    layers.back().Init(ninputs, 0, LINE, _bm, _wm);

    int nprevneurons;
    for (int i = 0; i < nlayers; ++i) {
        nprevneurons = layers.back().GetNumNeurons();
        layers.push_back(Layer());
        layers.back().Init(nlneurons[i], nprevneurons, _aft[i], _biases[i],
_weights[i]);
    }
}
void NeuroNet::AddTest(deque<Test>& ts, vector<vector<double>> _in, vector<vector<double>>
_out) {
    Matrix2d test_in;
    Matrix2d test_out;
    test_in = _in;
    test_out = _out;
    if (ts.size() >= TOTAL_TESTS_NUMBER)
        ts.pop_front();
    Test new_test(test_in, test_out);
    ts.push_back(new_test);
}
void NeuroNet::AddTest(deque<Test>& ts, Matrix2d _in, Matrix2d _out) {
    if (ts.size() >= TOTAL_TESTS_NUMBER)
        ts.pop_front();
    Test new_test(_in, _out);
    ts.push_back(new_test);
}
void NeuroNet::Running(Test& test) {
    int n = layers[0].states.GetNumRows();
    int m = layers[0].states.GetNumCols();
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            layers[0].states.at(i, j) = test.inputs.at(i, j);
        }
    }
    layers[0].CalcAxons();

    for (int i = 1; i < layers.size(); ++i) {
        layers[i].CalcStates(layers[i - 1]);
        layers[i].CalcAxons();
    }
    outputs = layers.back().axons;
}
void NeuroNet::Running(vector<vector<double>>& inputs) {
    Test new_test;
    new_test.inputs = inputs;
    Running(new_test);
}

double NeuroNet::RunningLearningOffline(deque<Test> & tests, vector<int>& tests_pos) {
    vector<Matrix2d> _delta(layers.size());
    vector<Matrix2d> _grad(layers.size());

    for (int i = 0; i < layers.size(); ++i) {
        _delta[i] = Matrix2d(layers[i].delta.GetNumRows(),
layers[i].delta.GetNumCols());
        _grad[i] = Matrix2d(layers[i].grad.GetNumRows(), layers[i].grad.GetNumCols());
    }

    for (int i = 0; i < tests_pos.size(); ++i) {
        Running(tests[tests_pos[i]]);
    }
}

```

Подп. и дата	
Инв. № дудл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

```

        CalcDeltaAndGrad(tests[tests_pos[i]]);

        for (int j = 1; j < layers.size(); ++j) {
            _delta[j] += layers[j].delta;
            _grad[j] += layers[j].grad;
        }
    }

    // Вычисление суммы квадратов градиентов
    double norm = 0.0;
    for (int i = 1; i < layers.size(); ++i) {
        norm += _delta[i].SumAllSqr();
        norm += _grad[i].SumAllSqr();
    }

    if (fabs(norm) < 1e-7) {
        return 0.0;
    }

    // Нормировка градиентов
    norm = sqrt(fabs(norm));

    for (int i = 1; i < layers.size(); ++i) {
        _delta[i] /= norm;
        _grad[i] /= norm;
    }

    CorrectWeightsAndBiases(_grad, _delta);

    return norm;
}

void NeuroNet::ResilientPropagation() {
    double EttaPlus = 1.2;
    double EttaMinus = 0.5;
    double MinCorrectVal = 1e-6;
    double MaxCorrectVal = 50.0;
    for (int i = 1; i < layers.size(); ++i)
    {
        for (int j = 0; j < layers[i].grad_sum.GetNumRows(); ++j)
        {
            for (int k = 0; k < layers[i].grad_sum.GetNumCols(); ++k)
            {
                double correct_val = 0.0;
                double curmatrixult = layers[i].grad_sum.at(j, k) *
layers[i].prev_grad_sum.at(j, k);
                if (curmatrixult == 0.0)
                    correct_val = rand() / RAND_MAX;
                else if (curmatrixult > 0.0)
                    correct_val = min(EttaPlus *
layers[i].weights_correct.at(j, k), MaxCorrectVal);
                else if (curmatrixult < 0.0)
                    correct_val = max(EttaMinus *
layers[i].weights_correct.at(j, k), MinCorrectVal);

                layers[i].weights_correct.at(j, k) = correct_val;

                if (layers[i].grad_sum.at(j, k) == 0.0)
                    continue;

                if (layers[i].grad_sum.at(j, k) > 0.0)
                    layers[i].weights.at(j, k) -= correct_val;
                else

```

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18



```

        layers[i].weights.at(j, k) += correct_val;
    }
}

for (int j = 0; j < layers[i].delta_sum.GetNumRows(); ++j)
{
    double cur_correct = 0.0;
    double curmatrixult = layers[i].delta_sum.at(0, j) *
layers[i].prev_delta_sum.at(0, j);
    if (curmatrixult == 0.0)
        cur_correct = rand() / RAND_MAX;
    else if (curmatrixult > 0.0)
        cur_correct = min(EttaPlus * layers[i].biases_correct.at(0, j),
MaxCorrectVal);
    else if (curmatrixult < 0.0)
        cur_correct = max(EttaMinus * layers[i].biases_correct.at(0, j),
MinCorrectVal);

    layers[i].biases_correct.at(0, j) = cur_correct;

    if (layers[i].delta_sum.at(0, j) == 0.0)
        continue;

    if (layers[i].delta_sum.at(0, j) > 0.0)
        layers[i].biases.at(0, j) -= cur_correct;
    else
        layers[i].biases.at(0, j) += cur_correct;
}
}
}

double NeuroNet::RPropLearningOffline(deque<Test> & tests, vector<int>& tests_pos) {
    for (int i = 0; i < layers.size(); ++i) {
        layers[i].delta_sum.InitValue(0.0);
        layers[i].grad_sum.InitValue(0.0);
    }

    for (int i = 0; i < tests_pos.size(); ++i) {
        Running(tests[tests_pos[i]]);
        CalcDeltaAndGrad(tests[tests_pos[i]]);

        for (int j = 1; j < layers.size(); ++j) {
            layers[j].delta_sum += layers[j].delta;
            layers[j].grad_sum += layers[j].grad;
        }

        // Вычисление суммы квадратов градиентов
        double norm = 0.0;
        for (int i = 1; i < layers.size(); ++i) {
            norm += layers[i].delta_sum.SumAllSqr();
            norm += layers[i].grad_sum.SumAllSqr();
        }

        if (norm < 1e-6) {
            return 0.0;
        }

        ResilientPropagation();

        for (int i = 1; i < layers.size(); ++i) {
            layers[i].prev_delta_sum = layers[i].delta_sum;
            layers[i].prev_grad_sum = layers[i].grad_sum;
        }
    }
}

```

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

```

double err = 0.0;
for (int i = 0; i < tests_pos.size(); ++i) {
    Running(tests[tests_pos[i]]);
    err += CalcError(tests[tests_pos[i]]);
}

return err;
}

void NeuroNet::RMSPropagation()
{
    for (int i = 1; i < layers.size(); ++i)
    {
        layers[i].weights -= (RMS_LEARN_RATE / (layers[i].rms -
layers[i].rmsn.MultElByEl(layers[i].rmsn) + RMS_EPS).Sqrt()).MultElByEl(layers[i].grad_sum);
        layers[i].biases -= (RMS_LEARN_RATE / (layers[i].rms_biases -
layers[i].rmsn_biases.MultElByEl(layers[i].rmsn_biases) +
RMS_EPS).Sqrt()).MultElByEl(layers[i].delta_sum);
    }
}

double NeuroNet::RMSLearningOffline(deque<Test> & tests, vector<int>& tests_pos)
{
    for (int i = 0; i < layers.size(); ++i)
    {
        layers[i].grad_sum.InitValue(0.0);
        layers[i].delta_sum.InitValue(0.0);
    }

    for (int i = 0; i < tests_pos.size(); ++i)
    {
        Running(tests[tests_pos[i]]);
        CalcDeltaAndGrad(tests[tests_pos[i]]);

        for (int j = 1; j < layers.size(); ++j) {
            layers[j].delta_sum += layers[j].delta;
            layers[j].grad_sum += layers[j].grad;
        }

        for (int i = 1; i < layers.size(); ++i)
        {
            layers[i].rms = layers[i].rms * RMS_GAMMA +
layers[i].grad_sum.MultElByEl(layers[i].grad_sum) * (1.0 - RMS_GAMMA);
            layers[i].rms_biases = layers[i].rms_biases * RMS_GAMMA +
layers[i].delta_sum.MultElByEl(layers[i].delta_sum) * (1.0 - RMS_GAMMA);
            layers[i].rmsn = layers[i].rmsn * RMS_GAMMA + layers[i].grad_sum * (1.0 -
RMS_GAMMA);
            layers[i].rmsn_biases = layers[i].rmsn_biases * RMS_GAMMA +
layers[i].delta_sum * (1.0 - RMS_GAMMA);
        }

        RMSPropagation();
        for (int i = 0; i < layers.size(); ++i)
        {
            layers[i].prev_grad_sum = layers[i].grad_sum;
            layers[i].prev_delta_sum = layers[i].delta_sum;
        }

        double err = 0.0;
        for (int i = 0; i < tests_pos.size(); ++i) {
            Running(tests[tests_pos[i]]);
            err += CalcError(tests[tests_pos[i]]);
        }
    }
}

```

Инд. № подл.	Подп. и дата
Взам. инв. №	Инд. № дубл.
Подп. и дата	

Изм	Лист	№ докум.	Подпись	Дата	<b>БР-02069964-090301-16-18</b>	Лист
						73

```

    }

    return err;
}

void NeuroNet::PrintWeightsAndBiases(ostream& fout, bool print_null) {
    fout << "-----Weights-----" << endl;
    for (int i = 1; i < layers.size(); ++i) {
        Matrix2d m = layers[i].weights.Transpose();
        for (int j = 0; j < m.GetNumRows(); ++j) {
            for (int k = 0; k < m.GetNumCols(); ++k) {
                if (!print_null)
                    fout << fixed << setprecision(7) << m.at(j, k) << " ";
                else
                    fout << fixed << setprecision(7) << 0.0 << " ";
            }
            fout << endl;
        }
    }
    fout << "-----Biases-----" << endl;
    for (int i = 1; i < layers.size(); ++i) {
        Matrix2d m = layers[i].biases;
        for (int j = 0; j < m.GetNumRows(); ++j) {
            for (int k = 0; k < m.GetNumCols(); ++k) {
                if (!print_null)
                    fout << fixed << setprecision(7) << m.at(j, k) << " ";
                else
                    fout << fixed << setprecision(7) << 0.0 << " ";
            }
            fout << endl;
        }
    }
}

Matrix2d NeuroNet::GetOutput() {
    return outputs;
}

void NeuroNet::CalcDeltaAndGrad(Test& test) {
    layers[layers.size() - 1].delta = (layers[layers.size() - 1].axons -
test.outputs).MultElByEl(layers[layers.size() - 1].CalcDiffs());
    for (int i = layers.size() - 2; i > 0; --i) {
        layers[i].delta = (layers[i + 1].delta*layers[i +
1].weights).MultElByEl(layers[i].CalcDiffs());
    }

    for (int i = 1; i < layers.size(); ++i) {
        layers[i].grad = layers[i].delta.Transpose() * layers[i - 1].axons;
    }
}

double NeuroNet::CalcError(Test& test) {
    double res = (test.outputs - layers.back().axons).SumAllSqr() / 2.0;
    return res;
}

void NeuroNet::CorrectWeightsAndBiases(vector<Matrix2d>& _gradient, vector<Matrix2d>&
_delta) {
    for (int i = 1; i < layers.size(); ++i) {
        layers[i].weights -= _gradient[i] * LearningRate;
        layers[i].biases -= _delta[i] * LearningRate;
    }
}

void NeuroNet::SetWeights(vector<Matrix2d> _w) {
    for (int i = 1; i < layers.size(); ++i) {
        Matrix2d m = _w[i - 1].Transpose();

```

Подп. и дата

Инв. № дубл.

Взам. инв. №

Подп. и дата

Инв. № подл.

БР-02069964-090301-16-18

Лист

74

Изм Лист № докум. Подпись Дата

```

        layers[i].weights = m;
    }
}
void NeuroNet::SetBiases(vector<Matrix2d> _b) {
    for (int i = 1; i < layers.size(); ++i) {
        layers[i].biases = _b[i - 1];
    }
}
ostream& operator << (ostream& out, NeuroNet& _nnet) {
    for (int i = 0; i < _nnet.num_outputs; ++i) {
        out << fixed << setprecision(6) << _nnet.outputs.at(0, i) << " ";
    }
    return out;
}

Matrix2d Layer::sgmd_func(Matrix2d& _states) {
    int n = _states.GetNumRows();
    int m = _states.GetNumCols();
    Matrix2d res(n, m);
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            if (_states.at(i, j) < (-1.0 * 35))
                res.at(i, j) = 1e-12;
            else
                res.at(i, j) = 1.0 / (1.0 + exp(-_states.at(i, j)));
        }
    }
    return res;
}

Matrix2d Layer::tanh_func(Matrix2d& _states) {
    int n = _states.GetNumRows();
    int m = _states.GetNumCols();
    Matrix2d res(n, m);
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            res.at(i, j) = tanh(_states.at(i, j));
        }
    }
    return res;
}

Matrix2d Layer::diff_sgmd_func(Matrix2d& _axons) {
    int n = _axons.GetNumRows();
    int m = _axons.GetNumCols();
    Matrix2d res(n, m);
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            res.at(i, j) = (1.0 - _axons.at(i, j)) * _axons.at(i, j);
        }
    }
    return res;
}

Matrix2d Layer::diff_tanh_func(Matrix2d& _axons) {
    int n = _axons.GetNumRows();
    int m = _axons.GetNumCols();
    Matrix2d res(n, m);
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            res.at(i, j) = (1.0 - _axons.at(i, j)) * _axons.at(i, j);
        }
    }
    return res;
}
}

```

Подп. и дата	
Инв. № дудл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

```

Layer::Layer() {
    activation_func = NA;
    num_prev_neurons = 0;
    num_neurons = 0;
    weights = Matrix2d();
    states = Matrix2d();
    biases = Matrix2d();
    axons = Matrix2d();
    delta = Matrix2d();
    grad = Matrix2d();
    prev_delta = Matrix2d();
    delta_sum = Matrix2d();
    prev_delta_sum = Matrix2d();
    prev_grad = Matrix2d();
    grad_sum = Matrix2d();
    prev_grad_sum = Matrix2d();
    weights_correct = Matrix2d();
    biases_correct = Matrix2d();
    rms = Matrix2d();
    rms_biases = Matrix2d();
    rmsn = Matrix2d();
    rmsn_biases = Matrix2d();
}

Layer::Layer(int nneurons, int nprevneurons, ActFuncTypes act_func) {
    activation_func = act_func;
    num_prev_neurons = nprevneurons;
    num_neurons = nneurons;

    weights = Matrix2d(nneurons, nprevneurons);
    weights.InitRandom(0.0, 1.0);
    states = Matrix2d(1, nneurons);
    biases = Matrix2d(1, nneurons);
    biases.InitRandom(-1.0, 1.0);
    axons = Matrix2d(1, nneurons);
    delta = Matrix2d(1, nneurons);
    grad = Matrix2d(nneurons, nprevneurons);

    prev_delta = Matrix2d(1, nneurons);
    delta_sum = Matrix2d(1, nneurons);
    prev_delta_sum = Matrix2d(1, nneurons);
    prev_grad = Matrix2d(nneurons, nprevneurons);
    grad_sum = Matrix2d(nneurons, nprevneurons);
    prev_grad_sum = Matrix2d(nneurons, nprevneurons);

    weights_correct = Matrix2d(nneurons, nprevneurons);
    weights_correct.InitRandom(0.0, 1.0);
    biases_correct = Matrix2d(1, nneurons);
    biases_correct.InitRandom(-1.0, 1.0);

    rms = Matrix2d(nneurons, nprevneurons);
    rms_biases = Matrix2d(1, nneurons);
    rmsn = Matrix2d(nneurons, nprevneurons);
    rmsn_biases = Matrix2d(1, nneurons);
}

Layer& Layer::operator = (Layer& _l) {
    activation_func = _l.activation_func;
    num_prev_neurons = _l.num_prev_neurons;
    num_neurons = _l.GetNumNeurons();
    weights = _l.weights;
    states = _l.states;
    biases = _l.biases;
    axons = _l.axons;
}

```

Инв.№ подл.	
Взам. инв. №	
Инв. № дубл.	
Подп. и дата	

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

Лист

76

```

delta = _l.delta;
grad = _l.grad;

prev_delta = _l.prev_delta;
delta_sum = _l.delta_sum;
prev_delta_sum = _l.prev_delta_sum;
prev_grad = _l.prev_grad;
grad_sum = _l.grad_sum;
prev_grad_sum = _l.prev_grad_sum;

weights_correct = _l.weights_correct;
biases_correct = _l.biases_correct;

rms = _l.rms;
rms_biases = _l.rms_biases;
rmsn = _l.rmsn;
rmsn_biases = _l.rmsn_biases;

return *this;
}
void Layer::Init(int nneurons, int nprevneurons, ActFuncTypes act_func, Matrix2d _biases,
Matrix2d _weights) {
activation_func = act_func;
num_prev_neurons = nprevneurons;
num_neurons = nneurons;
weights = _weights;
states = Matrix2d(1, nneurons);
biases = _biases;
axons = Matrix2d(1, nneurons);
delta = Matrix2d(1, nneurons);
grad = Matrix2d(nneurons, nprevneurons);

prev_delta = Matrix2d(1, nneurons);
delta_sum = Matrix2d(1, nneurons);
prev_delta_sum = Matrix2d(1, nneurons);
prev_grad = Matrix2d(nneurons, nprevneurons);
grad_sum = Matrix2d(nneurons, nprevneurons);
prev_grad_sum = Matrix2d(nneurons, nprevneurons);

weights_correct = Matrix2d(nneurons, nprevneurons);
weights_correct.InitRandom(0.0, 1.0);
biases_correct = Matrix2d(1, nneurons);
biases_correct.InitRandom(-1.0, 1.0);

rms = Matrix2d(nneurons, nprevneurons);
rms_biases = Matrix2d(1, nneurons);
rmsn = Matrix2d(nneurons, nprevneurons);
rmsn_biases = Matrix2d(1, nneurons);
}
int Layer::GetNumNeurons() {
return num_neurons;
}
void Layer::CalcStates(Layer& prev_layer) {
states = prev_layer.axons * weights.Transpose() + biases;
}
void Layer::CalcAxons() {
switch (activation_func) {
case TANH: axons = tanh_func(states); break;
case SGMD: axons = sgmd_func(states); break;
case LINE: axons = states; break;
default: break;
}
}
}

```

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

```

Matrix2d Layer::CalcDiffs() {
    switch (activation_func) {
        case TANH: return diff_tanh_func(axons); break;
        case SGMD: return diff_sgmd_func(axons); break;
        case LINE:
        {
            Matrix2d m;
            m.InitMatrix(axons.GetNumRows(), axons.GetNumCols(), 1.0);
            return m;
            break;
        }
        default: break;
    }
}

random_device rnd_dev;
mt19937 eng = mt19937(rnd_dev());
uniform_int_distribution<> rnd_gen = uniform_int_distribution<>(0, RAND_MAX);

int SomeRand() {
    return rnd_gen(eng);
}

double GetRandVal(double min_val, double max_val) {
    return min_val + (max_val - min_val) * SomeRand() / RAND_MAX;
}

Matrix2d::~~Matrix2d()
{
    if (num_rows != 0)
        delete[](matrix);
    matrix = nullptr;
}

Matrix2d::Matrix2d(int nrow, int ncol, double val) {
    num_rows = nrow;
    num_cols = ncol;
    matrix = new double[nrow*ncol];
    for (int i = 0; i < num_rows*num_cols; ++i) {
        matrix[i] = val;
    }
}

Matrix2d::Matrix2d(const Matrix2d& _m) {
    num_rows = _m.num_rows;
    num_cols = _m.num_cols;
    matrix = new double[num_rows*num_cols];
    memcpy_s(matrix, num_rows*num_cols * sizeof(*matrix), _m.matrix, num_rows*num_cols *
sizeof(*_m.matrix));
}

Matrix2d::Matrix2d(vector<double>& _m) {
    num_rows = 1;
    num_cols = _m.size();
    matrix = new double[num_cols];
    memcpy_s(matrix, num_rows*num_cols * sizeof(*matrix), _m.data(), num_rows*num_cols *
sizeof(*_m.data()));
}

void Matrix2d::InitMatrix(double val) {
    for (int i = 0; i < num_rows; ++i) {
        for (int j = 0; j < num_cols; ++j) {

```

Подп. и дата

Инв. № дубл.

Взам. инв. №

Подп. и дата

Инв. № подл.

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

Лист

78

```

        at(i, j) = val;
    }
}

void Matrix2d::InitMatrix(int nrow, int ncol, double val) {
    num_rows = nrow;
    num_cols = ncol;
    matrix = new double[nrow*ncol];
    for (int i = 0; i < num_rows*num_cols; ++i) {
        matrix[i] = val;
    }
}

void Matrix2d::InitMatrixDiagByOne() {
    int n = min(num_rows, num_cols);
    for (int i = 0; i < n; ++i) {
        at(i, i) = 1.0;
    }
}

void Matrix2d::InitRandom(double minval, double maxval) {
    for (int i = 0; i < num_rows; ++i) {
        for (int j = 0; j < num_cols; ++j) {
            at(i, j) = GetRandVal(minval, maxval);
        }
    }
}

void Matrix2d::InitValue(double val) {
    for (int i = 0; i < num_rows; ++i) {
        for (int j = 0; j < num_cols; ++j) {
            at(i, j) = val;
        }
    }
}

int Matrix2d::GetNumRows() const {
    return num_rows;
}

int Matrix2d::GetNumCols() const {
    return num_cols;
}

// Транспонирование матрицы
Matrix2d Matrix2d::Transpose() const {
    int n = GetNumCols();
    int m = GetNumRows();
    Matrix2d res(n, m);
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            res.at(i, j) = at(j, i);
        }
    }
    return move(res);
}

// Поэлементное умножение матриц
Matrix2d Matrix2d::MultElByEl(const Matrix2d& _m) const {
    if (_m.GetNumRows() != num_rows || _m.GetNumCols() != num_cols) {
        throw logic_error("Wrong sizes in multiplication operation");
    }
}

```

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18



```

Matrix2d res(num_rows, num_cols);
for (int i = 0; i < num_rows; ++i) {
    for (int j = 0; j < num_cols; ++j) {
        res.at(i, j) = at(i, j) * _m.at(i, j);
    }
}
return move(res);
}

// Сложение квадратов всех элементов матрицы
double Matrix2d::SumAllSqr() const {
    double res = 0.0;
    for (int i = 0; i < num_rows; ++i) {
        for (int j = 0; j < num_cols; ++j) {
            res += at(i, j) * at(i, j);
        }
    }
    return res;
}

// Сложение всех элементов матрицы
double Matrix2d::SumAll() const {
    double res = 0.0;
    for (int i = 0; i < num_rows; ++i) {
        for (int j = 0; j < num_cols; ++j) {
            res += at(i, j);
        }
    }
    return res;
}

Matrix2d Matrix2d::Sqrt() {
    Matrix2d res(num_rows, num_cols);
    for (int i = 0; i < num_rows; ++i) {
        for (int j = 0; j < num_cols; ++j) {
            res.at(i, j) = sqrt(fabs(at(i, j)));
        }
    }
    return move(res);
}

double& Matrix2d::at(const int& i, const int& j) {
    return matrix[i*num_cols + j];
}

double Matrix2d::at(const int& i, const int& j) const {
    return matrix[i*num_cols + j];
}

Matrix2d& Matrix2d::operator = (vector<double>& _m) {
    delete[] (matrix);
    num_rows = 1;
    num_cols = _m.size();
    matrix = new double[num_rows*num_cols];
    memcpy_s(matrix, num_rows*num_cols * sizeof(*matrix), _m.data(), num_rows*num_cols *
sizeof(*_m.data()));
    return *this;
}

Matrix2d& Matrix2d::operator = (vector<vector<double>>& _m) {

```

Инв.№ подл.	Взам. инв. №	Инв.№ дубл.	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата

БР-02069964-090301-16-18

Лист

80

```

delete[](matrix);
num_rows = _m.size();
num_cols = (num_rows != 0) ? _m[0].size() : 0;
matrix = new double[num_rows*num_cols];
for (int i = 0; i < num_rows; ++i) {
    for (int j = 0; j < num_cols; ++j) {
        at(i, j) = _m[i][j];
    }
}

return *this;
}

Matrix2d& Matrix2d::operator = (Matrix2d& _m) {
delete[](matrix);
num_rows = _m.GetNumRows();
num_cols = _m.GetNumCols();
matrix = new double[num_rows*num_cols];
memcpy_s(matrix, num_rows*num_cols * sizeof(*matrix), _m.matrix, num_rows*num_cols *
sizeof(*_m.matrix));

return *this;
}

// Сложение матрицы с числом
Matrix2d& Matrix2d::operator += (const double val) {
for (int i = 0; i < num_rows; ++i) {
for (int j = 0; j < num_cols; ++j) {
at(i, j) += val;
}
}

return *this;
}

// Сложение матрицы с числом
Matrix2d Matrix2d::operator + (const double val) {
Matrix2d res = *this;
res += val;
return move(res);
}

// Сложение матриц
Matrix2d& Matrix2d::operator += (Matrix2d& _m) {
if (_m.GetNumRows() != num_rows || _m.GetNumCols() != num_cols) {
//Error!
throw logic_error("Wrong sizes in addition operation");
}

for (int i = 0; i < num_rows; ++i) {
for (int j = 0; j < num_cols; ++j) {
at(i, j) += _m.at(i, j);
}
}

return *this;
}

// Сложение матриц
Matrix2d Matrix2d::operator + (Matrix2d& _m) {
Matrix2d res = *this;
res += _m;
return move(res);
}

```

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

```

}

// Вычитание числа из матрицы
Matrix2d& Matrix2d::operator -= (const double val) {
    for (int i = 0; i < num_rows; ++i) {
        for (int j = 0; j < num_cols; ++j) {
            at(i, j) -= val;
        }
    }

    return *this;
}

// Вычитание числа из матрицы
Matrix2d Matrix2d::operator - (const double val) {
    Matrix2d res = *this;
    res -= val;
    return move(res);
}

// Вычитание матриц
Matrix2d& Matrix2d::operator -= (Matrix2d& _m) {
    if (_m.GetNumRows() != num_rows || _m.GetNumCols() != num_cols) {
        //Error!
        throw logic_error("Wrong sizes in subtraction operation");
    }

    for (int i = 0; i < num_rows; ++i) {
        for (int j = 0; j < num_cols; ++j) {
            at(i, j) -= _m.at(i, j);
        }
    }

    return *this;
}

// Вычитание матриц
Matrix2d Matrix2d::operator - (Matrix2d& _m) {
    Matrix2d res = *this;
    res -= _m;
    return move(res);
}

// Умножение матрицы на число
Matrix2d& Matrix2d::operator *= (const double val) {
    for (int i = 0; i < num_rows; ++i) {
        for (int j = 0; j < num_cols; ++j) {
            at(i, j) *= val;
        }
    }

    return *this;
}

// Умножение матрицы на число
Matrix2d Matrix2d::operator * (const double val) {
    Matrix2d res = *this;
    res *= val;
    return move(res);
}

// Произведение матриц
Matrix2d Matrix2d::operator * (Matrix2d& _m) {

```

Инв.№ подл.	Взам. инв. №	Инв. № дубл.	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата

БР-02069964-090301-16-18

```

if (_m.GetNumRows() != num_cols) {
    //Error!
    throw logic_error("Wrong sizes in multiplication operation");
}

Matrix2d res(num_rows, _m.GetNumCols());
int _mnum_cols = _m.GetNumCols();
for (int i = 0; i < num_rows; ++i) {
    for (int j = 0; j < _mnum_cols; ++j) {
        res.at(i, j) = 0.0;
        for (int k = 0; k < num_cols; ++k) {
            res.at(i, j) += at(i, k) * _m.at(k, j);
        }
    }
}

return move(res);
}

// Деление элементов матрицы на число
Matrix2d& Matrix2d::operator /= (const double val) {
    for (int i = 0; i < num_rows; ++i) {
        for (int j = 0; j < num_cols; ++j) {
            at(i, j) /= val;
        }
    }

    return *this;
}

// Деление элементов матрицы на число
Matrix2d Matrix2d::operator / (const double val) {
    Matrix2d res = *this;
    res /= val;
    return move(res);
}

Matrix2d operator / (const double val, const Matrix2d& _m) {
    Matrix2d res(_m.num_rows, _m.num_cols);
    for (int i = 0; i < _m.num_rows; ++i) {
        for (int j = 0; j < _m.num_cols; ++j) {
            res.at(i, j) = val / _m.at(i, j);
            if (isinf(res.at(i, j)) || isnan(res.at(i, j)))
                res.at(i, j) = DBL_MAX;
        }
    }

    return move(res);
}

void Creature::InitCreature(vector<pair<double, double>> _joints, vector<pair<int, int>>
_lines, vector<pair<int, int>> _mvlines,
vector<pair<double, double>> _turnint, vector<pair<int, int>> states,
vector<vector<int>> _refs, vector<int> _head_points) {
    joints = _joints;
    lines = _lines;

    for (int i = 0; i < _lines.size(); ++i) {
        double d = GetDistance(joints[lines[i].first].first,
joints[lines[i].first].second, joints[lines[i].second].first,
joints[lines[i].second].second);
        lines_length.push_back(d);
    }
}

```

Подп. и дата

Инв. № дубл.

Взам. инв. №

Подп. и дата

Инв. № подл.

БР-02069964-090301-16-18

Лист

Изм Лист № докум. Подпись Дата

83

```

movable_lines = _mvlines;
turn_intervals = _turnint;
states_mvlines = states;
refs = _refs;
head_points = _head_points;

start_pos = GetCenterOfGravityX();
start_max_x = -DBL_MAX;
for (int i = 0; i < joints.size(); ++i) {
    if (joints[i].first > start_max_x) {
        start_max_x = joints[i].first;
    }
}

void Creature::SetJoints(vector<pair<double, double>> new_joints) {
    for (int i = 0; i < joints.size(); ++i){
        joints[i] = new_joints[i];
    }
}

void Creature::SetStates(vector<int> new_states) {
    for (int i = 0; i < states_mvlines.size(); ++i) {
        if (new_states[i] > states_mvlines[i].second - 1) return;
        if (new_states[i] == -1) {
            for (int j = 0; j < movable_lines.size(); ++j) {
                if (i == movable_lines[j].first)
                    return;
            }
            continue;
        }
        if (new_states[i] < 0)
            return;
    }
    for (int i = 0; i < states_mvlines.size(); ++i) {
        states_mvlines[i].first = new_states[i];
    }
}

vector<Line> Creature::GetLines() {
    vector<Line> res;

    for (int i = 0; i < lines.size(); ++i) {
        Point p1 = { joints[lines[i].first].first, joints[lines[i].first].second };
        Point p2 = { joints[lines[i].second].first, joints[lines[i].second].second };
        res.push_back({ p1, p2, lines_length[i] });
    }
    return res;
}

double Creature::GetCenterOfGravityX() {
    double sum_momets = 0.0;
    double sum_mass = 0.0;
    double res = 0.0;

    for (int i = 0; i < lines.size(); ++i) {
        double cx = 1.0*(joints[lines[i].first].first + joints[lines[i].second].first) / 2;
        // координата x центра тяжести
        sum_mass += lines_length[i]; // lines_length[i] - длина отрезка, и т.к. 1 ед.
        // длины = 1 ед. массы, то используем длину
        sum_momets += cx*lines_length[i]; // плечо на массу
    }
}

```

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

```

        return res = sum_momets / sum_mass;
    }

    double Creature::GetCenterOfGravityY() {
        double sum_momets = 0.0;
        double sum_mass = 0.0;
        double res = 0.0;

        for (int i = 0; i < lines.size(); ++i) {
            double cx = 1.0*(joints[lines[i].first].second +
joints[lines[i].second].second) / 2;
            sum_mass += lines_length[i]; // lines_length[i] - длина отрезка, и т.к. 1 ед.
длинны = 1 ед. массы, то используем длину
            sum_momets += cx*lines_length[i]; // плечо на массу
        }
        return res = sum_momets / sum_mass;
    }

    double Creature::GetCurDeltaDistance() {
        double cx = GetCenterOfGravityX();
        return cx - start_pos;
    }

    double Creature::GetTraveledDistance() {
        double tmp = -DBL_MAX;
        for (int i = 0; i < joints.size(); ++i) {
            if (joints[i].first > tmp) {
                tmp = joints[i].first;
            }
        }
        return tmp - start_max_x;
    }

    double Creature::GetHeadY()
    {
        double minY = DBL_MAX;
        for (int i = 0; i < head_points.size(); ++i) {
            minY = min(minY, joints[head_points[i]].second);
        }

        return minY;
    }

    pair<int, int> Creature::GetAction(int action_num) {
        int line = action_num / 2;
        int turn_dir;

        if (action_num % 2 == 0)
            turn_dir = 1;
        else
            turn_dir = -1;

        return make_pair(line, turn_dir);
    }

    bool Creature::CanDoAction(int action) {
        pair<int, int> line_and_dir = GetAction(action);

        //Если отрезок находится в одном из крайних состояний
        if (states_mvlines[movable_lines[line_and_dir.first].first].first <= 0 &&
line_and_dir.second == 1)
            return false;
    }

```

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

```

        if (states_mvlines[movable_lines[line_and_dir.first].first].first >=
(states_mvlines[movable_lines[line_and_dir.first].first].second - 1)
            && line_and_dir.second == -1)
            return false;

        return true;
    }

bool Creature::CanDoFullAction(int action) {
    pair<int, int> line_and_dir = GetAction(action);

    int line = line_and_dir.first;
    int tdir = line_and_dir.second;
    double angsign = 0.0;

    // Направление поворота
    if (tdir == 1)
        angsign = -1.0;
    if (tdir == -1)
        angsign = 1.0;

    int mvline = movable_lines[line].first;

    double unit_angle = turn_unit_angle;

    // Перемещение
    set<int> points;
    int os = movable_lines[line].second;

    // концы отрезка, который поворачивается
    if (lines[mvline].first != os)
        points.insert(lines[mvline].first);
    if (lines[mvline].second != os)
        points.insert(lines[movable_lines[line].first].second);

    for (int i = 0; i < refs[mvline].size(); ++i) {
        points.insert(lines[refs[mvline][i]].first);
        points.insert(lines[refs[mvline][i]].second);
    }

    for (auto it = points.begin(); it != points.end(); ++it) {
        double d = GetDistance(joints[os].first, joints[os].second, joints[*it].first,
joints[*it].second);
        double angle = GetAngle(joints[os].first, joints[os].second,
joints[*it].first, joints[*it].second) + angsign*unit_angle;
        double new_y = joints[os].second + d*sin(angle);
        if (new_y < 0.0)
            return false;
    }

    return true;
}

void Creature::Falling() {
    double cg = GetCenterOfGravityX();
    vector<pair<pair<double, double>, int>> sup_points; // точки опоры
    double err_y = 0.0000001; // допустимое расстояние от земли, при котором касание еще
существует

    for (int i = 0; i < joints.size(); ++i) {
        if (joints[i].second <= err_y) {

```

Инв.№ подл.	Взам. инв. №	Инв. № дубл.	Подп. и дата

БР-02069964-090301-16-18

Лист

86

Изм	Лист	№ докум.	Подпись	Дата

```

        sup_points.push_back(make_pair(joints[i], i));
    }
}

// Если после поворота некоторого отрезка, существо оказалось в воздухе
if (sup_points.size() == 0) {
    double ymn = 1.0*1e9;
    for (int i = 0; i < joints.size(); ++i) {
        if (joints[i].second < ymn) {
            ymn = joints[i].second;
        }
    }

    for (int i = 0; i < joints.size(); ++i) {
        joints[i].second -= ymn;
    }

    for (int i = 0; i < joints.size(); ++i) {
        if (joints[i].second <= err_y) {
            sup_points.push_back(make_pair(joints[i], i));
        }
    }
}

sort(sup_points.begin(), sup_points.end());

int fall_flag = 0; // balance
int turn_point;

if (sup_points[0].first.first > cg) { // left
    turn_point = sup_points[0].second;
    fall_flag = -1;
}
else if (sup_points[sup_points.size() - 1].first.first < cg) { //right
    turn_point = sup_points[0].second;
    fall_flag = 1;
}

if (fall_flag != 0) {
    falling = 0.0;

    double angle_sign = 0.0;
    if (fall_flag == 1) angle_sign = -1.0;
    else angle_sign = 1.0;

    vector<pair<double, double>> new_pos;
    vector<double> err_angles;
    for (int i = 0; i < joints.size(); ++i) {
        if (i == turn_point) continue;
        double d = GetDistance(joints[turn_point].first,
joints[turn_point].second, joints[i].first, joints[i].second);
        double angle = GetAngle(joints[turn_point].first,
joints[turn_point].second, joints[i].first, joints[i].second);
        double x = joints[turn_point].first + d*cos(angle +
angle_sign*fall_unit_angle);
        double y = joints[turn_point].second + d*sin(angle +
angle_sign*fall_unit_angle);
        if (y < 0.0) {
            double tmp_a = GetAngle(joints[turn_point].first,
joints[turn_point].second, joints[i].first + 1.0*fall_flag*d, 0.0);

            angle = GetEquivPositiveAngle(angle);

```

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

Лист

87



```

        tmp_a = GetEquivPositiveAngle(tmp_a);

        err_angles.push_back(fabs(angle - tmp_a));
    }
    new_pos.push_back(make_pair(x, y));
}

if (!err_angles.size()) {
    for (int i = 0, j = 0; i < joints.size(); ++i) {
        if (i == turn_point) continue;
        joints[i].first = new_pos[j].first;
        joints[i].second = new_pos[j].second;
        j++;
    }
}
else {
    sort(err_angles.begin(), err_angles.end());
    for (int i = 0; i < joints.size(); ++i) {
        if (i == turn_point) continue;
        double d = GetDistance(joints[turn_point].first,
joints[turn_point].second, joints[i].first, joints[i].second);
        double angle = GetAngle(joints[turn_point].first,
joints[turn_point].second, joints[i].first, joints[i].second) + angle_sign*err_angles[0];
        joints[i].first = joints[turn_point].first + d*cos(angle);
        joints[i].second = joints[turn_point].second + d*sin(angle);
    }
}
}
else {
    falling = min(fabs(CG - sup_points[0].first.first), fabs(CG -
sup_points.back().first.first));
}
}

int Creature::Rotate(int line, int tdir) {
    double angsign = 0.0;
    int mvline = movable_lines[line].first;
    // Обновление состояний
    if (tdir == 1) {
        states_mvlines[mvline].first--;
        angsign = -1.0;
    }
    if (tdir == -1) {
        states_mvlines[mvline].first++;
        angsign = 1.0;
    }
}

double unit_angle = turn_unit_angle;

set<int> points;
int os = movable_lines[line].second;
// концы отрезка, который поворачивается
if (lines[mvline].first != os)
    points.insert(lines[mvline].first);
if (lines[mvline].second != os)
    points.insert(lines[mvline].second);

for (int i = 0; i < refs[mvline].size(); ++i) {
    points.insert(lines[refs[mvline][i]].first);
    points.insert(lines[refs[mvline][i]].second);
}
}

```

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

Лист

88

```

double ymn = DBL_MAX;
int minpoint = -1;
for (auto it = points.begin(); it != points.end(); ++it) {
    double d = GetDistance(joints[os].first, joints[os].second, joints[*it].first,
joints[*it].second);
    double angle = GetAngle(joints[os].first, joints[os].second,
joints[*it].first, joints[*it].second) + angsign*unit_angle;
    joints[*it].first = joints[os].first + d*cos(angle);
    joints[*it].second = joints[os].second + d*sin(angle);
    if (joints[*it].second < ymn) {
        ymn = joints[*it].second;
        minpoint = *it;
    }
}

return minpoint;
}

void Creature::CorrectPos(int line, int tdir) {
    int mvline = movable_lines[line].first;
    int os = movable_lines[line].second;
    int nos;
    if (lines[mvline].first != os) // находим точку отрезка, которая НЕ является суставом
поворота
        nos = lines[mvline].first;
    else
        nos = lines[mvline].second;

    double dy = fabs(joints[nos].second);
    double dx = 0.0;
    double x1 = joints[os].first - sqrt(fabs(lines_length[mvline] * lines_length[mvline]
- joints[os].second*joints[os].second));
    double x2 = joints[os].first + sqrt(fabs(lines_length[mvline] * lines_length[mvline]
- joints[os].second*joints[os].second));

    if (tdir == 1) {
        dx = fabs(max(x1, x2) - joints[nos].first);
    }
    else if (tdir == -1) {
        dx = fabs(min(x1, x2) - joints[nos].first);
    }
}

for (int i = 0; i < joints.size(); ++i) {
    joints[i].first += 1.0*tdir*dx;
    joints[i].second += dy;
}
}

void Creature::CorrectPos(int line, int tdir, int point) {
    int mvline = movable_lines[line].first;
    int os = movable_lines[line].second;

    double dy = fabs(joints[point].second);
    double dx = 0.0;
    double dd = GetDistance(joints[os].first, joints[os].second, joints[point].first,
joints[point].second);
    double x1 = joints[os].first - sqrt(fabs(dd * dd -
joints[os].second*joints[os].second));
    double x2 = joints[os].first + sqrt(fabs(dd * dd -
joints[os].second*joints[os].second));

    if (tdir == 1) {
        dx = fabs(max(x1, x2) - joints[point].first);
    }
}

```

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

Лист

89

```

    }
    else if (tdir == -1) {
        dx = fabs(min(x1, x2) - joints[point].first);
    }

    for (int i = 0; i < joints.size(); ++i) {
        joints[i].first += tdir*dx;
        joints[i].second += dy;
    }
}

void Creature::UpdatePos(int action_num) {
    pair<int, int> p = GetAction(action_num); // номер отрезка и направление поворота (1 -
по ч.с., 2 - против ч.с.)
    int line = p.first;
    int mvline = movable_lines[line].first;
    int tdir = p.second;
    int angsign = 0;

    // Перемещение
    if (CanDoFullAction(action_num)) { // Если можно совершить полный поворот отрезком
        Rotate(line, tdir);
    }
    else {
        if (refs[mvline].size() == 0) {
            Rotate(line, tdir);
            CorrectPos(line, tdir);        }
        else {
            int point = Rotate(line, tdir);
            CorrectPos(line, tdir, point);
        }
    }

    // Падение
    Falling();
}

```

### Листинг А.3 — Реализация алгоритма Q-Learning

```

double GetReward() {
    double res = 0.0;
    for (int i = 0; i < used_reward.size(); ++i) {
        if (used_reward[i]) {
            res += GetReward(i);
        }
    }
    return res;
}

double GetReward(int rew_type) {
    switch (rew_type)
    {
    case ALL_DIST:
        return fabs(monster.GetCurDeltaDistance());
    case PREV_STEP_DIST:
        return fabs(prev_dist - monster.GetCurDeltaDistance());
    case CENTER_OF_GRAVITY_Y:
        return -k_reward["CENTER_OF_GRAVITY_Y"] / max(1.0,
monster.GetCenterOfGravityY());
    case FALLING:

```

Инв.№ подл.	Подп. и дата
Взам. инв. №	Инв. № дубл.
Подп. и дата	Подп. и дата

БР-02069964-090301-16-18

Лист

90

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

```

        return -k_reward["FALLING"] / max(1.0, monster.GetFalling());
    case HEAD_Y:
        return -k_reward["HEAD_Y"] / max(1.0, monster.GetHeadY());
    default:
        return 0.0;
        break;
    }
}

void DoNextStep() {
    prev_inputs = inputs;
    SetInputs(inputs);
    int action = -1;
    reward = 0.0;
    reward = GetReward();
    prev_dist = monster.GetCurDeltaDistance();
    cur_tick++;

    if (!firstStep) {
        nnet.Running(inputs);
        Q = nnet.GetOutput();

        double tmpQ = -DBL_MAX;
        for (int i = 0; i < Q.GetNumCols(); ++i) {
            if (tmpQ < Q.at(0, i) && monster.CanDoAction(i)) {
                tmpQ = Q.at(0, i);
                action = i;
            }
        }
        Q.at(0, prevAction) = reward + QGAMMA*tmpQ;

        nnet.AddTest(tests, prev_inputs, Q);
        int epoch = EPOCH;
        vector<int> tests_pos;
        for (int i = 0; i < min(CUR_TESTS_NUMBER, tests.size()); ++i) {
            int pos = max(0, (tests.size() - 1))*(double)rand() / RAND_MAX;
            tests_pos.push_back(pos);
        }

        while (epoch--) {
            if (run_type != RUN) {
                if (nnet.RMSLearningOffline(tests, tests_pos) < TRAIN_EPS)
                    break;
            }
        }
    }
    else {
        nnet.Running(inputs);
        Q = nnet.GetOutput();

        double tmpQ = -DBL_MAX;
        for (int i = 0; i < Q.GetNumCols(); ++i) {
            if ((tmpQ < Q.at(0, i)) && monster.CanDoAction(i)) {
                tmpQ = Q.at(0, i);
                action = i;
            }
        }

        firstStep = false;
    }

    monster.UpdatePos(action);
}

```

Подп. и дата	
Инв. № дудл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

Изм	Лист	№ докум.	Подпись	Дата
-----	------	----------	---------	------

БР-02069964-090301-16-18

Лист

91

```

if (run_type != RUN) {
    if (1.0*rand() / RAND_MAX) < 0.03) {
        int counter = 100;
        bool flag_do = false;
        do {
            action = (monster.GetNumActions() - 1)*rand() / RAND_MAX;
            if (monster.CanDoAction(action)) {
                flag_do = true;
                break;
            }
        } while (counter--);
        if (flag_do)
            monster.UpdatePos(action);
    }
}
prevAction = action;
prevQ = Q;

//Вывод текущей информации
cout << cur_tick << " " << monster.GetCurDeltaDistance() << endl;

if (cur_tick % T == 0) {
    string dirname = res_dir_str + nnet_name;
    ofstream wbfout(dirname + "\\ " + nnet_name + wb_filename_end);
    nnet.PrintWeightsAndBiases(wbfout, false);
    wbfout.close();

    ofstream crfout(dirname + "\\ " + nnet_name + curcr_filename_end);
    monster.PrintCreatureJoints(crfout);
    crfout.close();
}

fflush(stdout);
}

```

Инв.№ подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

Изм	Лист	№ докум.	Подпись	Дата

БР-02069964-090301-16-18

Лист

92

## Приложение Б

(обязательное)

### Графический материал

- 1 Модель сложного объекта. Процесс создания модели объекта;
- 2 Структура нейронной сети. Формулы расчета вознаграждения;
- 3 Диаграмма классов искусственной нейронной сети и модели объекта.

Схема взаимодействия модулей приложения;

- 4 Анализ эффективности;
- 5 Анализ эффективности перемещения различных моделей.

Инв.№ подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата	Инв.№ подл.	Подп. и дата	Изм	Лист	№ докум.	Подпись	Дата	Лист	93

Инв.№ подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

# Модель сложного объекта. Процесс создания модели объекта



- — Сустав модели объекта
- Отрезок модели объекта
- ⊕ — Центр тяжести модели объекта

Рисунок 1 — Модель сложного объекта

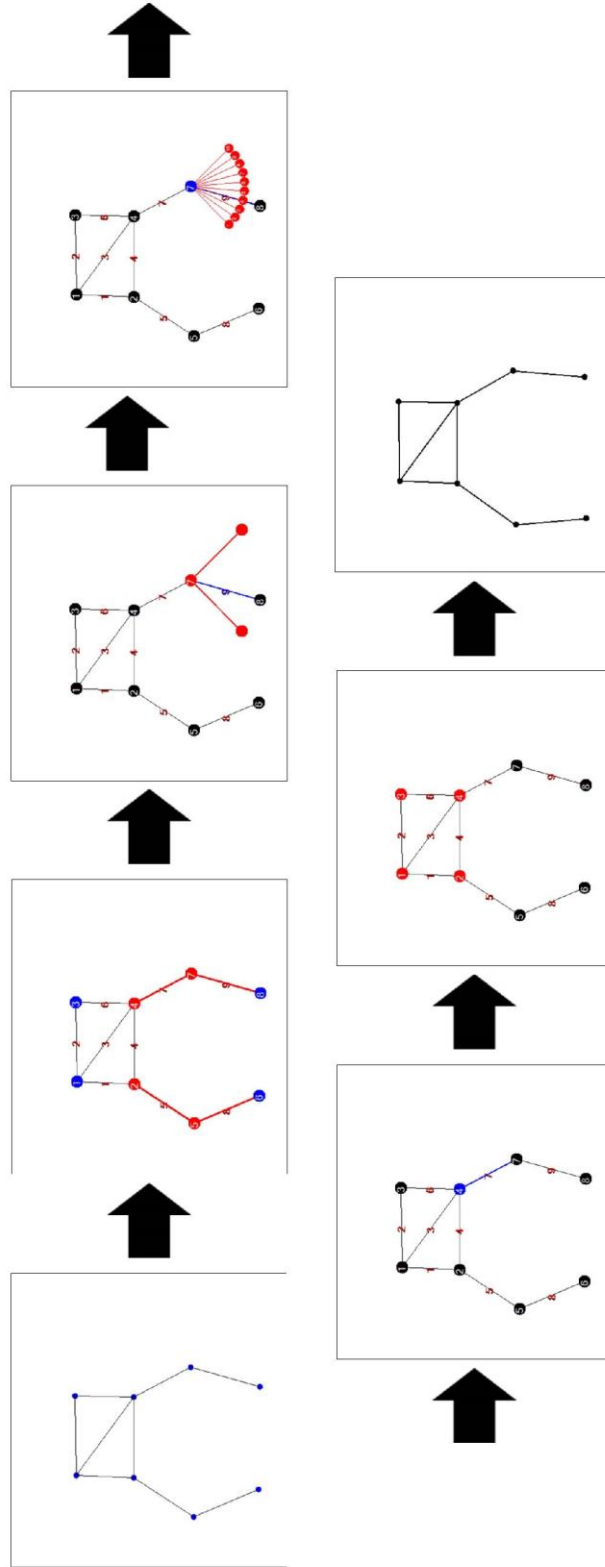
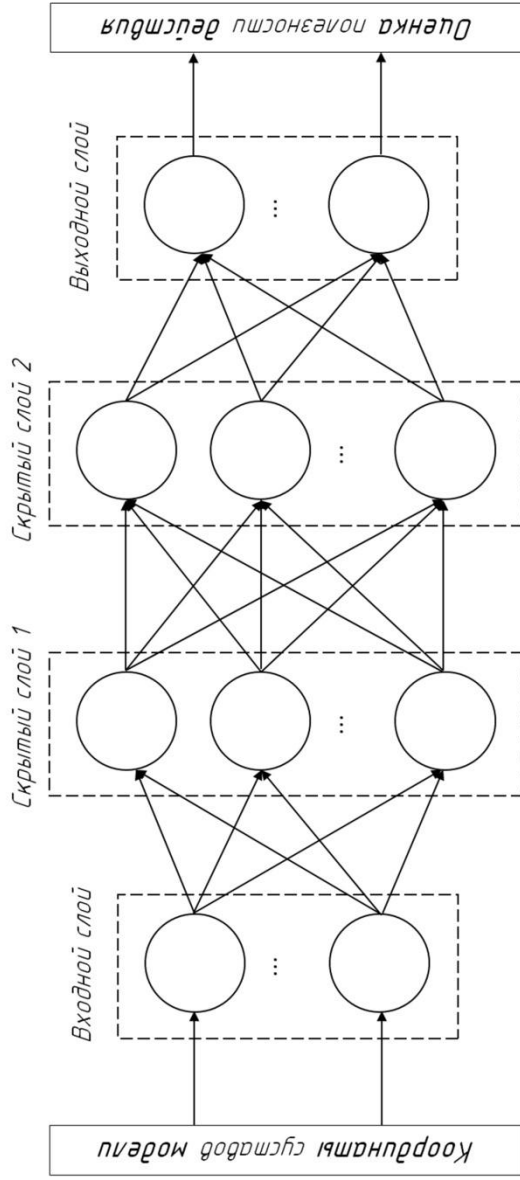


Рисунок 2 — Процесс создания модели объекта

Инв.№ подл.		Подп. и дата		Взам. инв. №		Инв. № дубл.		Подп. и дата	
БР-02069964-090301-16-18									
Исполнитель: инженер по АСУ									
Муницип. гос. предприятие:									
Производство: скважина									
Объект: 16.002147									
Город: Ижевск									
Улицы: 3.444098									
Итого: 116.002147									
Итого: 4									
Итого: 3.444098									
Итого: 116.002147									
Итого: 4									
Итого: 3.444098									

Инв.№ подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

## Структура нейронной сети. Формулы расчета вознаграждения



$$Reward_t = \alpha_1 \cdot AllDist_t + \alpha_2 \cdot PrevStepDist_t + \alpha_3 \cdot CenterOfGravity_t + \alpha_4 \cdot Falling_t + \alpha_5 \cdot Head_t$$

$$AllDist_t = | CurrentPositionX_t - StartPositionX_t |$$

$$PrevStepDist_t = | AllDist_t - AllDist_{t-1} |$$

$$CenterOfGravity_t = - k_1 / CenterOfGravityY_t$$

$$Falling_t = - k_2 / MinDistToSupportPoint_t$$

$$Head_t = - k_3 / MinHeadY_t$$

Инв.№ подл.		Подп. и дата		Взам. инв. №		Инв. № дубл.		Подп. и дата	

БР-02069964-090301-16-18

Исходный материал на тему: "Исследования по разработке нейронной сети, реализующей процесс поиска оптимального маршрута в пространстве"

Составитель: [Имя Фамилия И.О.]  
 Проверил: [Имя Фамилия И.О.]  
 Руководитель: [Имя Фамилия И.О.]



Инв.№ подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

Ил. № - 02069964 - 090301 - 16 - 18

# Диаграмма классов искусственной нейронной сети и модели объекта. Схема взаимодействия модулей приложения

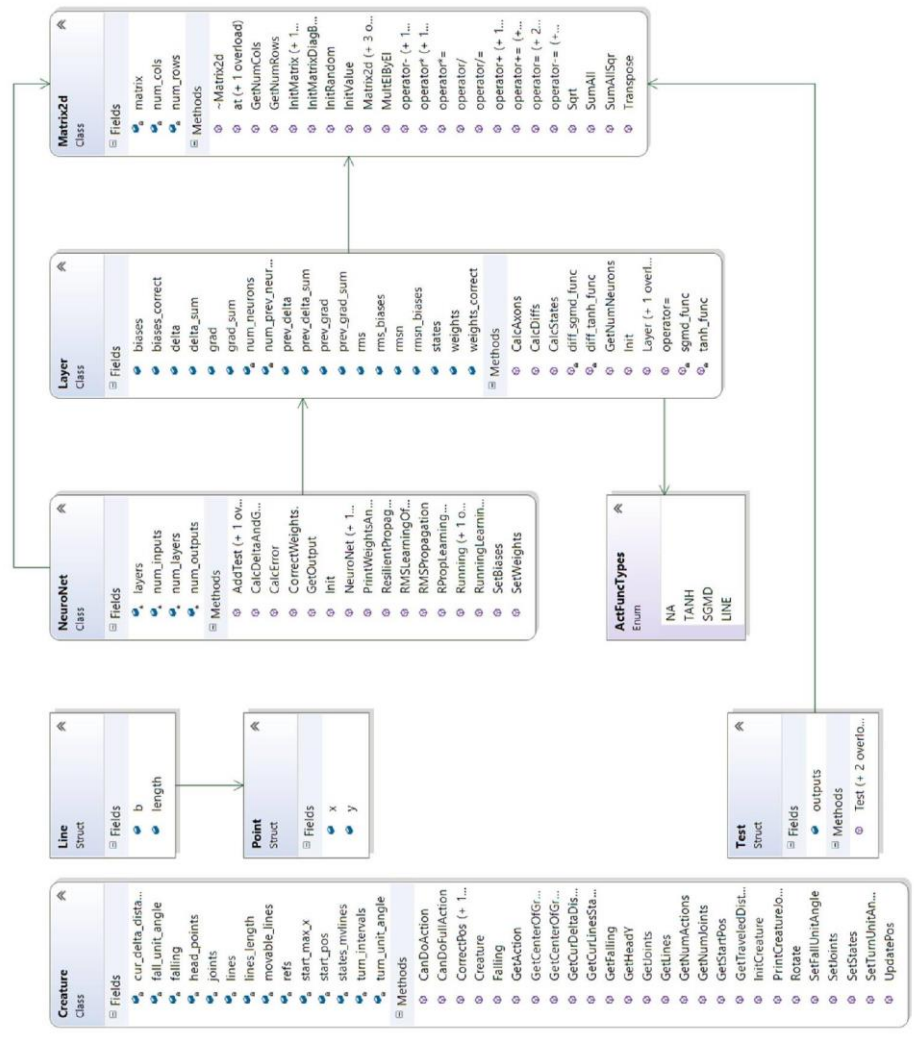


Рисунок 2 – Схема взаимодействия модулей приложения

Рисунок 1 – Диаграмма классов искусственной нейронной сети и модели объекта

ИЗМ.	№	ПОДПИСЬ	ДАТА	ОБЪЕКТ

Исполнитель: **БР-02069964-090301-16-18**

Модуль: **Искусственная нейронная сеть**

Версия: **1.0**

Дата: **2018.03.01**

Адрес: **г. Москва, ул. Давыдовская, д. 15, стр. 1**

Контакт: **тел. +7 (495) 980-00-00**

## Анализ эффективности

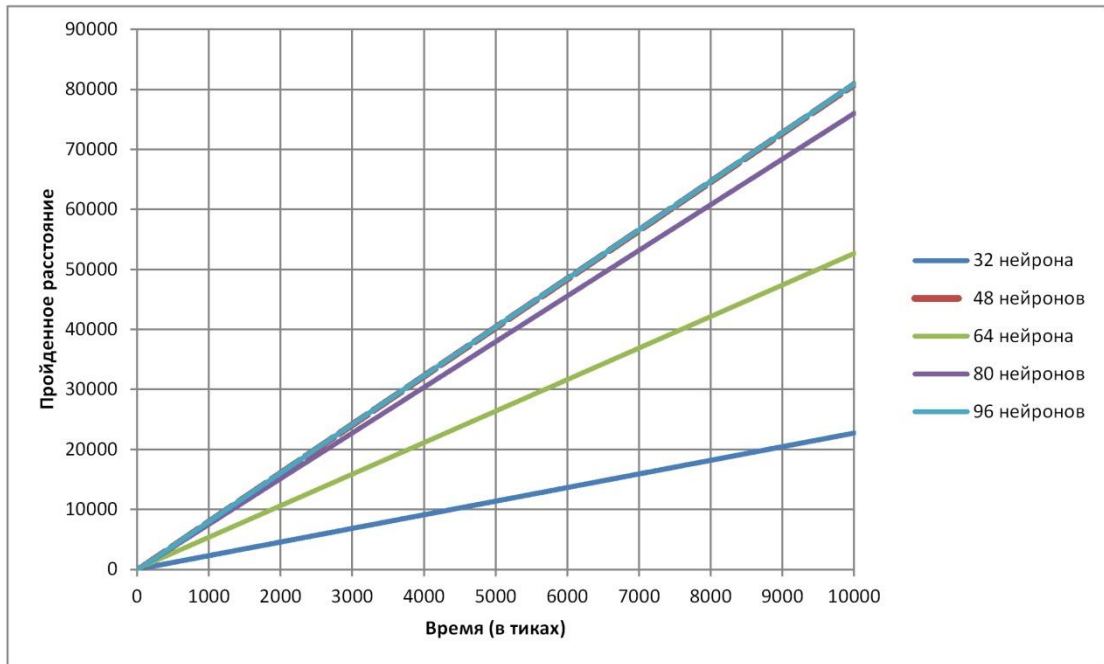


Рисунок 1 – Сравнение эффективности работы нейронной сети при различных количествах нейронов скрытого слоя

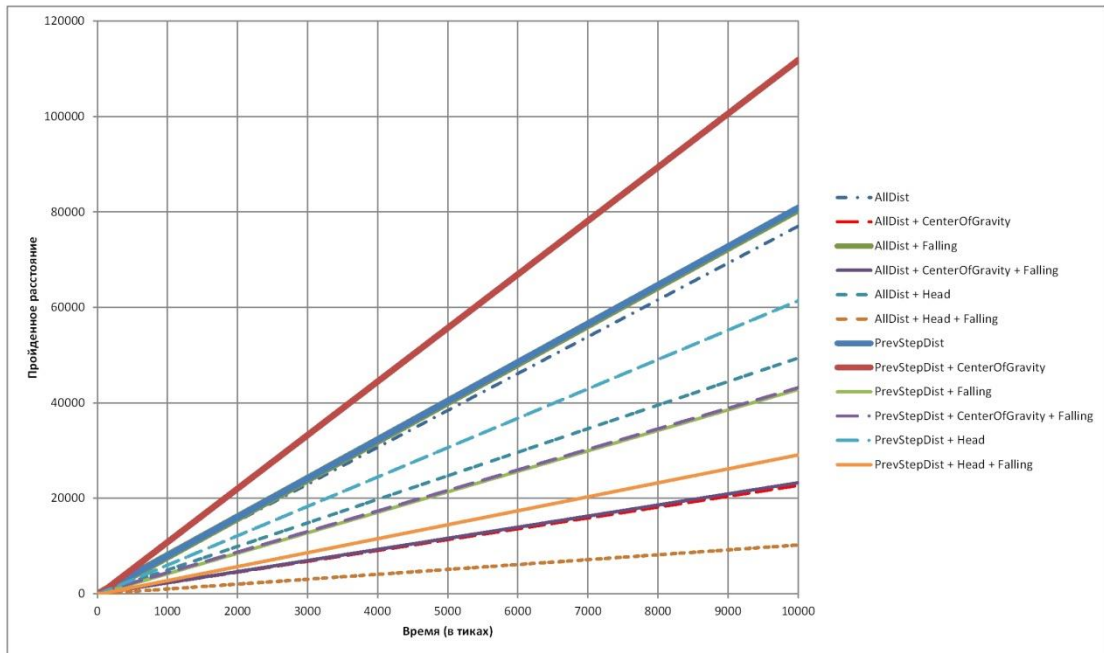


Рисунок 2 – Сравнение эффективности работы нейронной сети при различных способах вычисления вознаграждения

				БР-02069964-090301-16-18			
Имя	Фамилия	Имя	Фамилия	Искусственный интеллект на базе нейронной сети, реализующий парадигму модели сложного объекта в пространстве			
Степанов	Александр	Степанов	Александр	Имя 4	Имя 5	Имя 6	Имя 7
Григорьев	Павел	Григорьев	Павел	Анализ эффективности			
Иванов	Иван	Иванов	Иван	№ 9 от 10.06.2016 РСК АСОУ-ИИТ Группа ИИ4.1			
Сидоров	Игорь	Сидоров	Игорь				

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

Изм	Лист	№ докум.	Подпись	Дата

Инв.№ подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

БР - 09 - 090301 - 090301-16-18

## Анализ эффективности перемещения различных моделей

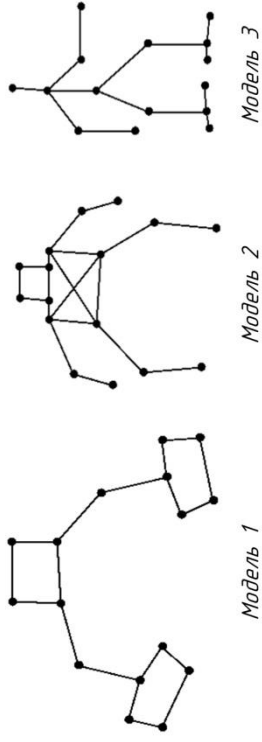


Рисунок 1 – Модели объектов

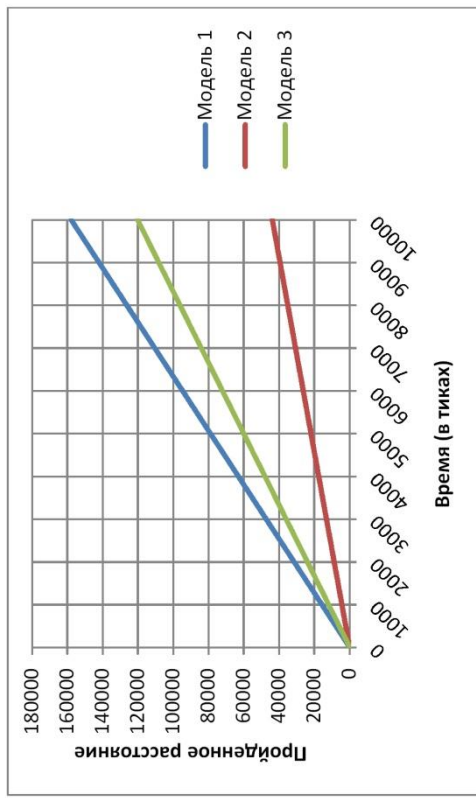


Рисунок 2 – Сравнение эффективности работы нейронной сети с вознаграждением, равным расстоянию, пройденному за последний шаг, для различных моделей

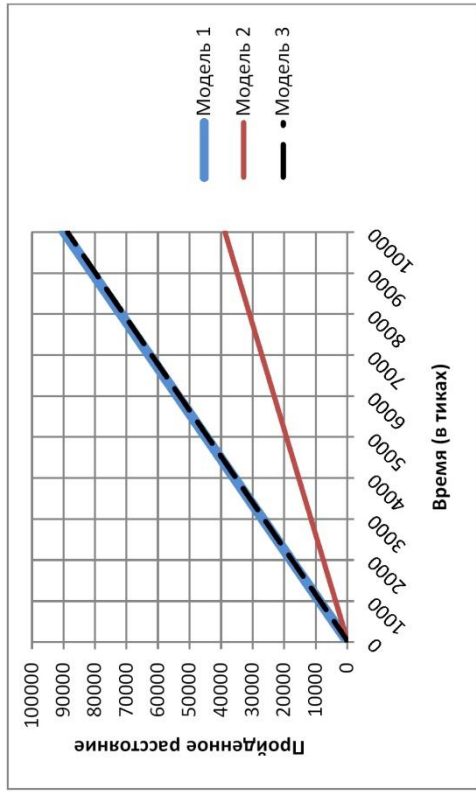


Рисунок 3 – Сравнение эффективности работы нейронной сети с вознаграждением, равным сумме расстояния, пройденного за последний шаг, и штрафа за цент тяжести, для различных моделей

БР-02069964-090301-16-18		Исследовательская работа на тему: «Исследование эффективности работы нейронной сети, реализующей перемещение объектов сложного объекта в пространстве»		Формат	PDF
Исполнитель	Иванов И.И.	Заказчик	ФГУП «В.О.Дальний»	Масштаб	1:1
Место выполнения работ	г. Москва	Сроки выполнения работ	с 01.01.2016 по 31.12.2016	Страна	Россия
Содержание работ	Анализ эффективности перемещения различных моделей				
Сроки выполнения работ	с 01.01.2016 по 31.12.2016				
Сроки сдачи работ	с 01.01.2016 по 31.12.2016				
Сроки оплаты работ	с 01.01.2016 по 31.12.2016				
Сроки подписания актов	с 01.01.2016 по 31.12.2016				
Сроки подписания актов	с 01.01.2016 по 31.12.2016				
Сроки подписания актов	с 01.01.2016 по 31.12.2016				