ProgramLab
INNOVATIVE DIGITAL SYSTEMS

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

ВИРТУАЛЬНЫЙ УЧЕБНЫЙ КОМПЛЕКС «ПРИМЕНЕНИЕ ПАКЕТА TENSORFLOW В ИГРОВЫХ ЗАДАЧАХ»

PLLLC.RU

ОГЛАВЛЕНИЕ

Инструкция по установке и запуску проекта 3

Запуск и управление в программе 5

Устранение проблем и ошибок 7

Введение в нейронные сети 9

Генетический алгоритм 26

Библиотека TensorFlow 31

Библиотека Keras 33

Установка и настройка сервера 36

Работа в программе 40

Спецификация API Парковка 47

Спецификация API Прохождение трассы 52

Спецификация API Попадание в обруч 57

Описание лабораторной работы 62

Инструкция по установке и запуску проекта

1. Распакуйте, соберите и подключите к сети компьютер.

2. Установите **«PLCore»**.

Модуль запуска программных комплексов **«PLCore»** предназначен для запуска, обновления и активации программных комплексов, поставляемых компанией «Програмлаб».

В случае поставки программного комплекса вместе с персональным компьютером модуль запуска **«PLCore»** устанавливается на компьютер перед отправкой заказчику.

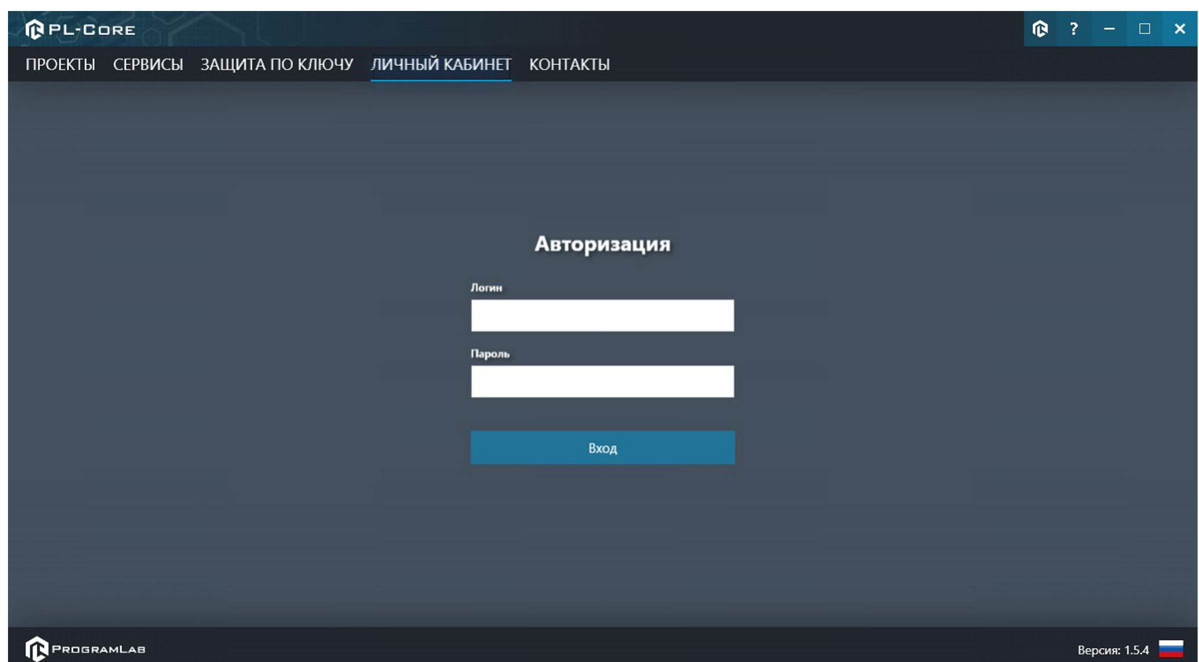
В случае поставки программного комплекса без ПК вам необходимо установить программное обеспечение с USB-носителя.

Перед установкой программного обеспечения установите модуль запуска учебных комплексов **«PLCore»**. Для этого запустите файл с названием вида PLCoreSetup_vX.X.X на USB-носителе (Значения после буквы v в названии файла обозначают текущую версию ПО) и следуйте инструкциям.

3.Войдите в личный кабинет **«PLCore»**.

ТУТ ПОНАДОБИТСЯ ЛОГИН И ПАРОЛЬ ИЗ КОНВЕРТА.

Во вкладке «Личный кабинет» располагается окно авторизации по уникальному логину и паролю. После прохождения авторизации в личном кабинете представляется информация о доступных программных модулях (описание, состояние лицензии, информация о версиях), с возможностями их удаленной загрузки, обновления и активации по сети интернет.



Вход в личный кабинет «PLCore»

4. Активируйте проект следуя руководству пользователя **«PLCore»**.

5. Установите **«PLStudy»** – Администрирование сервера данных учебных модулей.

Если ваш стенд предполагает автоматическую отправку результатов, а также систему ролей пользователей для работы группы, то вам понадобится программный модуль «Администрирование сервера данных учебных модулей». Модуль позволяет управлять базой данных студентов и их результатов для всех комплексов нашей компании сразу.

Установите сервер данных учебных модулей, если он ещё не установлен, на компьютер, который будет являться сервером. Для этого воспользуйтесь руководством пользователя **«PLStudy»**.

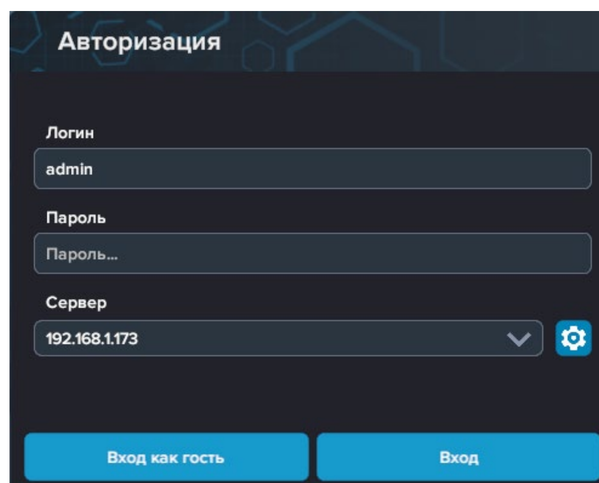
По умолчанию в системе создается пользователь с именем Администратор и ролью Администратор. Этот пользователь не может быть удален, но его параметры могут быть изменены.

По умолчанию логин пользователя: admin; Пароль: admin.

6. Запустите проект.

Перед входом программа запросит логин, пароль. Здесь необходимо ввести параметры администратора или созданного на сервере («PLStudy») пользователя. При авторизации в поле «Сервер» должен быть указан IP-адрес компьютера, на котором установлен сервер данных учебных модулей.

Чтобы изменить IP-адрес см. пункт «Запуск и управление в модуле» в руководстве пользователя **«PLStudy»**.

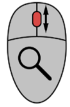


Окно авторизации

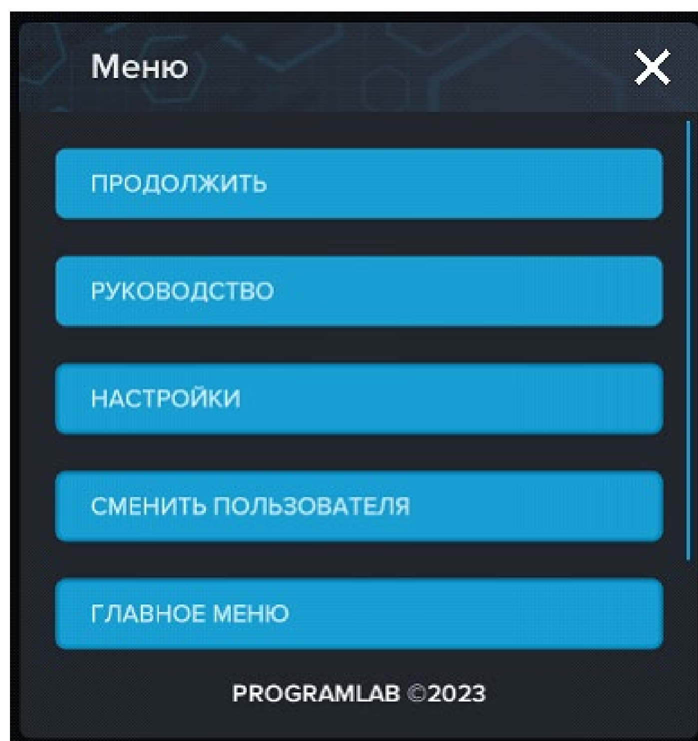
Запуск и управление в программе

— Левая кнопка мыши – действие, выбор объекта;

— Правая кнопка мыши – вращение камеры;

 — Вращение колеса мыши – приближение\отдаление камеры;

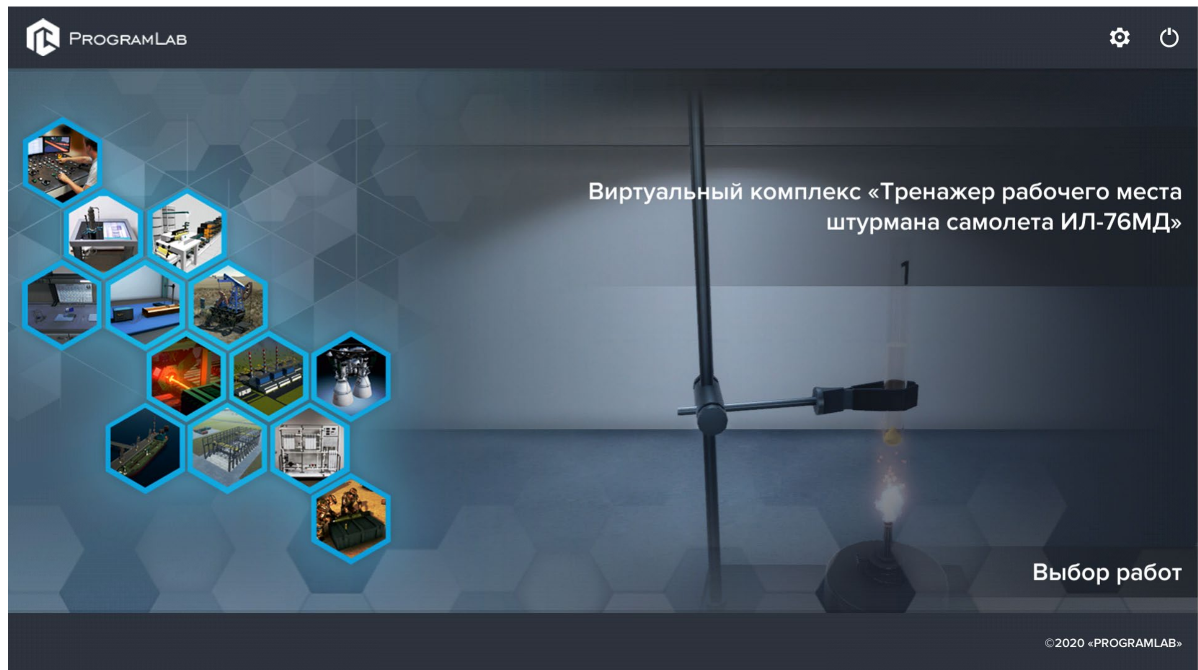
— Вызов меню программы.



Меню программы

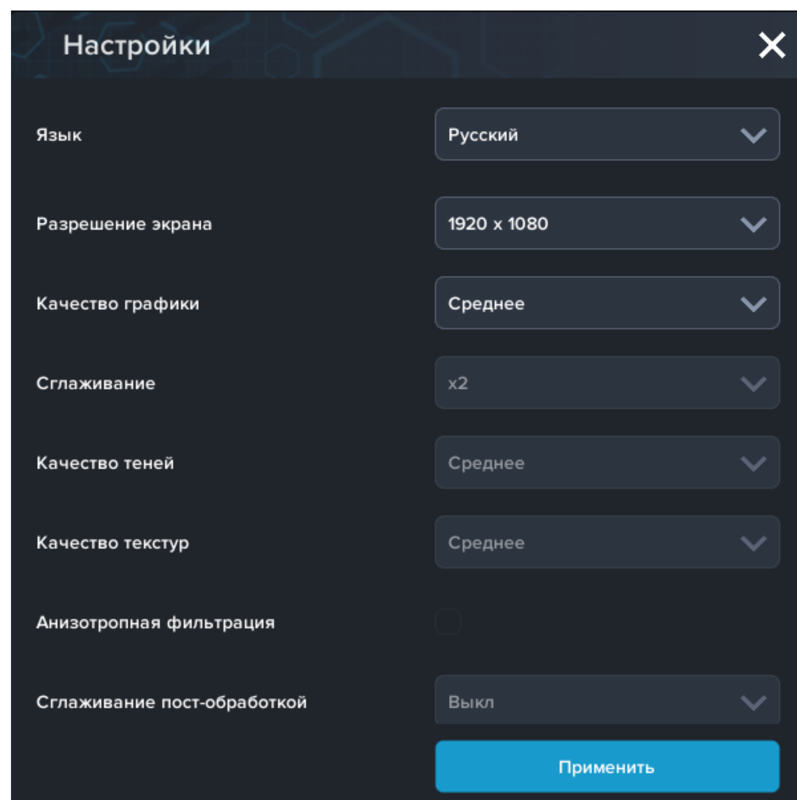
«Продолжить» – вернуться в программу;
«Руководство» – вызвать руководство пользователя;
«Настройки» – настройки параметров графики;
«Сменить пользователя» – пройти авторизацию повторно;
«Главное меню» – выход в главное меню;
«Выход» – выход из программы.

Для запуска программы нажмите кнопку **«Загрузить»**, либо нажмите кнопку **«Выбор работ»** и выберите из открывшегося списка режим работы.



Окно запуска программного модуля

Для изменения настроек графики нажмите кнопку .



Окно настроек графики

Нажмите **«Применить»** чтобы закрыть окно.

Для выхода из программы нажмите .

Устранение проблем и ошибок

При возникновении ошибок в работе с программным обеспечением свяжитесь со специалистом поддержки «Програмлаб». Для этого опишите вашу проблему в письме на почту support@pl-llc.ru либо позвоните по телефону 8 800 550 89 72.

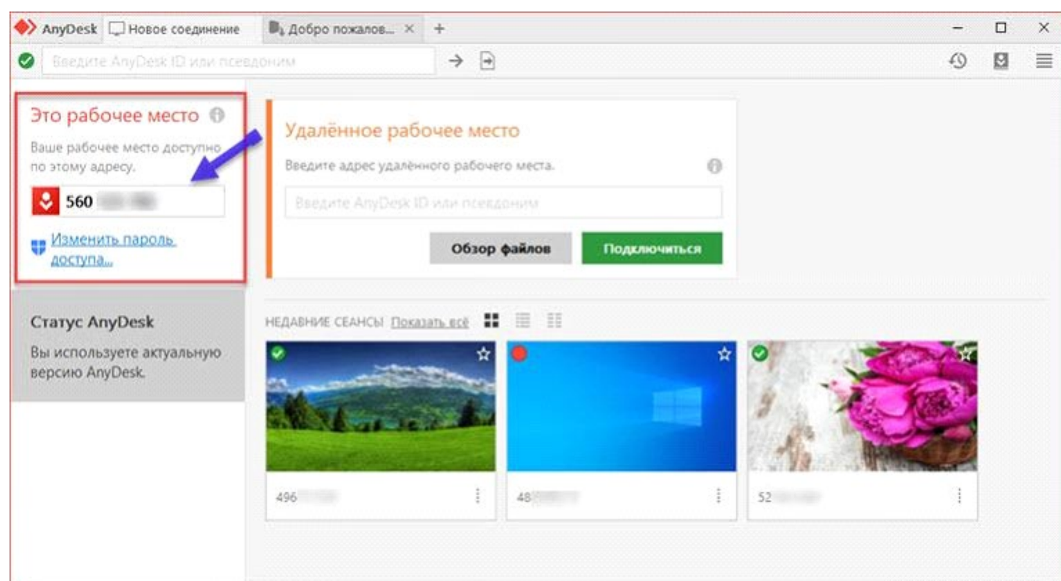
Для того чтобы специалист смог подключиться к вашему ПК и устранить проблемы вам необходимо запустить ПО для дистанционного управления ПК Anydesk и сообщить данные для доступа.

Приложение Anydesk можно найти на USB-носителе с дистрибутивом. Вставьте USB-носитель в ПК и запустите файл с названием Anydesk.exe

После того как приложение скачано нужно запустить его. Необходимый файл называется **AnyDesk.exe** и лежит папке «**Загрузки**».

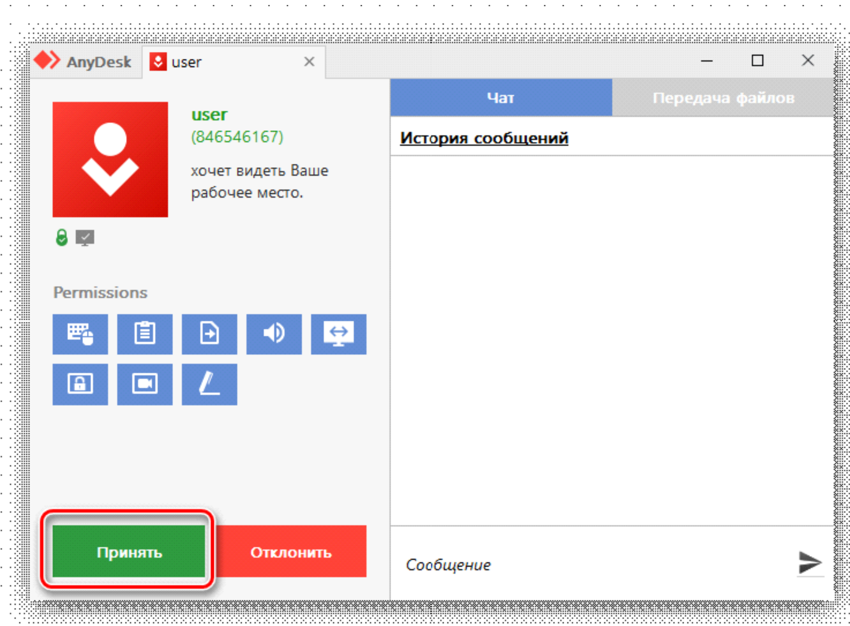
При первом запуске может возникнуть окно с требованием предоставить разрешение. Необходимо нажать на кнопку **Разрешить доступ.**

Для того, чтобы к вашему компьютеру мог подключиться другой пользователь, необходимо ему передать специальный адрес, который называется «Это рабочее место». Сообщите этот адрес специалисту.



Окно Anydesk с адресом

После того как специалист введет переданный вами адрес вам нужно будет подтвердить разрешение на доступ к вашему ПК. Откроется табличка с вопросом «Принять» или «Отклонить» удаленное соединение. Нажмите «Принять».



Окно Anydesk Принять/Отклонить

На этом настройка удаленного соединения завершена: специалист получил доступ к вашему ПК. В случае необходимости продолжайте следовать инструкциям специалиста.

Введение в нейронные сети

Нейронная сеть (neural network) – это компьютерный алгоритм, способный обрабатывать большие объемы данных, имитируя деятельность человеческого мозга. Как и человек, нейросеть изучает новые предметы, делает выводы и в дальнейшем использует полученную информацию. Нейросети представляют собой математические модели, созданные на основе биологических нейронных сетей, существующих в глубинах человеческого мозга.

Нервную систему человека образуют нейроны – клетки, которые получают информацию и транслируют ее в виде импульсов. Основная часть нейрона – аксон, а длинный отросток на его конце носит название дендрит, он выполняет роль своеобразного провода при передаче информации от одного нейрона к другому. Таким образом мозг, транслируя информацию, управляет всеми действиями человека.

На основе соответствующего принципа работают и компьютерные нейронные сети, ставшие цифровой моделью человеческого мозга. Главная же их особенность – **способность к обучению**. Стандартные компьютерные программы предполагают, что алгоритм для них пишет человек, то есть задает определенный набор действий, которые должны выполнить компьютеры. При использовании нейросети не нужно говорить ей, как решить задачу. Достаточно задать вводные данные, а способам решения задач нейронная сеть на основе искусственного интеллекта обучается сама, выявляя закономерности и обнаруживая на их основе способы решения задач.

История появления нейросети

Попытки математически описать сеть нейронов предпринимались еще в 1940-е годы. Идею создания нейронных сетей впервые предложили исследователи из Чикагского университета Уоррен Маккалоу и Уолтер Питтс. В 1950-е годы эта математическая модель была воссоздана психологом Корнеллского университета Фрэнком Розенблаттом с помощью компьютерного кода. Розенблатт был автор перцептрона – прототипа современных нейросетей. Даже такая элементарная структура в те годы могла обучаться и самостоятельно решать простые задачи.

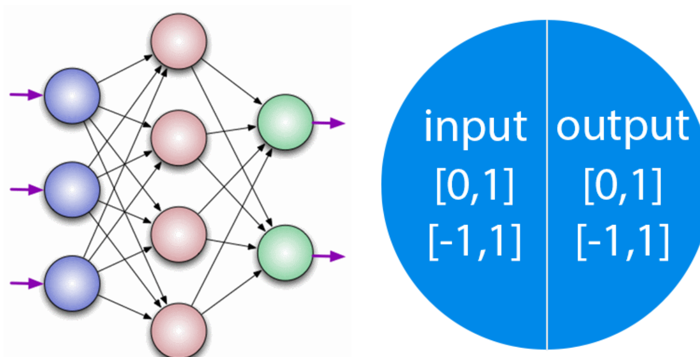
Возрождение интереса к нейронным сетям и революция в глубоком обучении произошли лишь в последние годы благодаря индустрии компьютерных игр. Современные игры требуют сложных вычислений для обработки большого числа операций. В итоге производители начали выпускать **графические процессоры (GPU)**, которые объединяют тысячи относительно простых вычислительных ядер на одном чипе. Исследователи вскоре поняли, что архитектура графического процессора очень похожа на архитектуру нейросети.

Современные GPU позволили развивать «глубокое обучение» — повышать глубину слоев нейросети. Именно благодаря ему появились самообучаемые нейросети, которые не требуют специальной настройки, а самостоятельно обрабатывают входящую информацию.

Структура нейросети

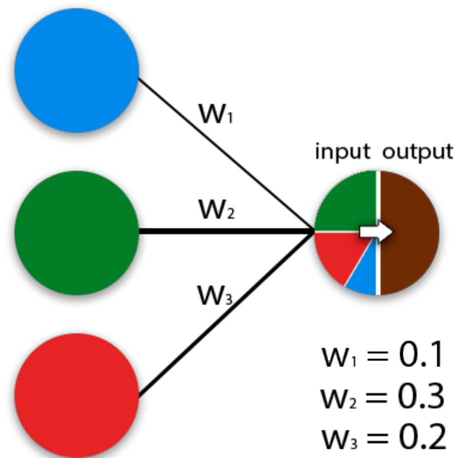
Главное отличие нейросетевых моделей от классических заключается в их структуре. Основные элементы, из которых он состоит – искусственные нейроны и связи между ними.

Нейрон — это вычислительная единица, которая получает информацию, производит над ней простые вычисления и передает ее дальше. Они делятся на три основных типа: входной (синий), скрытый (красный) и выходной (зеленый). В том случае, когда нейросеть состоит из большого количества нейронов, вводят термин слоя. У каждого из нейронов есть 2 основных параметра: входные данные (input data) и выходные данные (output data). В случае входного нейрона: input=output. В остальных, в поле input попадает суммарная информация всех нейронов с предыдущего слоя, после чего, она нормализуется, с помощью функции активации (представим ее f(x)) и попадает в поле output.



Важно помнить, что нейроны оперируют числами в диапазоне [0,1] или [-1,1]. Числа, которые выходят из данного диапазона необходимо обрабатывать, разделив 1 на это число. Этот процесс называется нормализацией, и он очень часто используется в нейронных сетях.

Синапс – это связь между двумя нейронами. У синапсов есть 1 параметр — вес. Благодаря ему, входная информация изменяется, когда передается от одного нейрона к другому. Допустим, есть 3 нейрона, которые передают информацию следующему. Тогда у нас есть 3 веса, соответствующие каждому из этих нейронов. У того нейрона, у которого вес будет больше, та информация и будет доминирующей в следующем нейроне (пример — смешение цветов).



На самом деле, совокупность весов нейронной сети или матрица весов — это своеобразный мозг всей системы. Именно благодаря этим весам, входная информация обрабатывается и превращается в результат.

Важно помнить, что во время инициализации нейронной сети, веса расставляются в случайном порядке.

Нейронов в нейросети много, поэтому они объединяются в **слои**:

- Входной, куда поступают данные. Они могут иметь любой формат – файлы, тексты, музыка, картинки, видео и другие.
- Скрытые, в которых производятся вычисления и обработка. Обычно скрытых слоев не больше трех.
- Выходной – отсюда выходят результаты.

Глобально нет разницы между искусственным интеллектом (ИИ) и нейросетями.

Нейросеть — это компьютерная система, которая имитирует работу нейронов в мозге человека. Она состоит из множества «нейронов», соединённых между собой и передающих информацию по цепочке. Нейросети используются во многих сферах для решения различных задач, в том числе для распознавания образов, обработки речи и прочего.

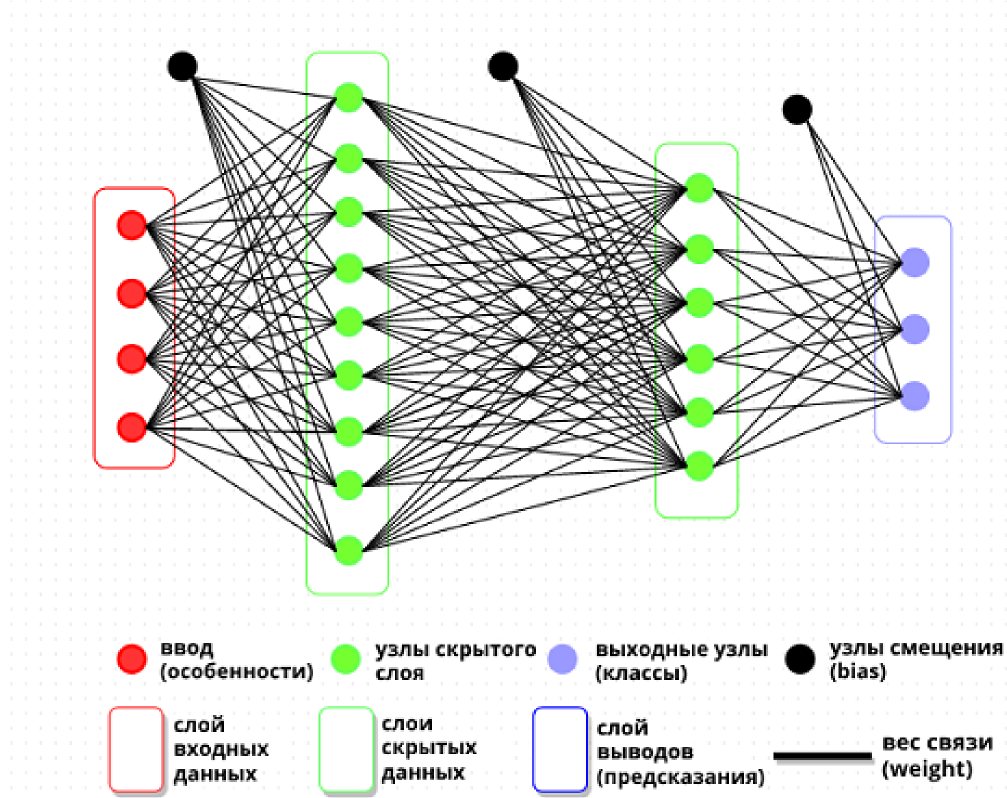
Искусственный интеллект — понятие более широкое. Оно включает в себя не только нейронные сети, но и другие методы обработки информации, в том числе экспертные и логические программы. Нейронные сети — один из видов искусственного интеллекта. Их отличительная особенность — обучение и адаптация в основе алгоритмов.

Искусственная нейронная сеть (ИНС) представляет собой систему соединённых и взаимодействующих между собой простых процессоров (искусственных нейронов). Такие процессоры обычно довольно просты (особенно в сравнении с процессорами, используемыми в персональных компьютерах). Каждый процессор подобной сети имеет дело только с сигналами, которые он периодически получает, и сигналами, которые он периодически посылает другим процессорам. И, тем не менее, будучи соединёнными в достаточно большую сеть с управляемым взаимодействием, такие по отдельности простые процессоры вместе способны выполнять довольно сложные задачи.

- С точки зрения машинного обучения, нейронная сеть представляет собой частный случай методов распознавания образов, дискриминантного анализа;
- С точки зрения математики, обучение нейронных сетей — это многопараметрическая задача нелинейной оптимизации;
- С точки зрения кибернетики, нейронная сеть используется в задачах адаптивного управления и как алгоритмы для робототехники;
- С точки зрения развития вычислительной техники и программирования, нейронная сеть — способ решения проблемы эффективного параллелизма;
- С точки зрения искусственного интеллекта, ИНС является основой философского течения коннекционизма и основным направлением в структурном подходе по изучению возможности построения (моделирования) естественного интеллекта с помощью компьютерных алгоритмов.

Принцип работы

Чем большее число слоев в нейронной сети, тем сложнее задачи, с которыми она может справляться.



- Входной слой нейронов воспринимает информацию. Это могут быть фото, видео, аудио, текстовые файлы — данные в любом формате и объёме.
- На скрытом слое происходит обработка и перевод данных в математические числовые коды. Количество скрытых слоёв не ограничено и зависит от объёма данных и поставленных задач, чаще всего их три.
- Ответ сети формируется в выходном слое. Формат ответа также может быть любым.

На входной слой поступает запрос и данные, которые необходимо обработать. На скрытом слое происходит непосредственно работа: сортировка, отбор по конкретному признаку и прочее. На выходном слое нейросеть выдаёт итог проделанной работы.

Например, для обучения и генерации конечного результата в виде изображения, сеть перерабатывает огромное количество текстовых данных и изображений. Это позволяет ей создавать красивые картинки на основе заданных параметров. Вот в чём состоит принцип действия:

1. Ввод запроса: пользователь вводит текст, который нейросети нужно преобразовать в изображение. Текст может быть любым: описание объекта, сцена, даже стихотворение.
2. Токенизация: нейросеть разбивает введённый текст на отдельные слова или фразы — токены. Каждый представляет собой часть информации, которую нейросеть может обрабатывать.
3. Представление токенов в числовом виде: сеть преобразует информацию в числовой формат. Этот процесс называется векторизацией. Она позволяет нейронной сети работать с токенами в скрытом слое.
4. Обработка токенов нейросетью: в зависимости от сложности задачи работа происходит на разных слоях. В результате многослойной обработки нейросеть формирует промежуточное представление токенов.
5. Генерация изображения: промежуточные токены преобразуются в изображение — подвергаются декодированию.
6. Вывод изображения: пользователь получает изображение, которое соответствует введённому тексту.

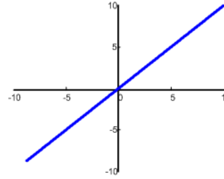
Чем точнее и подробнее запрос, тем быстрее и качественнее получится результат.

Функции нейросети

Функция активации — это способ нормализации входных данных. То есть, если на входе у вас будет большое число, пропустив его через функцию активации, вы получите выход в нужном вам диапазоне. Функций активации достаточно много поэтому мы рассмотрим самые основные: Линейная, Сигмоид (Логистическая) и Гиперболический тангенс. Главные их отличия — это диапазон значений.

Линейная функция

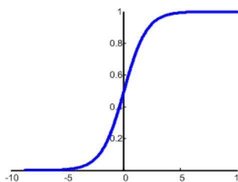
$$f(x) = x$$



Эта функция почти никогда не используется, за исключением случаев, когда нужно протестировать нейронную сеть или передать значение без преобразований.

Сигмоид

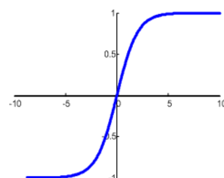
$$f(x) = \frac{1}{1+e^{-x}}$$



Это самая распространенная функция активации, ее диапазон значений [0,1]. Именно на ней показано большинство примеров в сети, также ее иногда называют логистической функцией.

Гиперболический тангенс

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



Имеет смысл использовать гиперболический тангенс, только тогда, когда ваши значения могут быть и отрицательными, и положительными, так как диапазон функции [-1,1]. Использовать эту функцию только с положительными значениями нецелесообразно так как это значительно ухудшит результаты вашей нейросети.

Тренировочный сет

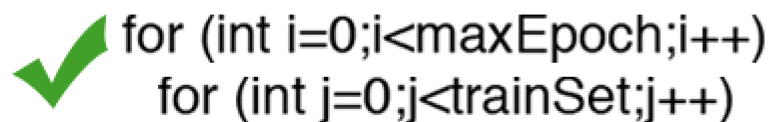
Тренировочный сет — это последовательность данных, которыми оперирует нейронная сеть.

Итерация

Это своеобразный счетчик, который увеличивается каждый раз, когда нейронная сеть проходит один тренировочный сет. Другими словами, это общее количество тренировочных сетов, пройденных нейронной сетью.

Эпоха

При инициализации нейронной сети эта величина устанавливается в 0 и имеет потолок, задаваемый вручную. Чем больше эпоха, тем лучше натренирована сеть и соответственно, ее результат. Эпоха увеличивается каждый раз, когда мы проходим весь набор тренировочных сетов.



Важно не путать итерацию с эпохой и понимать последовательность их инкремента. Сначала n раз увеличивается итерация, а потом уже эпоха и никак не наоборот. Другими словами, нельзя сначала тренировать нейросеть только на одном сете, потом на другом и т.д. Нужно тренировать каждый сет один раз за эпоху. Так, вы сможете избежать ошибок в вычислениях.

Ошибка

Ошибка — это процентная величина, отражающая расхождение между ожидаемым и полученным ответами. Ошибка формируется каждую эпоху и должна идти на спад. Если этого не происходит, значит, вы что-то делаете не так. Ошибку можно вычислить разными путями, но мы рассмотрим лишь три основных способа: Mean Squared Error (далее MSE), Root MSE и Arctan. Каждый метод считает ошибки по-разному. У Arctan, ошибка, почти всегда, будет больше, так как он работает по принципу: чем больше разница, тем больше ошибка. У Root MSE будет наименьшая ошибка, поэтому, чаще всего, используют MSE, которая сохраняет баланс в вычислении ошибки.

MSE:

$$\frac{(i_1-a_1)^2+(i_2-a_2)^2+\ldots+(i_n-a_n)^2}{n}$$

Root MSE:

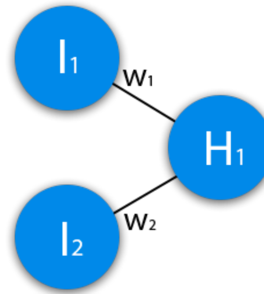
$$\sqrt{\frac{(i_1-a_1)^2+(i_2-a_2)^2+\ldots+(i_n-a_n)^2}{n}}$$

Arctan:

$$\frac{\arctan^2(i_1-a_1)+\ldots+\arctan^2(i_n-a_n)}{n}$$

Принцип подсчета ошибки во всех случаях одинаков. За каждый сет, мы считаем ошибку, отняв от идеального ответа, полученный. Далее, либо возводим в квадрат, либо вычисляем квадратный тангенс из этой разности, после чего полученное число делим на количество сетов.

Задача



$$1)\ H_{1input} = (I_1 * w_1) + (I_2 * w_2)$$

$$2)\ H_{1output} = f_{activation}(H_{1input})$$

В данном примере изображена часть нейронной сети, где буквами I обозначены входные нейроны, буквой H — скрытый нейрон, а буквой W — веса. Из формулы видно, что входная информация — это сумма всех входных данных, умноженных на соответствующие им веса.

Зададим на вход 1 и 0. Пусть W_1=0.4 и W_2=0.7

Входные данные нейрона H_1 будут следующими: 1*0.4+0*0.7=0.4.

Теперь, когда у нас есть входные данные, мы можем получить выходные данные, подставив входное значение в функцию активации.

Теперь, когда у нас есть выходные данные, мы передаем их дальше. И так, мы повторяем для всех слоев, пока не дойдем до выходного нейрона. Запустив такую сеть в первый раз, мы увидим, что ответ далек от правильно, потому что сеть не натренирована. Чтобы улучшить результаты мы будем ее тренировать.

Эпоха увеличивается каждый раз, когда мы проходим весь набор тренировочных сетов, в нашем случае, 4 сетов или 4 итераций.

Теперь, чтобы проверить себя, подсчитайте результат, данной нейронной сети, используя сигмоид, и ее ошибку, используя MSE.

Данные: I_1=1, I_2=0, W_1=0.45, W_2=0.78, W_3=-0.12, W_4=0.13, W_5=1.5, W_6=-2.3.

Решение:

H_1input = 1*0.45+0*-0.12=0.45

H_1output = sigmoid(0.45)=0.61

H_2input = 1*0.78+0*0.13=0.78

H_2output = sigmoid(0.78)=0.69

O_1input = 0.61*1.5+0.69*-2.3=-0.672

O_1output = sigmoid(-0.672)=0.33

O_1ideal = 1 (0xor1=1)

Error = ((1-0.33)^2)/1=0.45

Результат — 0.33, ошибка — 45%.

Методы обучения

Один из главных признаков нейросетей – способность к обучению. Перед началом обучения все веса нейронной сети определяются случайными значениями. Обучающие данные передаются на входной слой, проходят через следующие слои и достигают выходного. В процессе обучения данные постоянно подвергаются корректировке, и циклы повторяются до тех пор, пока данные обучения не станут показывать одинаковые результаты.

По сути, любая модель машинного обучения использует метод градиентного спуска. Он применяется и для обучения нейросетей и называется методом обратного распространения ошибки.

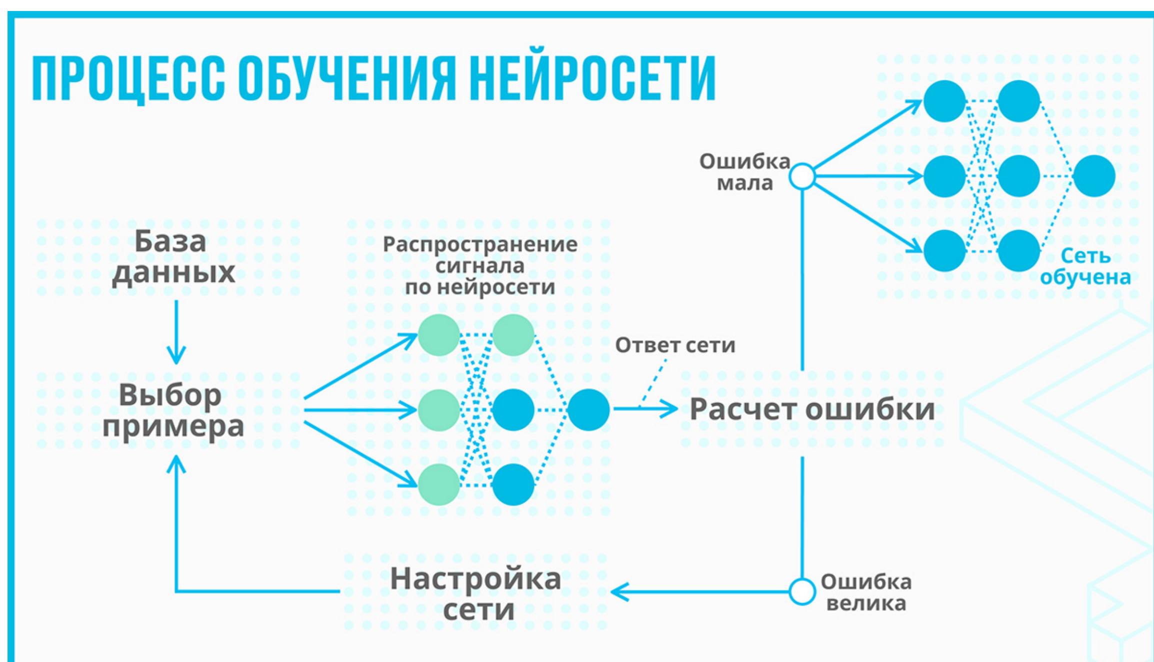
Существуют следующие методы обучения:

- **С учителем.** Пользователь дает сигнал на вход, получает на выходе ответ нейросети, затем сравнивает его с уже известным правильным. После этого с помощью специальных алгоритмов меняются веса связей и снова задается входной сигнал. Процесс продолжается до тех пор, пока нейросеть не начнет отвечать точно. Такое обучение называют также контролируемым.

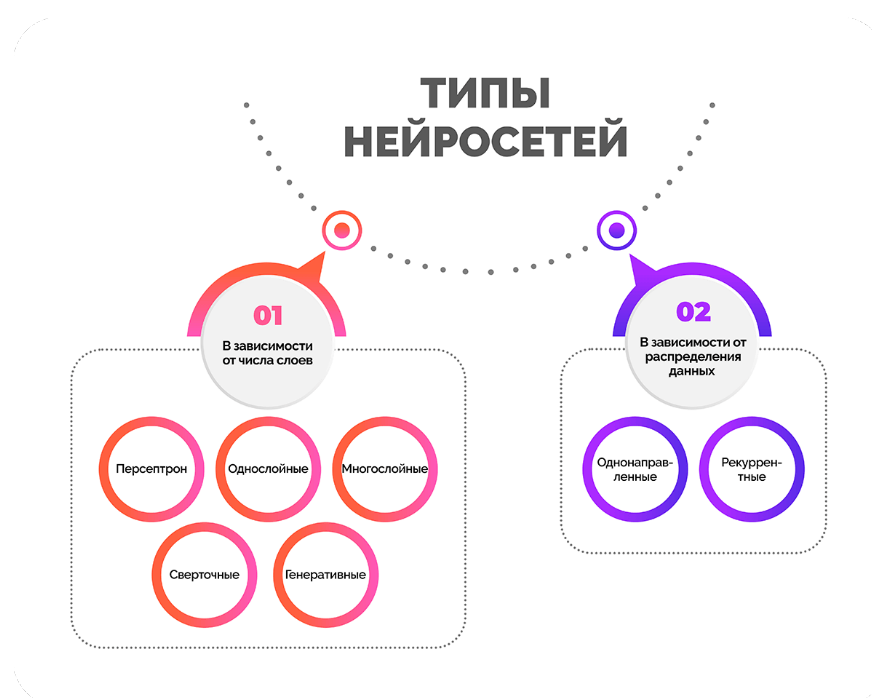
- **Без учителя.** Метод применяют, если нет правильных ответов на входные сигналы. Сеть в этом случае, используя собственную память, делит объекты на классы, то есть начинает кластеризацию. Эталонные ответы при этом не показаны. Данный тип обучения называют глубоким: система все время обучается сама.

- **С подкреплением.** Такие нейросети обучаются самостоятельно, но при этом взаимодействуют с окружающей средой, которая специально моделируется и становится обучающей. Чаще всего такой подход применяют в робототехнике и разработке игр.

В зависимости от типа входной информации выделяют аналоговые, двоичные и образные нейросети.



Типы нейросетей



В зависимости от числа слоев, в которых расположены нейроны, нейросети могут быть:

- **Персептрон** – самая старая форма. Один нейрон принимает информацию, применяет активацию, в результате становится доступным вывод в двоичной системе. Перцептрон можно использовать только для классификации данных на две группы. Из-за ограниченных возможностей такие нейронные сети в наше время практически не используются.
- **Однослойные**. Сигнал поступает во входной слой и сразу же отправляется к выходному, где происходят вычисления. Связь между нейронами входного и выходного слоев обеспечивают синапсы.
- **Многослойные**. Помимо входного и выходного слоев, в таких нейронных сетях есть еще несколько скрытых промежуточных. Обработка информации и вычисления производятся на нескольких этапах, поэтому решения, предлагаемые такими сетями, более точные.
- **Сверточные**. В структуру таких нейросетей входят два дополнительных слоя - сверточные и объединяющие. Сверточные нейронные сети используются для обработки изображений, картинок и фото.
- **Генеративные**. В эту группу входят нейросети, способные что-то создавать. Это, к примеру, генераторы картинок или текстов.

Еще одна классификация делит нейросети на однонаправленные и рекуррентные в зависимости от распределения данных по синапсам:

- **Однонаправленные** (прямого распространения). Сигнал движется от входного слоя к выходному, обратного движения нет. Нейросети такого типа используют для распознавания речи, кластеризации, составления прогнозов.
- **Рекуррентные** (с обратными связями). Рекуррентные нейронные сети предполагают, что любое количество сигналов может перемещаться в разных направлениях, в том числе от выхода к входу.

По типам нейронов сети могут быть однородными или гибридными. Первые состоят из нейронов одного типа, вторые сочетают несколько классов нейронов. По характеру настройки синапсов нейронные сети бывают с фиксированными либо с динамическими связями.

Применения нейросетей

Разные варианты нейросетей создаются для решения нескольких типов различных задач:

- Классификация – отнесение объектов к нужному классу.
- Регрессия – предсказывание результата в виде чисел (например, стоимости дома в зависимости от его площади и района, в котором он расположен).
- Распознавание – выделение объекта среди огромного множества других похожих (пример - сеть может выделить конкретное лицо в толпе).
- Кластеризация – разделение объектов на несколько групп по какому-либо признаку, неизвестному ранее. Это, например, разбивка документов на разные классы.
- Генерация – рождение чего-то нового в рамках заданной тематики.
- Прогнозирование – на основе полученных данных искусственный интеллект формулирует прогнозы по заданной теме на определенное время.

В зависимости от задачи, которую могут решать искусственные нейронные сети (она у каждого своя), они используются в разных областях. Перечислим сферы, где они наиболее востребованы:

1. **Медицина.** Искусственный интеллект помогает обрабатывать снимки и другие данные исследований и тем самым позволяет врачам устанавливать точный диагноз, при этом тратить меньше времени.

2. **Образование.** Преподаватели с помощью искусственных сетей имеют возможность быстрее проверять домашние задания, за короткое время составлять сложные презентации и планы уроков.

3. **Искусство.** Нейросети создают изображения, произведения литературы и музыку.

4. **Строительство и архитектура.** Искусственный интеллект полезен застройщикам, чтобы выбрать материалы, прогнозировать время выполнения работ.

5. **Безопасность.** Нейросети имеют возможность распознавать обычные лица и путем слежки в общественных местах вычислять преступников, которые находятся в розыске.

6. **Банковская сфера.** Нейронная сеть анализирует кредитную историю клиентов, создает прогнозы биржевых индексов.

7. **Производство**. Искусственный интеллект участвует в отслеживании производственных процессов, дают возможность контролировать продукции на предприятиях.

Применение:

Генерация и обработка изображений. Нейронные сети из этой категории рисуют на основе текста и пользовательских изображений с любом указанном стиле, в том числе используя вектор. Сервисы могут изменять фон картинки, дорисовывать изображения по описанию, генерировать картинку на основе фотографий, создавать визуальный контент для брендов и логотипы, а также реалистичные изображения в дополнение к текстовому описанию карточек товаров в интернет-магазинах и на маркетплейсов, фотографии для социальных сетей.

Генерация игровых миров и персонажей. Нейронные сети, создающие персонажей для игр, уровни, анимацию, видео, изображения для интерфейса. Упрощают разработку сюжетных линий и хода игры.

Работа с аудио. Нейронные сети могут просто преобразовать аудио в текст и обратно, расшифровывать в форме текста записи конференций, интервью и лекций. Используются для озвучивания роликов и прочего видеоконтента, для улучшения качества аудиозаписей и избавления их от шумов и посторонних звуков, для генерации музыки. Сервисы поддерживают несколько языков, включая русский. Многие подобные сети разработаны на основе языковой модели ChatGPT.

Музыка. Нейронные сети с ИИ могут создать музыку в разных стилях с нуля или обрабатывать и аранжировать мелодии.

Видео. Нейросети создают видео ролики с персонажами с возможностями настройки голоса и стиля речи. Источниками для видео роликов могут быть собственные сценарии или контент сайтов, соцсетей, приложений. Некоторые инструменты могут также создавать GIF-анимацию, озвучивать тексты, накладывать на видео фоновую музыку и даже делать фильмы.

Написание кода. Нейронные сети ускоряют разработку кода на разных языках программирования. Могут находить ошибки в уже написанных кодах, генерировать коды по текстовому запросу, создавать тесты.

Создание документов и презентаций. Умеют по запросу генерировать любой контент, структурировать информацию и разбивать ее по слайдам, добавлять диаграммы. Сеть генерирует изображения, обрабатывает фотографии и прочие визуальные элементы.

Преимущества и недостатки нейросетей

Преимущества нейросетей:

– незаменимы в сфере автоматизации процессов;

– спасают там, где может навредить человеческий фактор;

– экономят время на выполнении рутинных задач;

– постоянно обучаются.

Недостатки:

– напрямую зависят от вводимых данных, поэтому сильно подвержены влиянию;

– чтобы получить действительно хорошую рабочую сеть, нужно потратить много времени на её обучение;

– занимают много места на сервере и требуют больших вычислительных мощностей.

Учитывая то, с какой скоростью развивается искусственный интеллект сегодня, плюсы и минусы нейросетей достаточно относительны. Но их нельзя игнорировать.

Генетический алгоритм

Генетический алгоритм — эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путём случайного подбора, комбинирования и вариации искомых параметров с использованием механизмов, аналогичных естественному отбору в природе. Является разновидностью эволюционных вычислений, с помощью которых решаются оптимизационные задачи с использованием методов естественной эволюции, таких как наследование, мутации, отбор и кроссинговер. Отличительной особенностью генетического алгоритма является акцент на использование оператора «скрещивания», который производит операцию рекомбинации решений-кандидатов, роль которой аналогична роли скрещивания в живой природе.

Описание алгоритма

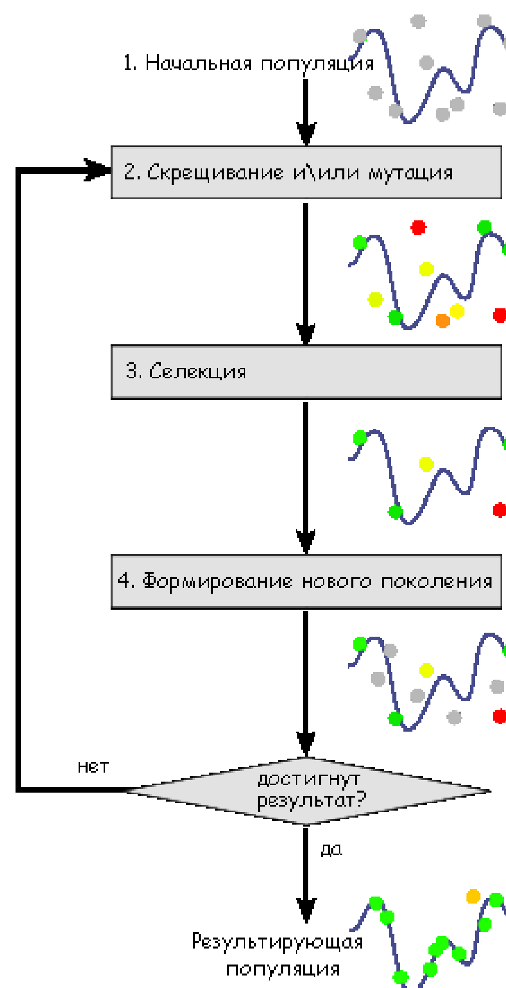


Схема работы генетического алгоритма

Задача формализуется таким образом, чтобы её решение могло быть закодировано в виде вектора («генотипа») генов, где каждый ген может быть битом, числом или неким другим объектом. В классических реализациях генетического алгоритма (ГА) предполагается, что генотип имеет

фиксированную длину. Однако существуют вариации ГА, свободные от этого ограничения.

Некоторым, обычно случайным, образом создаётся множество генотипов начальной популяции. Они оцениваются с использованием «функции приспособленности», в результате чего с каждым генотипом ассоциируется определённое значение («приспособленность»), которое определяет, насколько хорошо описываемый им фенотип, решает поставленную задачу.

При выборе «функции приспособленности» важно следить, чтобы её «рельеф» был «гладким».

Из полученного множества решений («поколения») с учётом значения «приспособленности» выбираются решения, к которым применяются «генетические операторы» (в большинстве случаев «скрещивание» и «мутация»), результатом чего является получение новых решений. Для них также вычисляется значение приспособленности, и затем производится отбор («селекция») лучших решений в следующее поколение.

Этот набор действий повторяется итеративно, так моделируется «эволюционный процесс», продолжающийся несколько жизненных циклов, пока не будет выполнен критерий остановки алгоритма. Таким критерием может быть:

- нахождение глобального, либо субоптимального решения;
- исчерпание числа поколений, отпущенных на эволюцию;
- исчерпание времени, отпущенного на эволюцию.

Генетические алгоритмы служат, главным образом, для поиска решений в многомерных пространствах поиска.

Таким образом, можно выделить следующие этапы генетического алгоритма:

1. Задать целевую функцию (приспособленности) для особей популяции;
2. Создать начальную популяцию.

   Начало цикла:

1. Размножение (скрещивание);
2. Мутирование;
3. Вычислить значение целевой функции для всех особей;
4. Формирование нового поколения (селекция);
5. Если выполняются условия остановки, то (конец цикла), иначе (начало цикла).

Создание начальной популяции

Перед первым шагом нужно случайным образом создать начальную популяцию; даже если она окажется совершенно неконкурентоспособной, вероятно, что генетический алгоритм всё равно достаточно быстро переведёт её в жизнеспособную популяцию. Таким образом, на первом шаге можно особенно не стараться сделать слишком уж приспособленных особей,

достаточно, чтобы они соответствовали формату особей популяции, и на них можно было подсчитать функцию приспособленности (Fitness). Итогом первого шага является популяция H, состоящая из N особей.

Отбор (селекция)

На этапе отбора нужно из всей популяции выбрать определённую её долю, которая останется «в живых» на этом этапе эволюции. Есть разные способы проводить отбор. Вероятность выживания особи **h** должна зависеть от значения функции приспособленности **Fitness(h)**. Сама доля выживших **s** обычно является параметром генетического алгоритма, и её просто задают заранее. По итогам отбора из **N** особей популяции **H** должны остаться **sN** особей, которые войдут в итоговую популяцию **H'**. Остальные особи погибают.

Турнирная селекция — сначала случайно выбирается установленное количество особей (обычно две), а затем из них выбирается особь с лучшим значением функции приспособленности

Метод рулетки — вероятность выбора особи тем вероятнее, чем лучше её значение функции приспособленности

- Турнирная селекция — сначала случайно выбирается установленное количество особей (обычно две), а затем из них выбирается особь с лучшим значением функции приспособленности
- Метод рулетки — вероятность выбора особи тем вероятнее, чем лучше её значение функции приспособленности

$$p_i = \frac{f_i}{\sum_{i=1}^{N} f_i},$$

где p_i — вероятность выбора i особи,
f_i — значение функции приспособленности для i особи,
N — количество особей в популяции

- Метод ранжирования — вероятность выбора зависит от места в списке особей, отсортированном по значению функции приспособленности

$$p_i = \frac{1}{N}\left(a - (a-b)\frac{i-1}{N-1}\right),$$

где $a \in [1,2], b = 2 - a, i$ — порядковый номер особи в списке особей, отсортированном по значению функции приспособленности (то есть $\forall i \forall j > i\ f_i \leq f_j$ — если мы минимизируем значение функции приспособленности)

- Равномерное ранжирование — вероятность выбора особи определяется выражением:

$$p_i = \begin{cases} \frac{1}{\mu},\ if 1 \leq i \leq \mu \\ 0,\ if \mu < i \leq N \end{cases},$$

где $\mu \leq N$ параметр метода

- Сигма-отсечение — для предотвращения преждевременной сходимости генетического алгоритма используются методы,

масштабирующие значение целевой функции. Вероятность выбора особи тем больше, чем оптимальнее значение масштабируемой целевой функции

$$p_i = \frac{F_i}{\sum_{i=1}^{N} F_i},$$

где $F_i = 1 + \frac{f_i - f_{avg}}{2\sigma}$ — среднее значение целевой функции для всей популяции, σ — среднеквадратичное отклонение значения целевой функции.

Выбор родителей

Размножение в генетических алгоритмах требует для производства потомка нескольких родителей, обычно двух.

Можно выделить несколько операторов выбора родителей:

1. Панмиксия — оба родителя выбираются случайно, каждая особь популяции имеет равные шансы быть выбранной;
2. Инбридинг — первый родитель выбирается случайно, а вторым выбирается такой, который наиболее похож на первого родителя;
3. Аутбридинг — первый родитель выбирается случайно, а вторым выбирается такой, который наименее похож на первого родителя.

Инбридинг и аутбридинг бывают в двух формах: фенотипной и генотипной. В случае фенотипной формы похожесть измеряется в зависимости от значения функции приспособленности (чем ближе значения целевой функции, тем особи более похожи), а в случае генотипной формы похожесть измеряется в зависимости от представления генотипа (чем меньше отличий между генотипами особей, тем особи похожее).

Размножение (скрещивание)

Размножение в разных алгоритмах определяется по-разному — оно, конечно, зависит от представления данных. Главное требование к размножению — чтобы потомок или потомки имели возможность унаследовать черты обоих родителей, «смешав» их каким-либо способом.

Почему особи для размножения обычно выбираются из всей популяции **H**, а не из выживших на первом шаге элементов **H'** (хотя последний вариант тоже имеет право на существование)? Дело в том, что главный недостаток многих генетических алгоритмов — отсутствие разнообразия в особях. Достаточно быстро выделяется один-единственный генотип, который представляет собой локальный максимум, а затем все элементы популяции проигрывают ему отбор, и вся популяция «забивается» копиями этой особи. Есть разные способы борьбы с таким нежелательным эффектом; один из них — выбор для размножения не самых приспособленных, но вообще всех особей. Однако такой подход вынуждает хранить всех существовавших ранее особей, что увеличивает вычислительную сложность задачи. Поэтому часто применяют методы отбора особей для скрещивания таким образом, чтобы «размножались» не только самые приспособленные, но и другие особи,

обладающие плохой приспособленностью. При таком подходе для разнообразия генотипа возрастает роль мутаций.

Мутации

К мутациям относится все то же самое, что и к размножению: есть некоторая доля мутантов **m**, являющаяся параметром генетического алгоритма, и на шаге мутаций нужно выбрать **mN** особей, а затем изменить их в соответствии с заранее определёнными операциями мутации.

Пример простой реализации на C++

Поиск в одномерном пространстве, без скрещивания.

```
#include <cstdlib>
#include <ctime>
#include <algorithm>
#include <iostream>
#include <numeric>

int main()
{
    srand((unsigned int)time(NULL));
    const size_t N = 1000;
    int a[N] = { 0 };
    for ( ; ; )
    {
        //мутация в случайную сторону каждого элемента:
        for (size_t i = 0; i < N; ++i)
            a[i] += ((rand() % 2 == 1) ? 1 : -1);

        //теперь выбираем лучших, отсортировав по возрастанию
        std::sort(a, a + N);
        //и тогда лучшие окажутся во второй половине массива.
        //скопируем лучших в первую половину, куда они оставили потомство, а 
первые умерли:
        std::copy(a + N / 2, a + N, a);
        //теперь посмотрим на среднее состояние популяции. Как видим, оно всё 
лучше и лучше.
        std::cout << std::accumulate(a, a + N, 0) / N << std::endl;
    }
}
```

Библиотека TensorFlow

TensorFlow — это опенсорсная библиотека, созданная Google, которая используется при разработке систем, использующих технологии машинного обучения. Эта библиотека включает в себя реализацию множества мощных алгоритмов, рассчитанных на решение распространённых задач машинного обучения, среди которых можно отметить распознавание образов и принятие решений.

Использование различных слоев и функций

TensorFlow предоставляет большой набор готовых слоев и функций для построения нейронных сетей:

- Полносвязные слои (Dense) - основа для создания полносвязных сетей;
- Сверточные слои (Conv2D, Conv3D) - используются в сверточных сетях для работы с изображениями;
- Пулинг слои (MaxPooling2D) - применяются после сверточных для снижения размерности;
- Рекуррентные слои (LSTM, GRU) - для сетей RNN и обработки последовательностей;
- Слои нормализации (BatchNormalization) - для нормализации активаций в сети;
- Функции активации (ReLU, LeakyReLU, Sigmoid, Softmax и др.) - добавляют нелинейность;
- Функции потерь (Losses) - MSE, CrossEntropy, SparseCategoricalCrossentropy и др.;
- Оптимизаторы (Optimizers) - Adam, SGD, RMSprop и др. для обновления весов.

Комбинируя разные слои, можно создавать нейросети практически любой архитектуры для решения широкого круга задач.

Визуализация работы нейросети в TensorFlow

Чтобы лучше понимать и отлаживать нейронные сети, очень полезно визуализировать их работу. В TensorFlow есть инструменты для визуализации:

TensorBoard - позволяет строить графики потерь, метрик, весов нейронов в процессе обучения:

```
tensorboard = TensorBoard(log_dir="logs")

model.fit(data, labels, epochs=10, callbacks=[tensorboard])
```

TensorBoard запускается в браузере и отображает графики из логов обучения.

Кроме того, можно визуализировать активации конкретных нейронов при обработке входных данных с помощью Keras:

```
layer = model.layers[2]
activations = layer.activations

import matplotlib.pyplot as plt
plt.imshow(activations[0][0,:,:], cmap='viridis')
```

Это помогает понимать, на что именно реагируют нейроны в разных слоях сети.

Визуализация значительно упрощает отладку и оптимизацию нейронных сетей в TensorFlow.

Библиотека Keras

Keras — это высокоуровневая нейро-сетевая библиотека для Python, которая может использовать TensorFlow в качестве бэкенда.

Keras имеет простой и понятный API для быстрой разработки нейронных сетей.

Препроцессинг

Keras содержит в себе инструменты для удобного препроцессинга текстов, картинок и временных рядов, иными словами, самых распространенных типов данных. Необходимо разбить их на токены и привести в матричную форму.

```
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(newsgroups_train["data"])  # теперь токенизатор знает словарь для этого корпуса текстов

x_train = tokenizer.texts_to_matrix(newsgroups_train["data"], mode='binary')
x_test = tokenizer.texts_to_matrix(newsgroups_test["data"], mode='binary')
```

На выходе получаются бинарные матрицы таких размеров:

```
x_train shape: (11314, 1000)
x_test shape: (7532, 1000)
```

Первое число — количество документов в выборке, а второе — размер словаря (одна тысяча в этом примере).

Еще понадобится преобразовать метки классов к матричному виду для обучения с помощью кросс-энтропии. Для этого необходимо перевести номер класса в так называемый one-hot вектор, т.е. вектор, состоящий из нулей и одной единицы:

```
y_train = keras.utils.to_categorical(newsgroups_train["target"], num_classes)
y_test = keras.utils.to_categorical(newsgroups_test["target"], num_classes)
```

На выходе получим также бинарные матрицы вот таких размеров:

```
y_train shape: (11314, 20)
y_test shape: (7532, 20)
```

Из этого видно, что размеры этих матриц частично совпадают с матрицами данных (по первой координате — числу документов в обучающей и тестовой выборках), а частично — нет. По второй координате стоит число классов (20, как следует из названия датасета).

Модель

Модель в Keras можно описать двумя основными способами:

- Последовательный (Sequential) - для линейных стеков слоев, например, вот так:

```
model = Sequential()
model.add(Dense(512, input_shape=(max_words,)))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

или вот так:

```
model = Sequential([
          Dense(512, input_shape=(max_words,)),
          Activation('relu'),
          Dropout(0.5),
          Dense(num_classes),
          Activation('softmax')
        ])
```

- Функциональный (Functional) - для произвольных графов слоев

```
a = Input(shape=(max_words,))
b = Dense(512)(a)
b = Activation('relu')(b)
b = Dropout(0.5)(b)
b = Dense(num_classes)(b)
b = Activation('softmax')(b)
model = Model(inputs=a, outputs=b)
```

Принципиального отличия между способами нет.

Класс `Model` (и унаследованный от него `Sequential`) имеет удобный интерфейс, позволяющий посмотреть, какие слои входят в модель — `model.layers`, входы — `model.inputs`, и выходы — `model.outputs`.

Также очень удобный метод отображения и сохранения модели — `model.to_yaml`.

Это позволяет сохранять модели в читаемом виде, а также инстанциировать модели из такого описания:

```
from keras.models import model_from_yaml

yaml_string = model.to_yaml()
model = model_from_yaml(yaml_string)
```

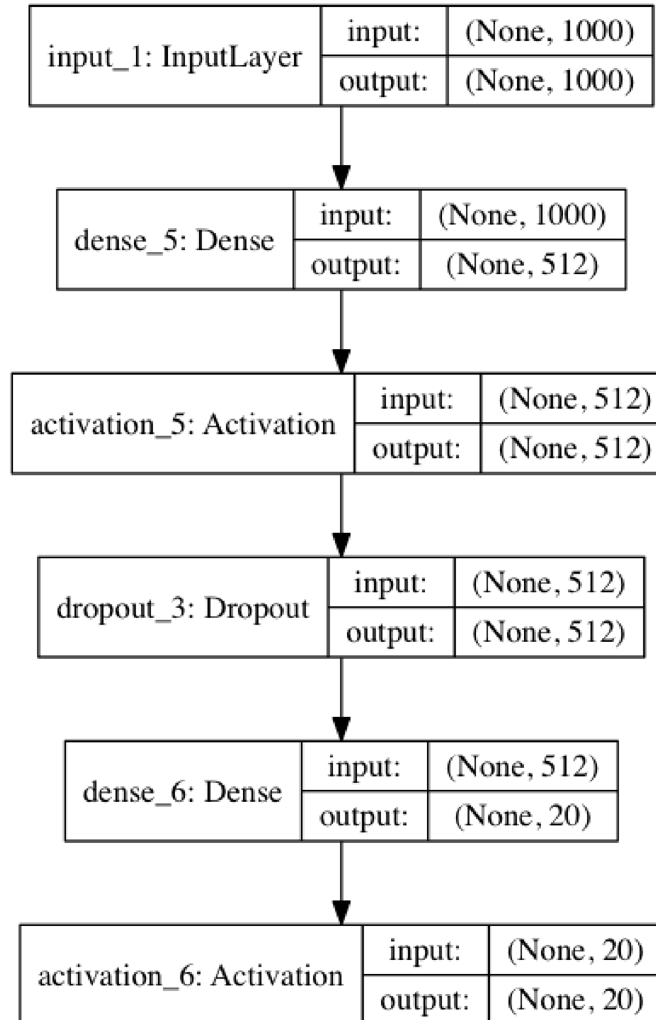
Важно отметить, что модель, сохраненная в текстовом виде (кстати, возможно сохранение также и в JSON) не содержит весов. Для сохранения и загрузки весов необходимо использовать функции `save_weights` и `load_weights` соответственно.

Визуализация модели

Keras имеет встроенную визуализацию для моделей:

```
from keras.utils import plot_model
plot_model(model, to_file='model.png', show_shapes=True)
```

Этот код сохранит под именем model.png вот такую картинку:



Здесь дополнительно отображены размеры входов и выходов для слоев. None, идущий первым в кортеже размеров — это размерность батча. Т.к. стоит None, то батч может быть произвольным.

Установка и настройка сервера

Перед установкой серверов вам необходимо скачать и установить на компьютер следующий список программ: python, anaconda,VSCOD

Данные программы можно скачать по следующим ссылкам:

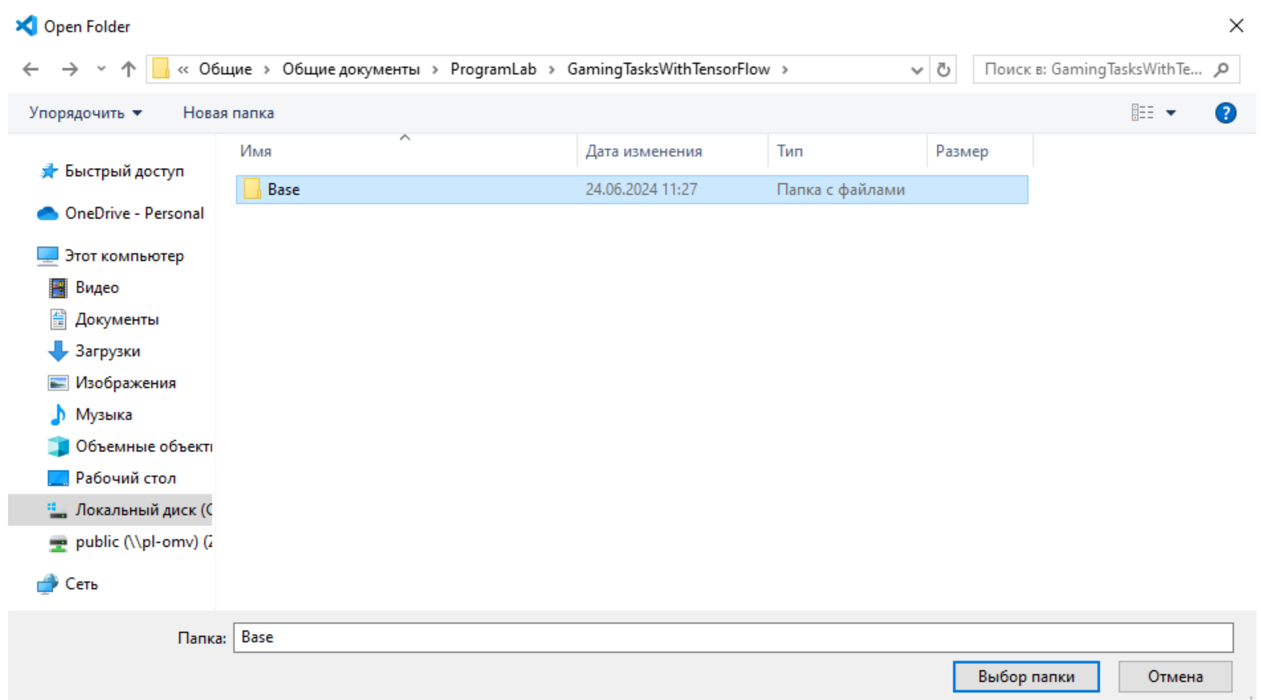
https://code.visualstudio.com/docs/?dv=win64user

https://www.python.org/downloads/release/python-3124/

После установки всех программ перезапустите компьютер.

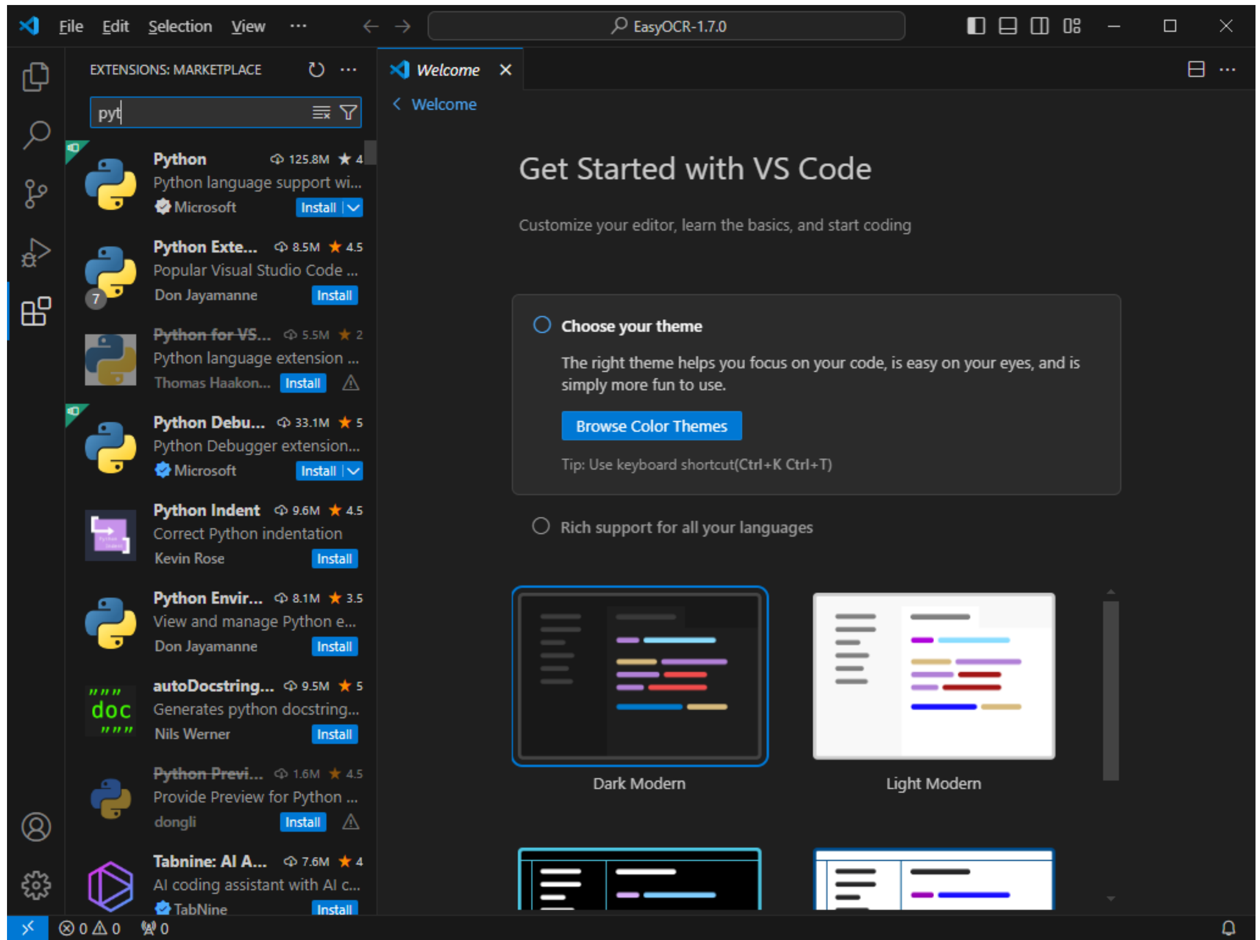
Когда все необходимые программы установлены, запустите Visual studio code, нажмите на вкладку File -> Open folder и выберите путь к папке, где располагается необходимый вам сервер (при установке ПО сервер по умолчанию находится в папке проекта:

C:\Users\Public\Documents\ProgramLab\GamingTasksWithTensorFlow\Base).



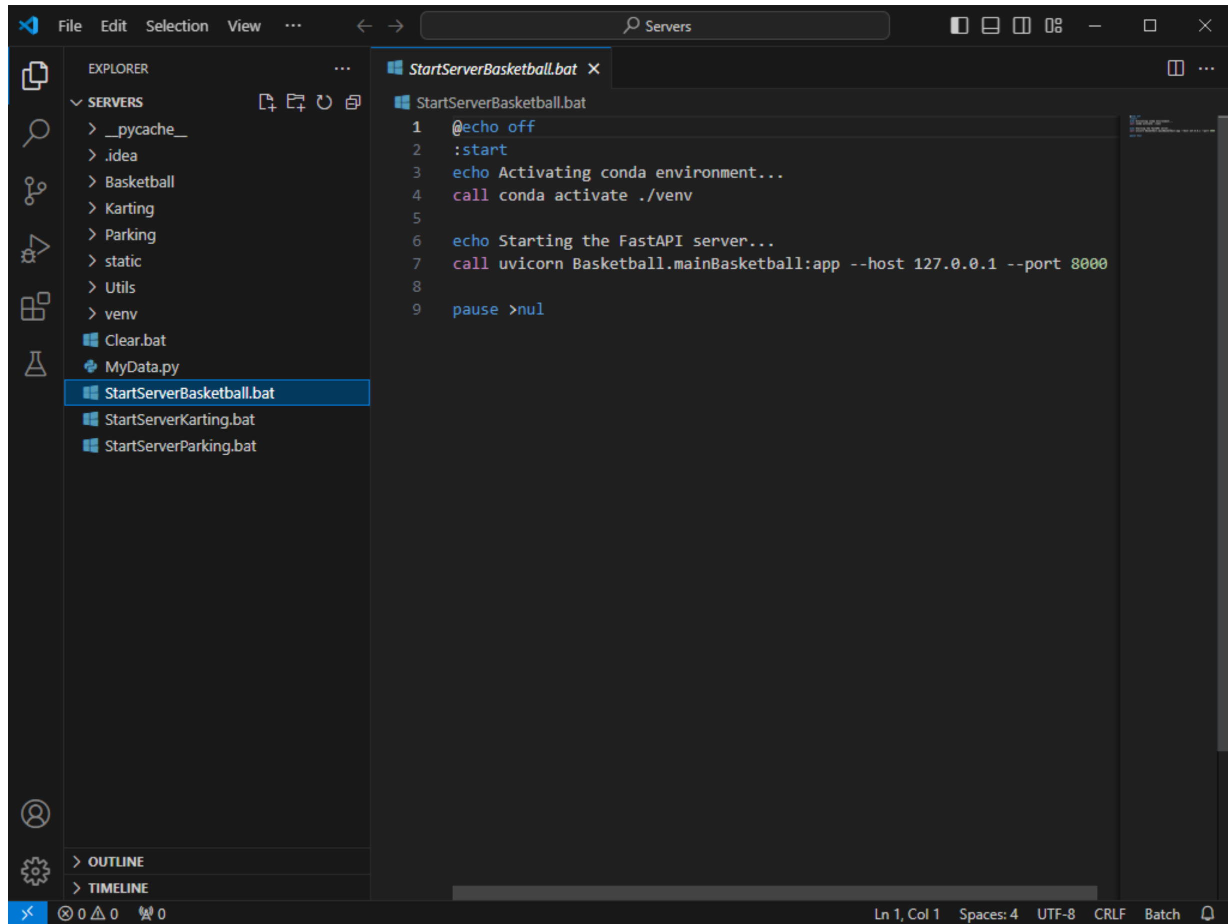
Открытие папки с сервером

В открытом Visual studio code перейдите во вкладку Extensions и в нем установите Python.



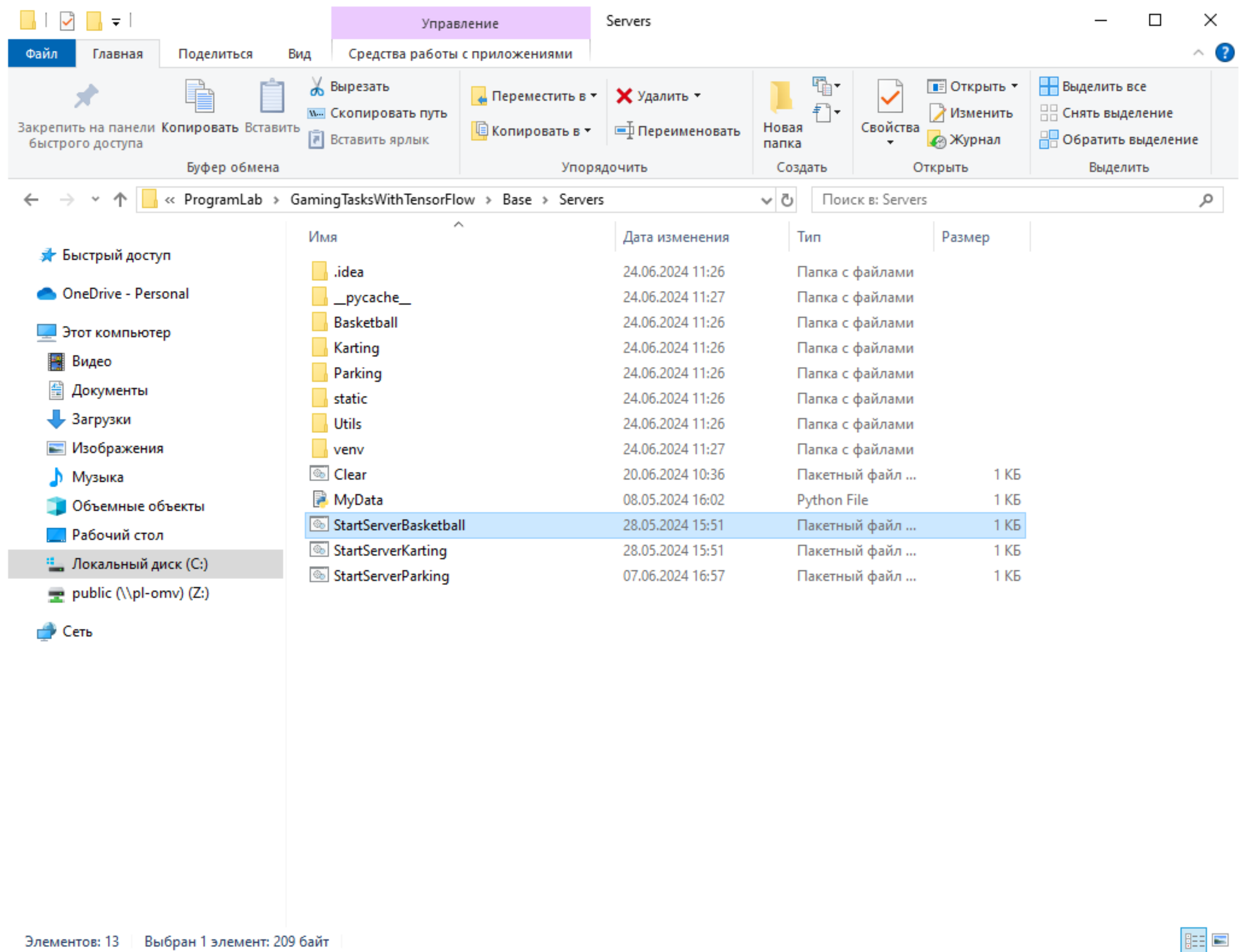
Установка Python

Вы можете изменить IP адрес сервера, если захотите запустить сервер на отдельном компьютере, для этого зайдите в файл start и поменяйте IP адрес на IP адрес компьютера, на котором будет расположен сервер.



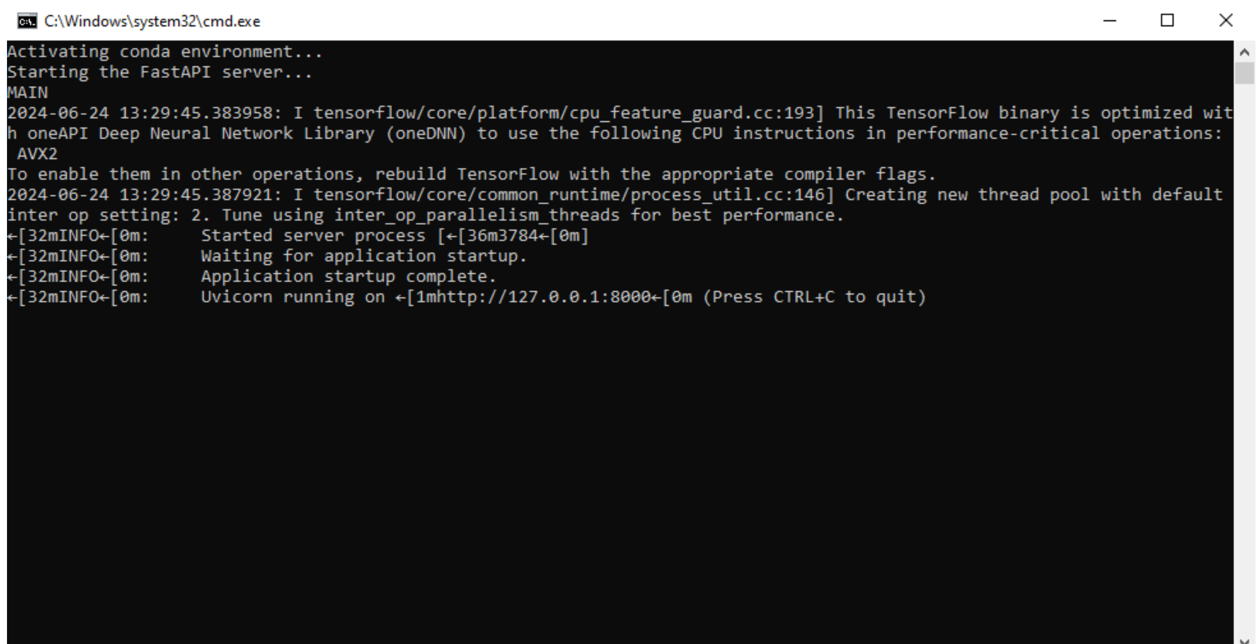
Смена IP адреса

Сохраните изменения, теперь необходимо проверить правильность настроек и работы сервера, для чего перейдите в папку местонахождения сервера и запустите файл *start.bat.*



Запуск сервера

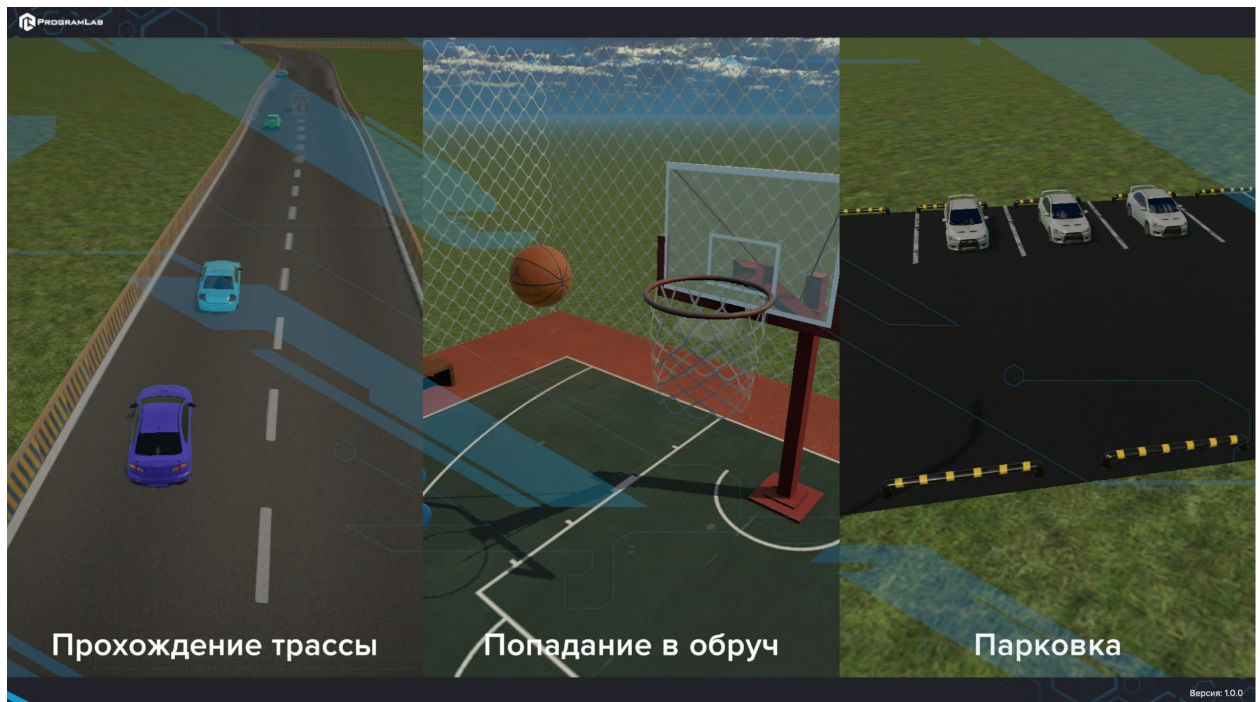
Запустите сервер запуском файла *start.bat*, откроется командная строка, в которой появятся информация о запуске сервера.



Запуск сервера

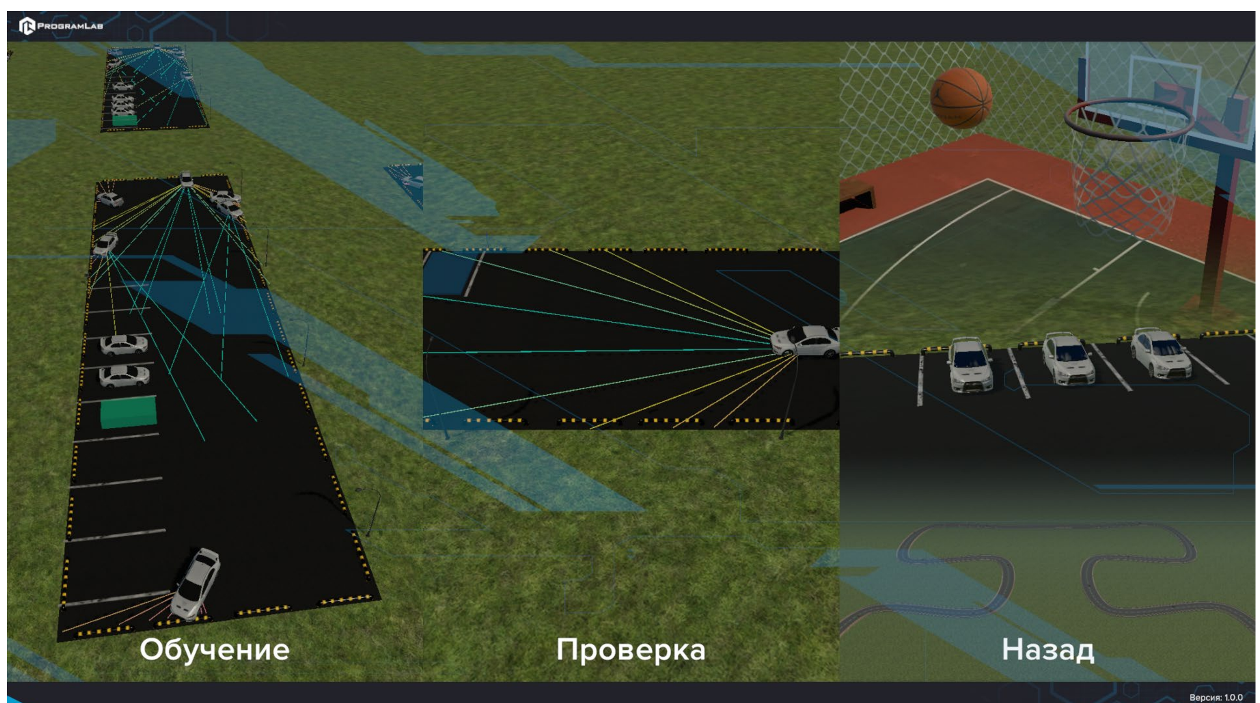
Работа в программе

После запуска программы откроется следующий интерфейс



Начальный интерфейс

Необходимо выбрать один из модулей, после чего откроется следующее меню



Меню режимов

В этом меню необходимо выбрать один из режимов, **Обучение** или **Проверка**, для возвращения в предыдущее меню нажмите **Назад**

Выбрав режим **Обучение** откроется следующий интерфейс

Режим Обучение

1 – Нажмите чтобы создать новую модель;

2 – Нажмите чтобы выбрать готовую модель;

3 – Нажмите чтобы сохранить модель;

4 – Нажмите чтобы запустить обучение;

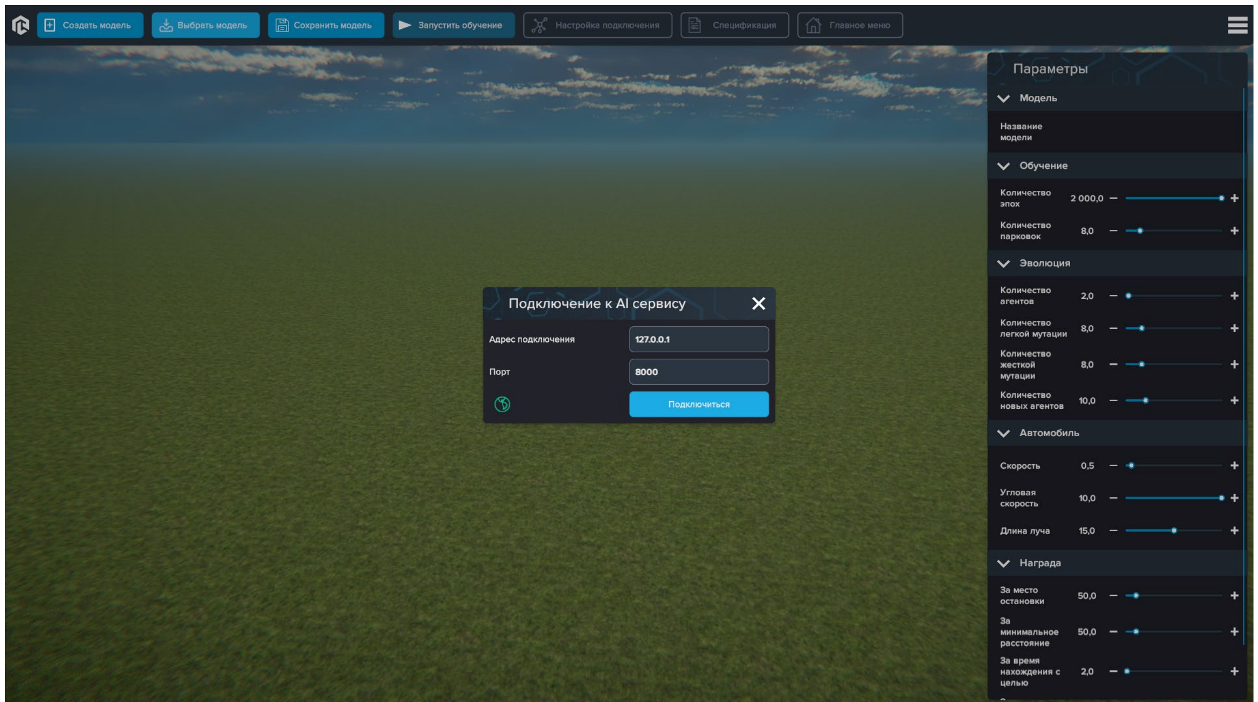
5 – Нажмите чтобы настроить подключение;

6 – Нажмите чтобы открыть спецификацию;

7 – Нажмите чтобы вернуться в главное меню;

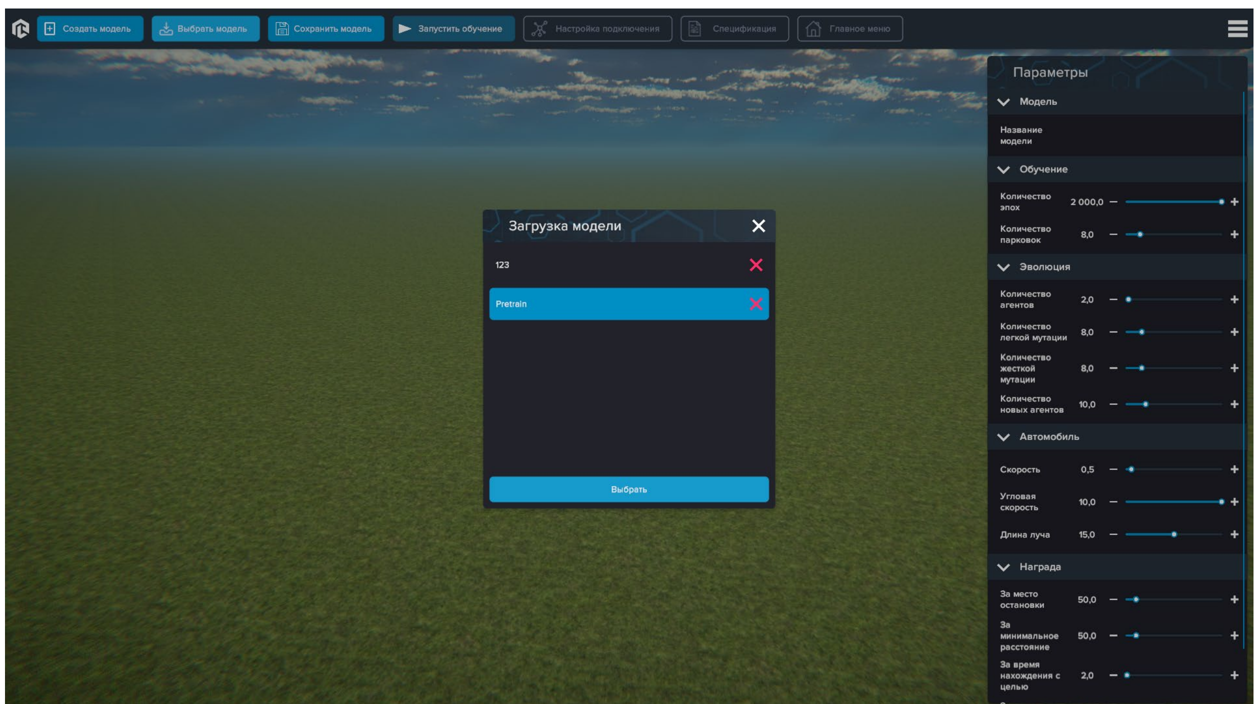
8 – Окно параметров, у каждого из модулей свои параметры, которые меняются на нужные значения.

Перед началом работы необходимо запустить сервер выбранной задачи как описано в пункте **Установка и настройка сервера**, после чего выбрать **Настройка подключения** и нажать **Подключиться**, должен появиться зеленый значок, означающий что подключение прошло успешно.



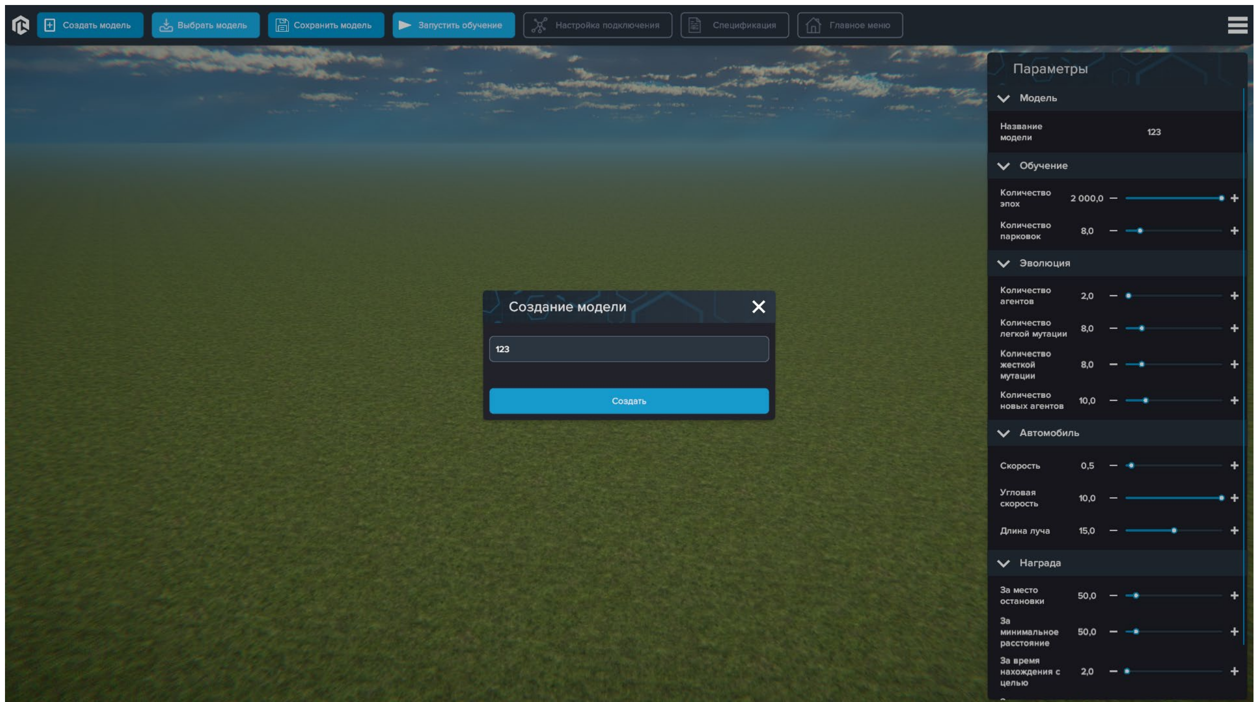
Подключение к AI сервису

После подключение можно создать новую модель или выбрать готовую



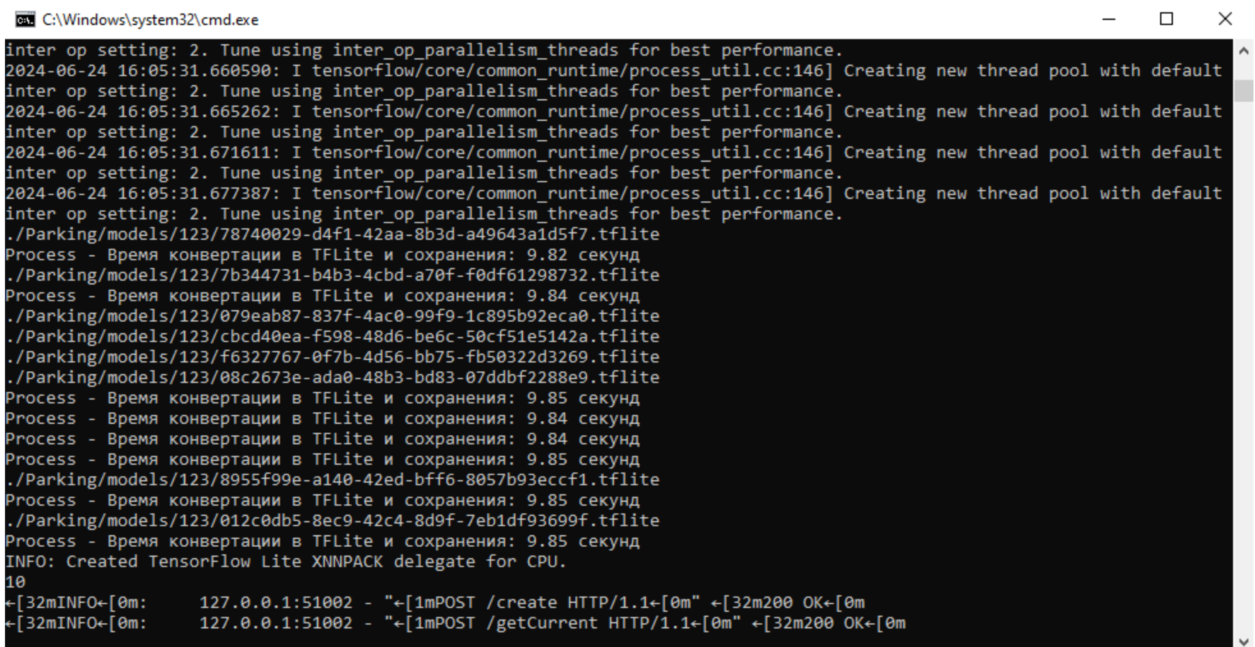
Выбор готовой модели

Выбрав создать модель откроется следующее окно



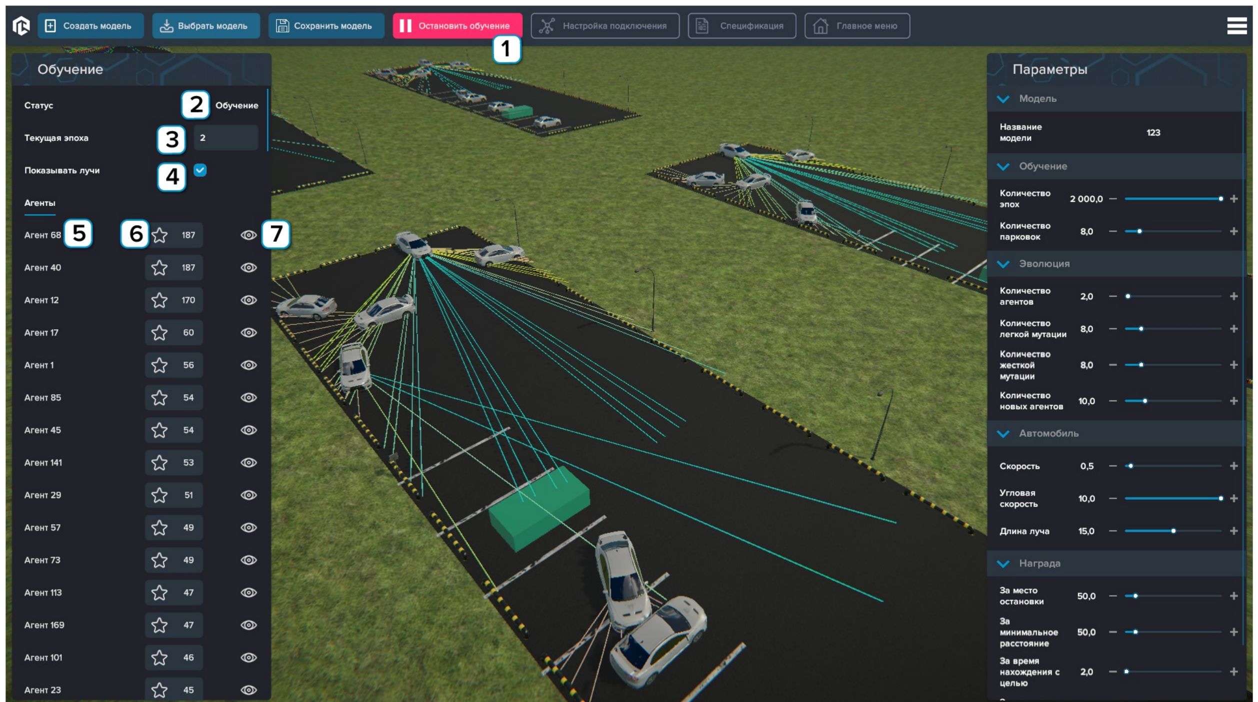
Окно создания новой модели

Нажав создать, на сервере начнется процесс создания модели, необходимо дождаться пока он не будет завершен



Процесс создания модели

После создания модели необходимо нажать **Запустить обучение**, после чего запустится процесс обучения



Процесс обучения модели

1 – Нажмите чтобы остановить обучение;

2 – Текущий статус обучения;

3 – Текущий этап обучения;

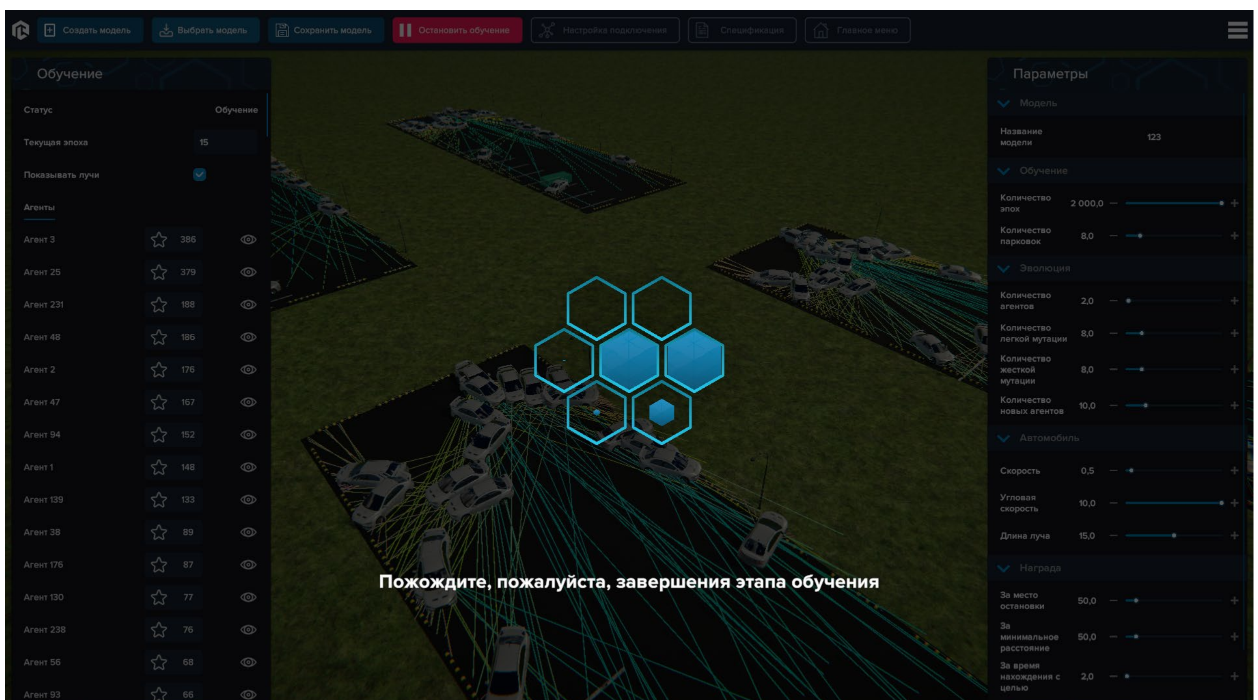
4 – Нажмите чтобы включить/отключить показ лучей;

5 – Показывает номер агента;

6 – Показывает количество наград агента;

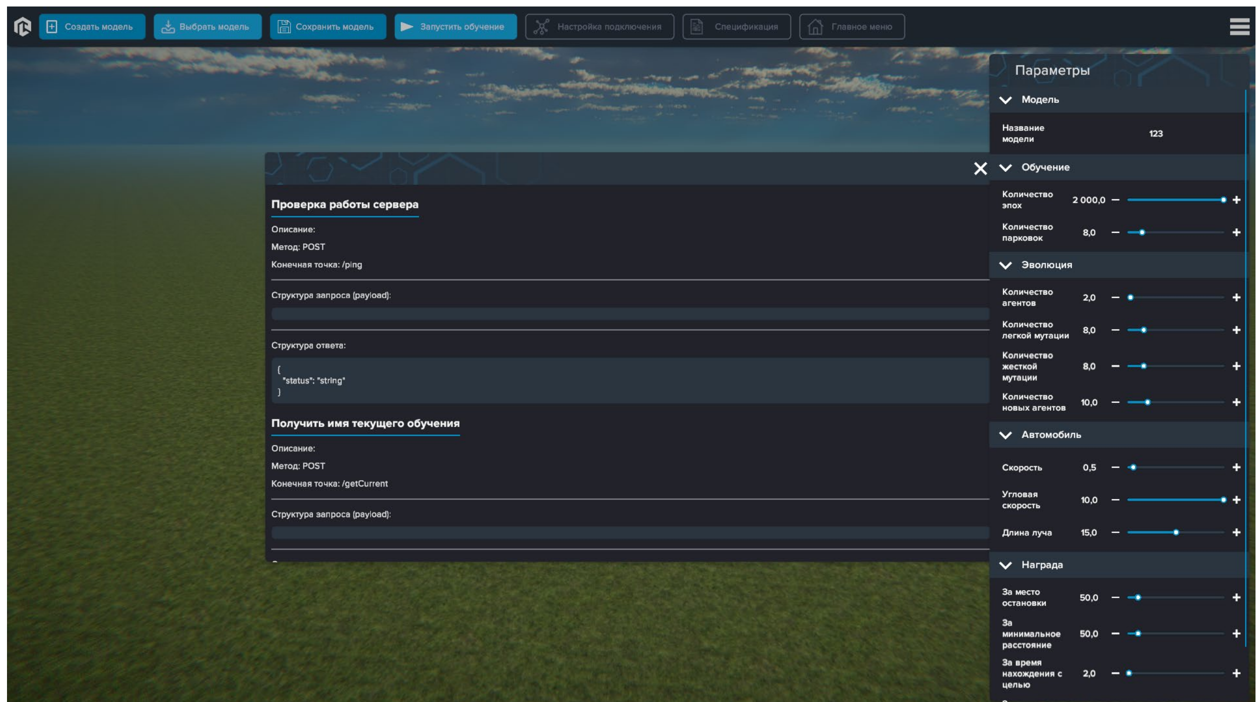
7 – Переводит камеру на агента.

Чтобы завершить обучение нажмите **Остановить обучение**, после чего необходимо дождаться завершения



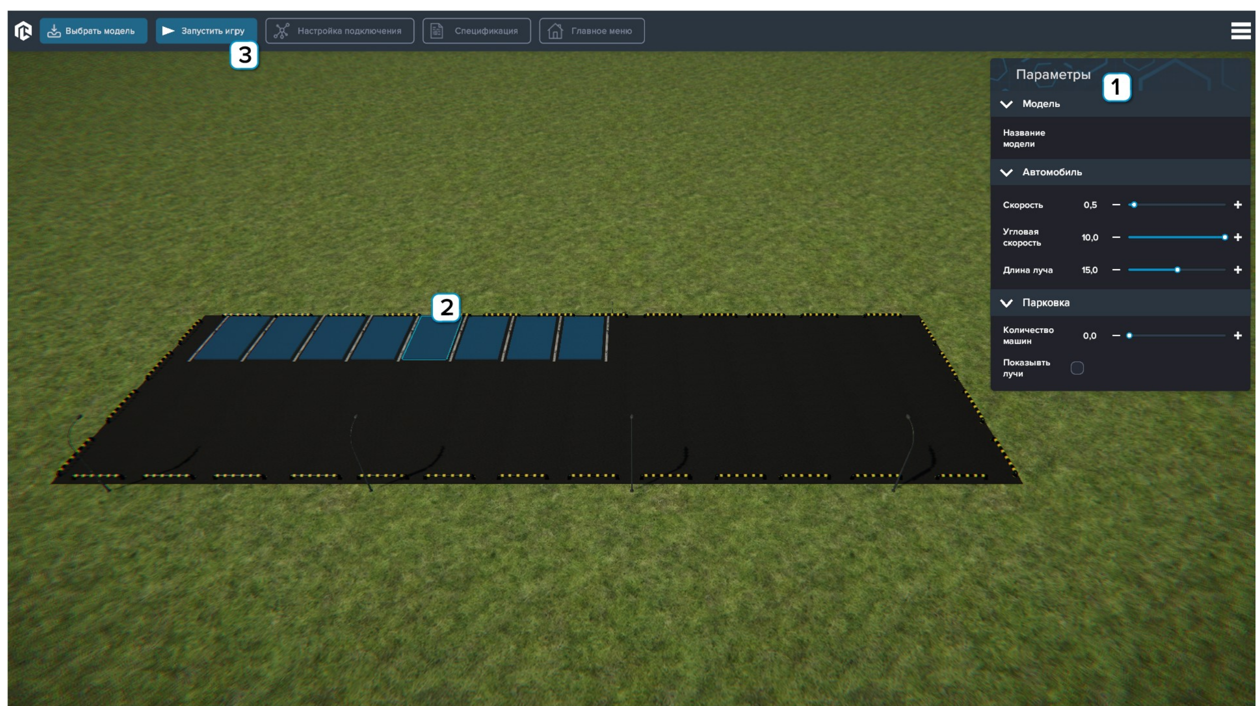
Остановка обучения

При нажатии на кнопку спецификация откроется окно с спецификацией отправляемых и получаемых запросов.



Спецификация

Выбрав режим **Проверка** откроется следующий интерфейс



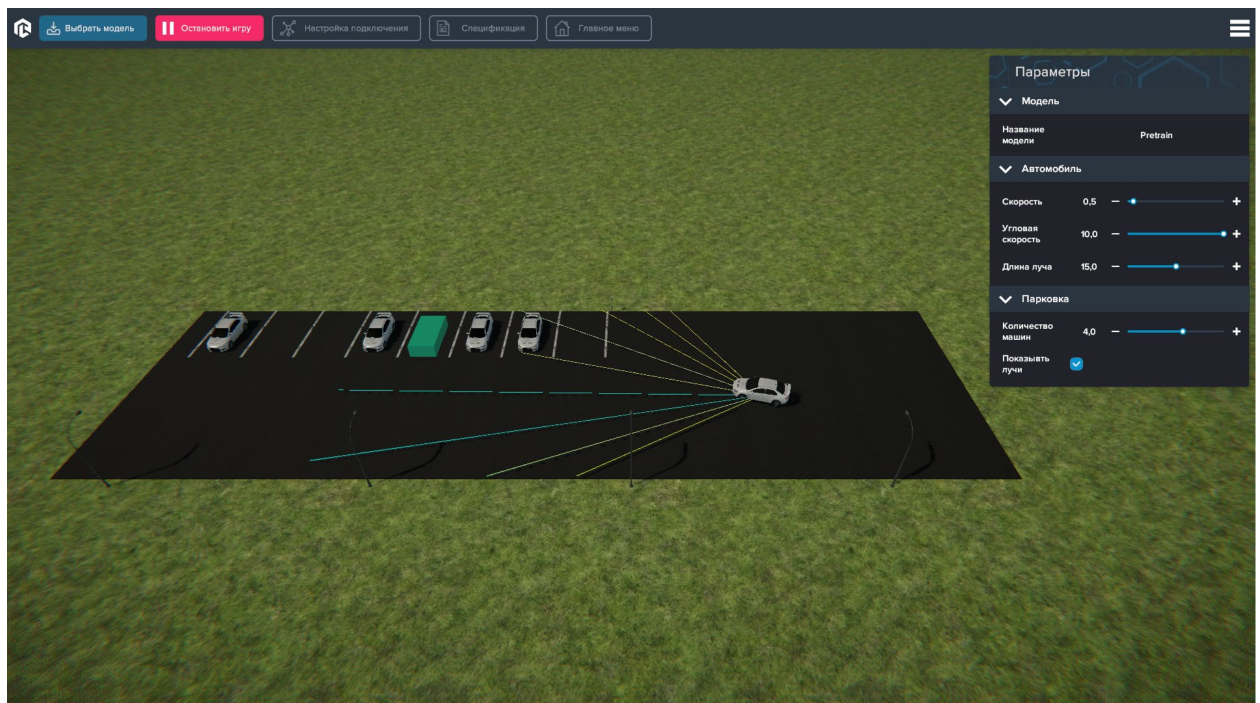
Режим Проверка

1 – Параметры модели;

2 – Выбор конечной точки для завершения задачи;

3 – Нажмите чтобы запустить проверку.

Выполняется последовательность действий, как и при режиме обучения, необходимо запустить нужный сервер, выполнить подключение в программе, выбрать модель и при необходимости изменить параметры. После выполнения всех действий нажать **Запустить игру**



Процесс выполнения проверки

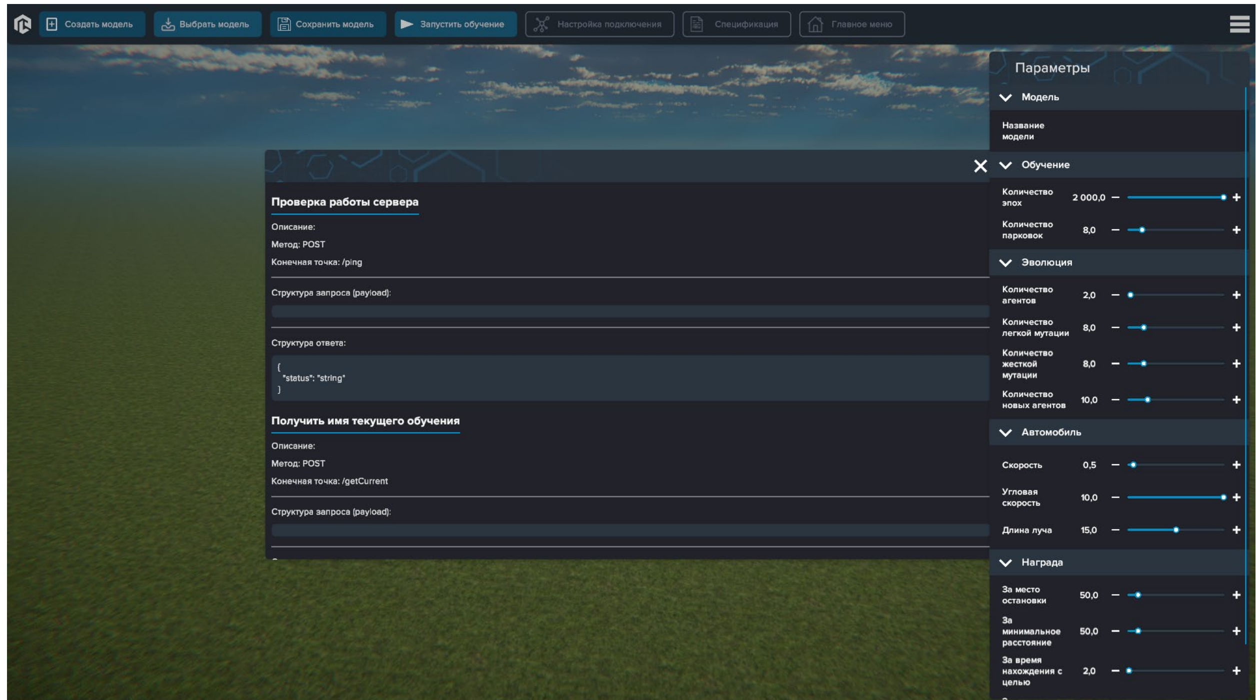
Нажмите **Остановить игру** для сброса текущей проверки

Для других модулей выполняются аналогичные действия.

Спецификация API Парковка

В приложении вы можете ознакомиться со спецификацией отправки и получения запросов приложения от приложения к серверу.

При нажатии на вкладку спецификации откроется окно с основными запросами и ответами от приложения к серверу.



Спецификация

Проверка работы сервера

Команда ***Пинг сервера***, приложение посылает запрос к конечной точке ***/ping***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON ответа:

```
1 {
2   "status": "string"              //Ответ работает ли сервер
3 }
```

Получить имя текущего обучения

Команда ***Получить имя текущего обучения***, приложение посылает запрос к конечной точке ***/getCurrent***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON ответа:

```
1 {
2   "name": "string" // имя текущего обучения
3 }
```

Получить имена обучения

Команда ***Получить имена обучения***, приложение посылает запрос к конечной точке ***/getAll***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON ответа:

```
1 {
2   "names": [ // список всех обучений
3     {
4       "name": "string" // имя обучения
5     }
6   ]
7 }
```

Создать новое обучение

Команда ***Создать новое обучение***, приложение посылает запрос к конечной точке ***/create***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON запроса:

```
1 {
2   "name": "string" //имя нового обучения
3 }
```

Пример JSON ответа:

```
1 {
2   "status": "string" //Состояние создания нового обучения
3 }
```

Сохранить текущее обучение

Команда ***Сохранить текущее обучение***, приложение посылает запрос к конечной точке ***/save***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON ответа:

```
1 {
2   "status": "string" //Состояние сохранения обучения
3 }
```

Загрузить обучение

Команда ***Загрузить обучение***, приложение посылает запрос к конечной точке ***/load***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON запроса:

```
1 {
2   "name": "string" //имя нового обучения
3 }
```

Пример JSON ответа:

```
1 {
2   "status": "string" //Состояние загрузки обучения
3 }
```

Удалить обучение

Команда ***Удалить обучение***, приложение посылает запрос к конечной точке ***/delete***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON запроса:

```
1 {
2   "name": "string" //имя нового обучения
3 }
```

Пример JSON ответа:

```
1 {
2   "status": "string" //Состояние удаления обучения
3 }
```

Установить параметры обучения

Команда ***Установить параметры обучения***, приложение посылает запрос к конечной точке ***/setParametersTraining***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON запроса:

```
1 {
2   "population_size": 0, // общее количество агентов
3   "n_parents_from_population": 0, // количество агентов из предыдущей итерации
4   "countLightMutationPopulation": 0, // количество агентов из предыдущей
5 итерации с небольшими изменениями
6   "countHardMutationPopulation": 0,// количество агентов из предыдущей
7 итерации с большими изменениями
```

```
8   "countNewPopulation": 0// количество новых агентов
9 }
```

Пример JSON ответа:

```
1 {
2   "status": "string" //Состояние настройки обучения
3 }
```

Получить id агентов

Команда ***Получить id агентов***, приложение посылает запрос к конечной точке ***/getGuids***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON ответа:

```
1 {
2  // список идентификаторов обучения
3   "guids": [
4     "string" // уникальный идентификатор агента
5   ]
6 }
```

Запустить обучение

Команда ***Запустить обучение***, приложение посылает запрос к конечной точке ***/fitModels***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON ответа:

```
1 {
2   "status": "string" // Окончание обучения агентов
3 }
```

Обновление окружения

Команда ***Обновление окружения***, приложение посылает запрос к конечной точке ***/updateEnviromnetParking***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON запроса:

```
1 {
2   "data": [
3     {
4       "guidEnv": "string", // идентификатор трассы
5       "guidNN": "string", // идентификатор агента
6       "rays": [ // 32 нормализованых расстояний до препятствий
```

```
7        0
8      ]
9    }
10  ]
11 }
```

Набор параметров окружения. Каждая машина ориентируется по 9 расстояниям до препятствий перед собой и хранит 18 прошлых значения расстояний.

28 параметр — это направления до цели по оси x

29 параметр — это направления до цели по оси z

30 параметр — это дистанция до цели

31 параметр — это направление движения по оси x

32 параметр — это направления движения по оси z

Пример JSON ответа:

```
1  {
2   "moveSteps": [
3     {
4       "guidEnv": "string", // идентификатор трассы
5       "guidNN": "string", // идентификатор агента
6       "angle": 0, // нормализованный угол поворота
7       "speed": 0 // нормализованная скорость
8     }
9   ]
10 }
```

Обновление награды

Команда ***Обновление награды***, приложение посылает запрос к конечной точке ***/updateRewardParking***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON запроса:

```
1 {
2  "rewards": [
3    {
4      "guidNN": "string", // идентификатор агента
5      "reward": 0// награда
6    }
7  ]
8 }
```

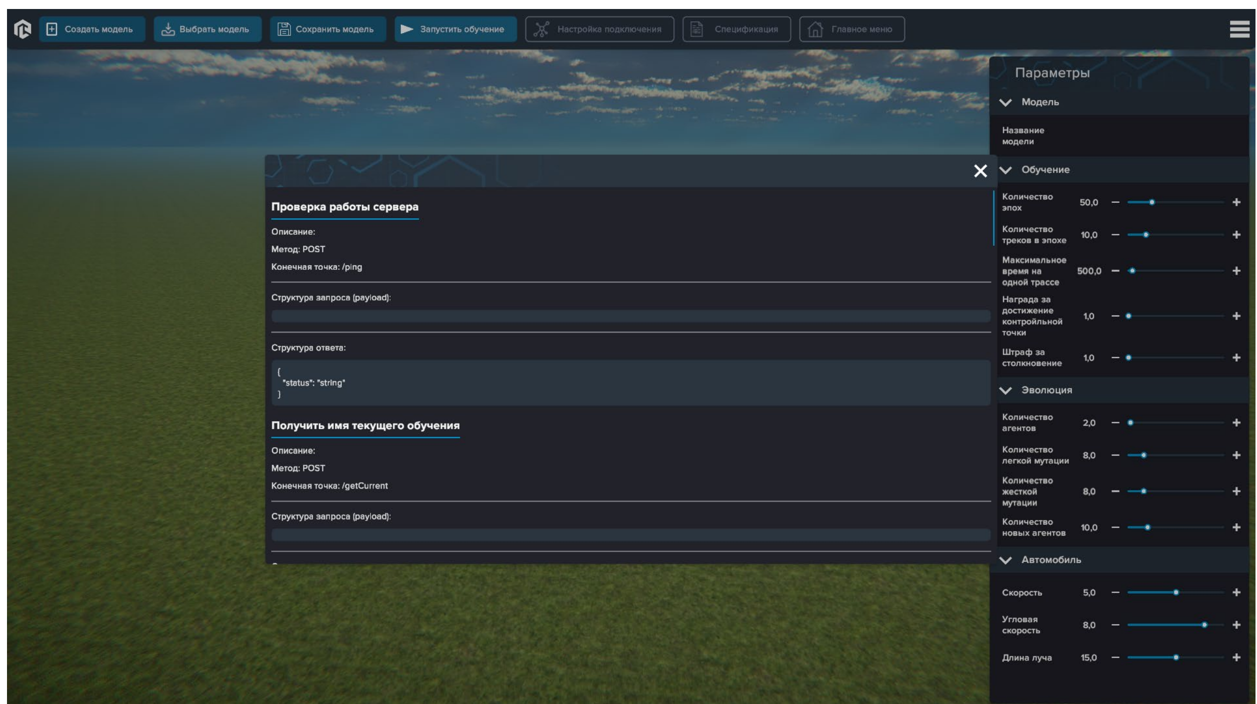
Пример JSON ответа:

```
1 {
2   "status": "string" //Окончание обновления награды
3 }
```

Спецификация API Прохождение трассы

В приложении вы можете ознакомиться со спецификацией отправки и получения запросов приложения от приложения к серверу.

При нажатии на вкладку спецификации откроется окно с основными запросами и ответами от приложения к серверу.



Спецификация

Проверка работы сервера

Команда ***Пинг сервера***, приложение посылает запрос к конечной точке ***/ping***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON ответа:

```
1 {
2   "status": "string"            //Ответ работает ли сервер
3 }
```

Получить имя текущего обучения

Команда ***Получить имя текущего обучения***, приложение посылает запрос к конечной точке ***/getCurrent***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON ответа:

```
1 {
2   "name": "string" // имя текущего обучения
3 }
```

Получить имена обучения

Команда ***Получить имена обучения***, приложение посылает запрос к конечной точке ***/getAll***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON ответа:

```
1 {
2   "names": [ // список всех обучений
3     {
4       "name": "string" // имя обучения
5     }
6   ]
7 }
```

Создать новое обучение

Команда ***Создать новое обучение***, приложение посылает запрос к конечной точке ***/create***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON запроса:

```
1 {
2   "name": "string" //имя нового обучения
3 }
```

Пример JSON ответа:

```
1 {
2   "status": "string" //Состояние создания нового обучения
3 }
```

Сохранить текущее обучение

Команда ***Сохранить текущее обучение***, приложение посылает запрос к конечной точке ***/save***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON ответа:

```
1 {
2   "status": "string" //Состояние сохранения обучения
3 }
```

Загрузить обучение

Команда ***Загрузить обучение***, приложение посылает запрос к конечной точке ***/load***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON запроса:

```
1 {
2   "name": "string" //имя нового обучения
3 }
```

Пример JSON ответа:

```
1 {
2   "status": "string" //Состояние загрузки обучения
3 }
```

Удалить обучение

Команда ***Удалить обучение***, приложение посылает запрос к конечной точке ***/delete***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON запроса:

```
1 {
2   "name": "string" //имя нового обучения
3 }
```

Пример JSON ответа:

```
1 {
2   "status": "string" //Состояние удаления обучения
3 }
```

Установить параметры обучения

Команда ***Установить параметры обучения***, приложение посылает запрос к конечной точке ***/setParametersTraining***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON запроса:

```
1 {
2  "population_size": 0, // общее количество агентов
3  "n_parents_from_population": 0, // количество агентов из предыдущей итерации
4  "countLightMutationPopulation": 0, // количество агентов из предыдущей
5 итерации с небольшими изменениями
6  "countHardMutationPopulation": 0, // количество агентов из предыдущей
7 итерации с большими изменениями
8  "countNewPopulation": 0 // количество новых агентов
9 }
```

Пример JSON ответа:

```
1 {
2  "status": "string" //Состояние настройки обучения
3 }
```

Получить id агентов

Команда ***Получить id агентов***, приложение посылает запрос к конечной точке ***/getGuids***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON ответа:

```
1 {
2 // список идентификаторов обучения
3  "guids": [
4   "string"// уникальный идентификатор агента
5  ]
6 }
```

Запустить обучение

Команда ***Запустить обучение***, приложение посылает запрос к конечной точке ***/fitModels***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON ответа:

```
1 {
2  "status": "string" // Окончание обучения агентов
3 }
```

Обновление окружения

Команда ***Обновление окружения***, приложение посылает запрос к конечной точке ***/updateEnviromnet***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON запроса:

```
1  {
2   "data": [
3     {
4       "guidEnv": "string", // идентификатор трассы
5       "guidNN": "string", // идентификатор агента
6       "rays": [ // 100 нормализованых расстояний до препятствий
7         0
8       ]
9     }
10  ]
11 }
```

Пример JSON ответа:

```
1  {
2   "moveSteps": [
3     {
4       "guidEnv": "string",// идентификатор трассы
5       "guidNN": "string",// идентификатор агента
6       "angle": 0,// нормализованный угол поворота
7       "speed": 0// нормализованная скорость
8     }
9   ]
10 }
```

Обновление награды

Команда ***Обновление награды***, приложение посылает запрос к конечной точке ***/updateReward***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON запроса:

```
1 {
2  "rewards": [
3    {
4      "guidNN": "string", // идентификатор агента
5      "reward": 0// награда
6    }
7  ]
8 }
```

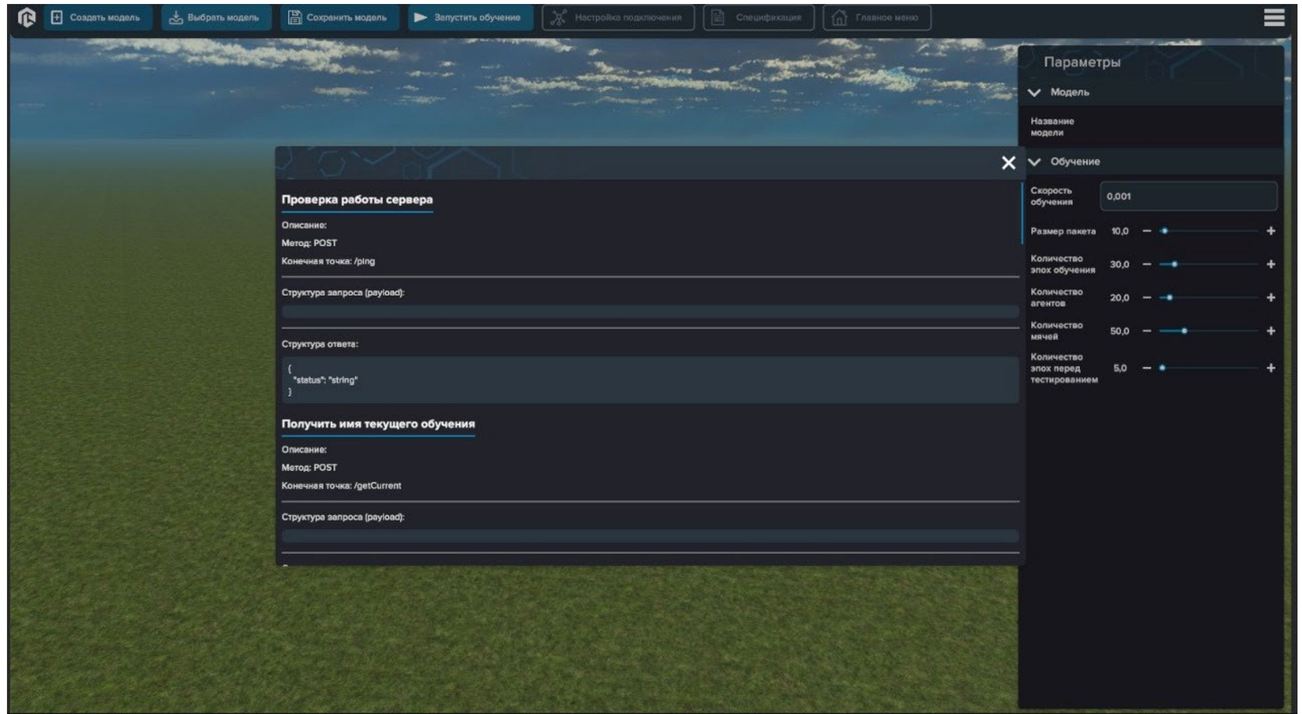
Пример JSON ответа:

```
1 {
2  "status": "string"//Окончание обновления награды
3 }
```

Спецификация API Попадание в обруч

В приложении вы можете ознакомиться со спецификацией отправки и получения запросов приложения от приложения к серверу.

При нажатии на вкладку спецификации откроется окно с основными запросами и ответами от приложения к серверу.



Спецификация

Проверка работы сервера

Команда ***Пинг сервера***, приложение посылает запрос к конечной точке ***/ping***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON ответа:

```
1 {
2   "status": "string"              //Ответ работает ли сервер
3 }
```

Получить имя текущего обучения

Команда ***Получить имя текущего обучения***, приложение посылает запрос к конечной точке ***/getCurrent***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON ответа:

```
1 {
2   "name": "string" // имя текущего обучения
3 }
```

Получить имена обучения

Команда ***Получить имена обучения***, приложение посылает запрос к конечной точке ***/getAll***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON ответа:

```
1 {
2   "names": [ // список всех обучений
3     {
4       "name": "string" // имя обучения
5     }
6   ]
7 }
```

Создать новое обучение

Команда ***Создать новое обучение***, приложение посылает запрос к конечной точке ***/create***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON запроса:

```
1 {
2   "name": "string" //имя нового обучения
3 }
```

Пример JSON ответа:

```
1 {
2   "status": "string" //Состояние создания нового обучения
3 }
```

Сохранить текущее обучение

Команда ***Сохранить текущее обучение***, приложение посылает запрос к конечной точке ***/save***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON ответа:

```
1 {
2   "status": "string" //Состояние сохранения обучения
3 }
```

Загрузить обучение

Команда ***Загрузить обучение***, приложение посылает запрос к конечной точке ***/load***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON запроса:

```
1 {
2   "name": "string" //имя нового обучения
3 }
```

Пример JSON ответа:

```
1 {
2   "status": "string" //Состояние загрузки обучения
3 }
```

Удалить обучение

Команда ***Удалить обучение***, приложение посылает запрос к конечной точке ***/delete***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON запроса:

```
1 {
2   "name": "string" //имя нового обучения
3 }
```

Пример JSON ответа:

```
1 {
2   "status": "string" //Состояние удаления обучения
3 }
```

Установить параметры обучения

Команда ***Установить параметры обучения***, приложение посылает запрос к конечной точке ***/startTrainingBasketBall***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON запроса:

```
1 {
2   "learning_rate": 0,// скорость обучения
3   "batch_size": 0,// скорость обучения
4 }
```

Пример JSON ответа:

```
1 {
2   "status": "string"//Состояние настройки обучения
3 }
```

Запустить обучение

Команда ***Запустить обучение***, приложение посылает запрос к конечной точке ***/resultShotsBasketBall***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON ответа:

```
1  {
2    "objects": [// список всех примеров попадания в кольцо
3      {
4        "x": "float"// позиция по x места броска
5        "z": "float"// позиция по z места броска
6        "zAngle": "float"// Нормализованный угол броска
7        "force": "float"// Нормализованная сила броска
8      }
9    ]
10 }
```

Пример JSON ответа:

```
1 Ответ:
2 {
3   "status": "string"//Завершение обучения
4 }
```

Произвести оценку

Команда ***Произвести оценку***, приложение посылает запрос к конечной точке ***/requestStatusTraining***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON ответа:

```
1 {
2     "accurancy": "float" //Точность
3     "loss": "float" // Значение функции потерь
4 }
```

Получить параметры броска мяча

Команда ***Получить параметры броска мяча***, приложение посылает запрос к конечной точке ***/ringShotsBasketBall***, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON запроса:

```
1 {
2   "objects": [// список всех примеров попадания в кольцо
3     {
4       "x": "float"// позиция по x места броска
5       "z": "float"// позиция по z места броска
6     }
7   ]
8 }
```

Пример JSON ответа:

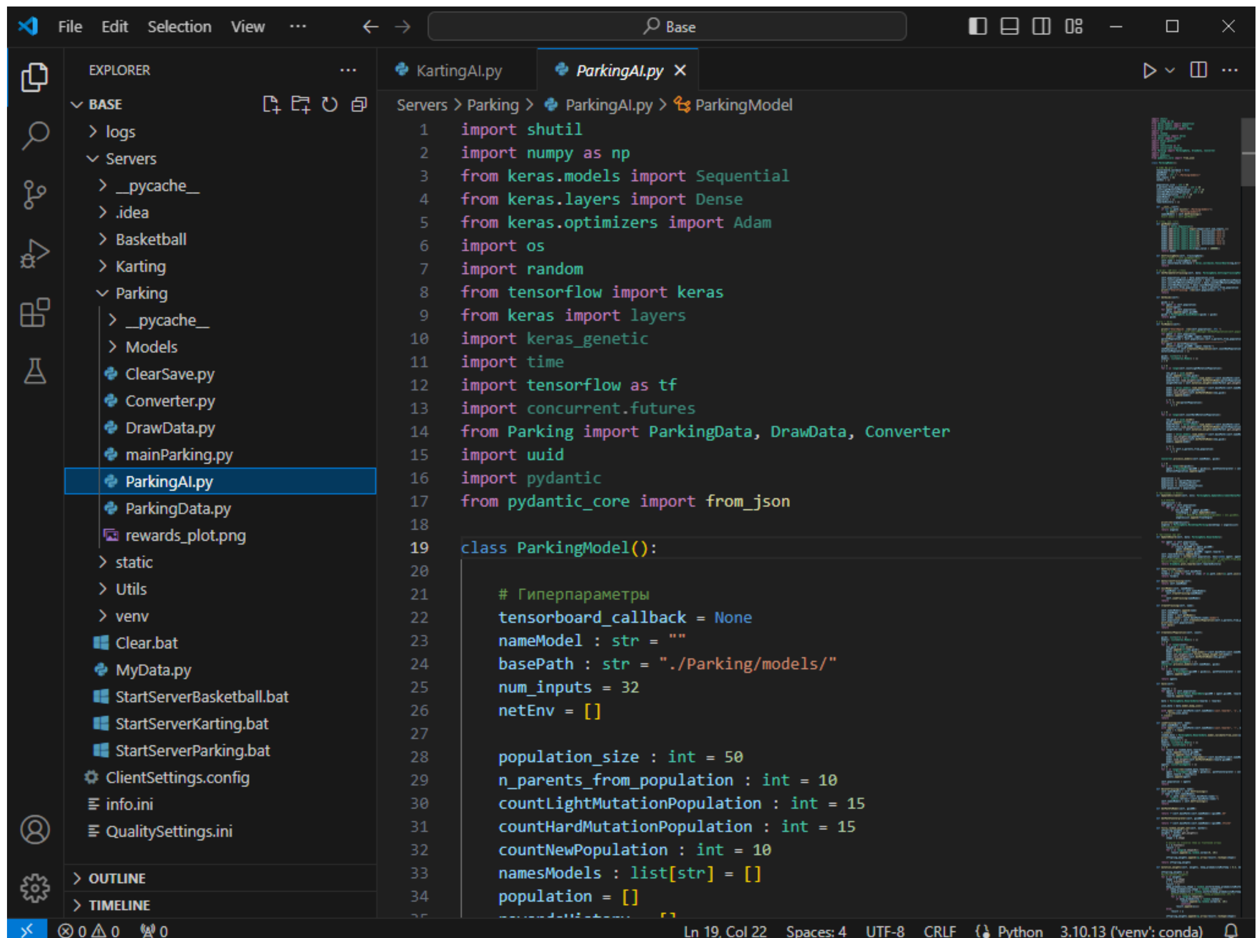
```
1  {
2    "objects": [ // список всех примеров попадания в кольцо
3      {
4        "x": "float"// позиция по x места броска
5        "z": "float"// позиция по z места броска
6        "zAngle": "float"// Нормализованный угол броска
7        "force": "float"// Нормализованная сила броска
8      }
9    ]
10 }
```

Описание лабораторной работы

В данном разделе описывается основной код лабораторной работы, все лабораторные работы написаны на языке программирования Python.

Описание сервера модуля Парковка

Рассмотрим код сервера модуля **Парковка**, а именно файл *ParkingAI.py*.



Сервер управления

init - инициализирует объект `ParkingModel`. Если папка `./Parking/models` не существует, создает ее. Получает список уже существующих тренировок моделей.

1	def __init__(self):
2	if not os.path.exists("./Parking/models"):
3	os.mkdir("./Parking/models")
4	namesModels = self.GetTrainings()
5	#self.model = self.getModel()

getModel - создает и возвращает модель нейросети с заданной архитектурой.

```
1  def getModel(self):
2        model = keras.Sequential()
3        model.add(keras.layers.Input(shape=(self.num_inputs,)))
4        model.add(keras.layers.Dense(32, activation='relu'))
5        model.add(keras.layers.Dense(32, activation='relu'))
6        model.add(keras.layers.Dense(64, activation='relu'))
7        model.add(keras.layers.Dense(64, activation='elu'))
8        model.add(keras.layers.Dense(32, activation='relu'))
9        model.add(keras.layers.Dense(16, activation='relu'))
10       model.add(keras.layers.Dense(2))
11       model.add(keras.layers.ReLU(max_value = 100000))
12       return model
```

SetTrainigData - устанавливает данные для тренировки, включая имя тренировки и коллбек для TensorBoard.

```
1  def SetTrainigData(self, trainingData):
2        #self.model = self.getModel(env)
3        self.name = trainingData.name
4        self.tensorboard_callback  =  keras.callbacks.TensorBoard(log_dir="./parking/"  +
5  trainingData.name +"/logs", update_freq = 50)
6        return
```

SetParametersTraining - устанавливает параметры тренировки из переданных данных `ParkingData.SettingsTrainingParking`.

```
1  def SetParametersTraining(self, data: ParkingData.SettingsTrainingParking):
2
3        self.population_size = data.population_size
4        self.countLightMutationPopulation = data.countLightMutationPopulation
5        self.countHardMutationPopulation = data.countHardMutationPopulation
6        self.countNewPopulation = data.countNewPopulation
7        self.n_parents_from_population = data.n_parents_from_population
8        print(f"StartTraining: {len(self.population):.5f} ")
9        return
```

GetGuids - возвращает список GUID всех агентов в текущей популяции.

```
1  def GetGuids(self):
2
3        guids = []
4        for agent in self.population:
5           print(agent)
6        for agent in self.population:
7           guids.append(agent.guidNN)
8        guids = ParkingData.GuidsModels(guids = guids)
```

9	return guids

FitModels - запускает эпоху тренировки, создавая новую популяцию агентов с мутациями и новыми агентами.

```
1  def FitModels(self):
2
3          print(f"StartEpoch: {len(self.population):.5f} ")
4          #self.population = self.searchManager.GetNewPopulation(self.population)
5          for agent in self.population:
6              print(f"{agent.guidNN} {agent.reward}")
7          parentPopulation = self.population[:self.n_parents_from_population]
8          print("______________________________________________")
9          for agent in parentPopulation:
10             print(f"{agent.guidNN} {agent.reward}")
11         newPopulation = self.CreateInitPopulation(self.countNewPopulation)
12         mutationPopulation = []
13
14         guids: list[str] = []
15         models: list[keras.Model] = []
16         i = 0
17
18         j = 0
19         for i in range(self.countLightMutationPopulation):
20
21             new_guid = uuid.uuid4()
22             guids.append(str(new_guid))
23             modelParent                                                    =
24 keras.models.load_model(f"{self.basePath}{self.nameModel}/model")
25
26 modelParent.load_weights(self.GetPathToModel(parentPopulation[j].guidNN))
27             weightsPerson = self.mutation_weights(modelParent.get_weights())
28
29             model                                                          =
30 keras.models.load_model(f"{self.basePath}{self.nameModel}/model")
31             model.set_weights(weightsPerson)
32             model.save_weights(self.GetPathToModel(new_guid))
33             models.append(model)
34
35             j += 1
36             if j >= len(parentPopulation):
37                 j = 0
38
39
40         j = 0
41         for i in range(self.countHardMutationPopulation):
42
43             new_guid = uuid.uuid4()
```

44	guids.append(str(new_guid))
45	modelParent =
46	keras.models.load_model(f"{self.basePath}{self.nameModel}/model")
47	
48	modelParent.load_weights(self.GetPathToModel(parentPopulation[j].guidNN))
49	weightsPerson = self.mutation_weights(modelParent.get_weights(), 0.85, 0.95)
50	
51	model =
52	keras.models.load_model(f"{self.basePath}{self.nameModel}/model")
53	model.set_weights(weightsPerson)
54	model.save_weights(self.GetPathToModel(new_guid))
55	models.append(model)
56	
57	j += 1
58	if j >= self.n_parents_from_population:
59	j = 0
60	
61	
62	Converter.processs_models(self.nameModel, guids)
63	
64	i = 0
65	for i in range(len(guids)):
66	agent = TrainAgent(guidNN = guids[i], pathToInterpreter =
67	self.GetPathToInterpreter(guids[i]))
68	mutationPopulation.append(agent)
69	
70	
71	population = []
72	population += (parentPopulation)
73	population += (newPopulation)
74	population += (mutationPopulation)
75	self.population = population

UpdateEnviromnet - обновляет окружение для каждого агента на основе переданных данных `ParkingData.UpdateEnviromentDatasParking`.

1	def UpdateEnviromnet(self, envs: ParkingData.UpdateEnviromentDatasParking):
2	
3	angelesList = []
4	for agent in self.population:
5	for env in envs.data:
6	if (env.guidNN == agent.guidNN):
7	floatAngle = agent.UpdateEnv(env)
8	#angel = MyData.MoveStepAngle(guidEnv = env.guidEnv, angle =
9	floatAngle)
10	angelesList.append(floatAngle)
11	
12	print(len(angelesList))

```
13        angeles = ParkingData.MoveStepsParking(moveSteps = angelesList)
14        #print(angeles)
15        return angeles
```

UpdateReward - обновляет награды для агентов на основе переданных данных `ParkingData.RewardsData` и сортирует популяцию по возрастанию наград.

```
1   def UpdateReward(self, data: ParkingData.RewardsData):
2
3         for agent in self.population:
4             for reward in data.rewards:
5                 if (reward.guidNN == agent.guidNN):
6                     agent.SetReward(reward)
7                     print(f"{agent.guidNN} {agent.reward}")
8         self.rewardsHistory.append(data)
9         self.population = sorted(self.population, key=lambda agent: agent.reward,
10  reverse=True)
11        #self.searchManager.Sort(self.population, self.n_parents_from_population)
12        #print(f"UpdateReward: {len(self.population):.5f} ")
13        return DrawData.plot_rewards(self.rewardsHistory)
```

GetTrainings - возвращает список всех существующих тренировок моделей.

```
1   def GetTrainings(self):
2         items = os.listdir(self.basePath)
3         folders = [item for item in items if os.path.isdir(os.path.join(self.basePath, item))]
4         return folders
```

GetCurrentTraining - возвращает текущее имя модели тренировки.

```
1   def GetCurrentTraining(self):
2         return self.nameModel
```

InitModel - инициализирует модель: либо создает новую тренировку, либо загружает существующую.

```
1   def InitModel(self, nameModel):
2         if nameModel not in self.namesModels:
3             self.CreateTraining(nameModel)
4         else:
5             self.LoadTraining(nameModel)
6         return
```

CreateTraining - создает новую тренировку с заданным именем и сохраняет начальную популяцию агентов.

```
1  def CreateTraining(self, name):
2
3          self.namesModels.append(name)
4          self.nameModel = name
5          self.model = self.getModel()
6          self.model.save(f"{self.basePath}{name}/model")
7          self.population = self.CreateInitPopulation(self.n_parents_from_population)
8          print(len(self.population))
9          self.Save()
10         return
```

CreateInitPopulation - создает начальную популяцию агентов с полностью случайными весами.

```
1  def CreateInitPopulation(self, count):
2
3          guids: list[str] = []
4          models: list[keras.Model] = []
5          i = 0
6          for i in range(count):
7              new_guid = uuid.uuid4()
8              guids.append(str(new_guid))
9              model                                                          =
10 keras.models.load_model(f"{self.basePath}{self.nameModel}/model")
11             model.set_weights(self.fully_random_weight_set(model))
12             model.save_weights(self.GetPathToModel(new_guid))
13             models.append(model)
14         agents: list[TrainAgent] = []
15         Converter.processs_models(self.nameModel, guids)
16         i = 0
17         for i in range(count):
18             agent    =    TrainAgent(guidNN    =    guids[i],    pathToInterpreter    =
19 self.GetPathToInterpreter(guids[i]))
20             agents.append(agent)
21
22         return agents
```

Save - сохраняет текущее состояние наград агентов в файл.

```
1  def Save(self):
2
3          rewards = []
4          for agent in self.population:
5
```

6	reward = ParkingData.RewardData(guidNN = agent.guidNN, reward =
7	agent.reward)
8	rewards.append(reward)
9	
10	data = ParkingData.RewardsData(rewards = rewards)
11	
12	json_data = data.model_dump_json()
13	
14	with open(f"{self.basePath}{self.nameModel}/Last.rewards", 'w', encoding='utf-
15	8') as f:
16	f.write(json_data)
17	f.close()
18	return

LoadTraining - загружает тренировку с заданным именем из файла и восстанавливает состояние агентов.

1	def LoadTraining(self, name):
2	self.nameModel = name
3	with open(f"{self.basePath}{self.nameModel}/Last.rewards", 'r', encoding='utf-
4	8') as f:
5	save = f.read()
6	f.close()
7	loaded_data = ParkingData.RewardsData.model_validate(from_json(save))
8	print(loaded_data)
9	guids: list[str] = []
10	models: list[keras.Model] = []
11	rewards: list[float] = []
12	i = 0
13	for reward in loaded_data.rewards:
14	guids.append(reward.guidNN)
15	rewards.append(reward.reward)
16	model =
17	keras.models.load_model(f"{self.basePath}{self.nameModel}/model")
18	model.load_weights(self.GetPathToModel(reward.guidNN))
19	models.append(model)
20	agents: list[TrainAgent] = []
21	i = 0
22	for i in range(len(loaded_data.rewards)):
23	agent = TrainAgent(guidNN = guids[i], pathToInterpreter =
24	self.GetPathToInterpreter(guids[i]))
25	agent.reward = rewards[i]
26	agents.append(agent)
27	
28	self.population = agents
29	return

DeleteTraining - удаляет тренировку с заданным именем и обновляет список существующих тренировок.

```
1  def DeleteTraining(self, name):
2          self.namesModels = self.GetTrainings()
3          if name in self.namesModels:
4              if os.path.isdir(f"{self.basePath}{name}"):
5                  shutil.rmtree(f"{self.basePath}{name}")
6          self.namesModels = self.GetTrainings()
7          return
```

GetPathToModel - возвращает путь к файлу модели для заданного GUID.

```
1  def GetPathToModel(self, guidNN):
2
3          return f"{self.basePath}{self.nameModel}/{guidNN}.h5"
```

GetPathToInterpreter - возвращает путь к файлу интерпретатора для заданного GUID.

```
1  def GetPathToInterpreter(self, guidNN):
2
3          return f"{self.basePath}{self.nameModel}/{guidNN}.tflite"
```

fully_random_weight_set - возвращает набор полностью случайных весов для модели.

```
1  def fully_random_weight_set(self, mother):
2          offspring_weights = []
3          weights = mother.get_weights()
4          for m in weights:
5              shape = m.shape
6
7              # easier to traverse them as flattened arrays
8              m = m.flatten()
9              result = []
10             for i in range(m.shape[0]):
11                 result.append(np.random.normal(0, 10))
12
13             offspring_weights.append(np.array(result).reshape(shape))
14
15         return offspring_weights
```

mutation_weights - возвращает мутированные веса для модели на основе заданных вероятностей.

```
1  def mutation_weights(self, weights, keep_probabilityMinTemp = 0.5,
2  keep_probabilityMaxTemp = 0.85):
3
4      offspring_weights = []
5      #print(parent.weights)
6      for w in weights:
7          shape = w.shape
8          w = w.flatten()
9          result = []
10         keep_probability_shape = random.uniform(keep_probabilityMinTemp,
11 keep_probabilityMaxTemp)
12         if (keep_probability_shape > random.random()):
13             keep_probability = random.uniform(keep_probabilityMinTemp,
14 keep_probabilityMaxTemp)
15             #print(f"keep_probability: {keep_probability:.5f} ")
16             for i in range(w.shape[0]):
17                 if keep_probability > random.random():
18                     result.append(np.random.normal(0, 10))
19                     continue
20                 result.append(w[i])
21         else:
22             result = w
23
24         offspring_weights.append(np.array(result).reshape(shape))
25     return offspring_weights
```

TrainAgent.__init__ - инициализирует объект `TrainAgent`, загружая интерпретатор модели TFLite из файла.

```
1  def __init__(
2      self,
3      guidNN,
4      pathToInterpreter
5    ):
6      #print("Create")
7      self.guidNN = guidNN
8
9      with open(pathToInterpreter, "rb") as f:
10         tflite_model = f.read()
11     interpreter = tf.lite.Interpreter(model_content=tflite_model)
12     interpreter.allocate_tensors()
13
14     self.interpreter = interpreter
15     self.input_details = self.interpreter.get_input_details()
16     self.output_details = self.interpreter.get_output_details()
```

TrainAgent.UpdateEnv - обновляет окружение агента на основе переданных данных `ParkingData.EnviromentDataParking` и возвращает новое действие агента.

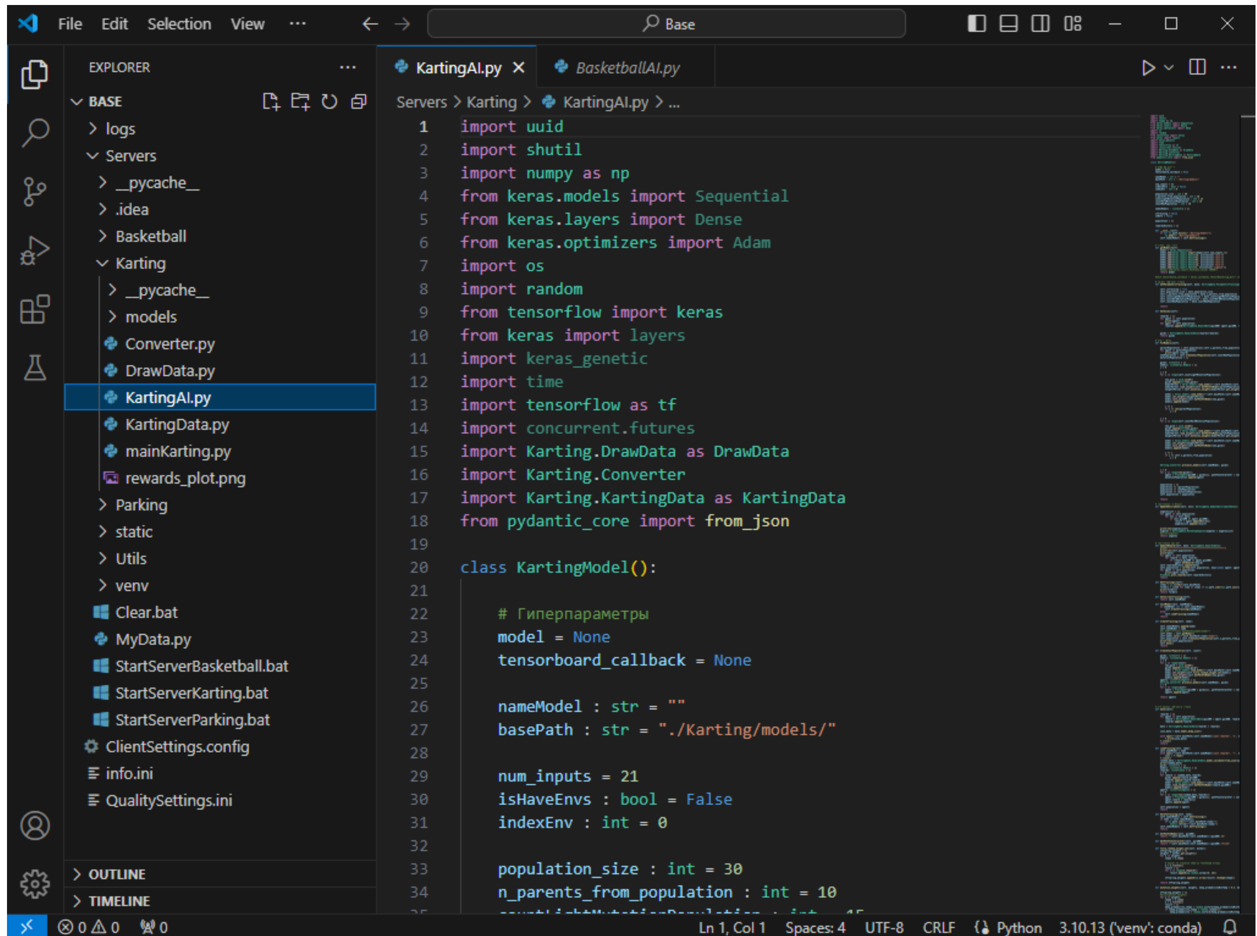
```
1  def UpdateEnv(self, env: ParkingData.EnviromentDataParking):
2
3          self.interpreter.set_tensor(self.input_details[0]['index'],
4  np.expand_dims(np.float32(env.rays), axis=0))
5          self.interpreter.invoke()
6          action = self.interpreter.get_tensor(self.output_details[0]['index'])
7          action = action / 100000
8          angel = ParkingData.MoveStepParking(guidEnv = env.guidEnv, guidNN =
9  env.guidNN, angle = round(action[0][0], 3), speed = round(action[0][1], 3))
10         return angel
```

TrainAgent.SetReward - Устанавливает награду для агента на основе переданных данных `ParkingData.RewardData`.

```
1  def SetReward(self, reward: ParkingData.RewardData):
2          self.reward = reward.reward
```

Описание сервера модуля Прохождение трассы

Рассмотрим код сервера модуля **Прохождение трассы**, а именно файл *KartingAI.py*.



Сервер управления

init - инициализирует объект `KartingModel`. Если директория для моделей отсутствует, создает её. Загружает список доступных тренировок.

1	def __init__(self):
2	if not os.path.exists("./Karting/models"):
3	os.mkdir("./Karting/models")
4	namesModels = self.GetTrainings()
5	#self.model = self.getModel()

getModel - создает и возвращает модель нейросети с заданной архитектурой.

1	def getModel(self):
2	model = keras.Sequential()
3	model.add(keras.layers.Input(shape=(self.num_inputs,)))
4	model.add(keras.layers.Dense(16, activation='relu'))
5	model.add(keras.layers.Dense(32, activation='relu'))
6	model.add(keras.layers.Dense(64, activation='relu'))
7	model.add(keras.layers.Dense(32, activation='relu'))
8	model.add(keras.layers.Dense(16, activation='relu'))
9	model.add(keras.layers.Dense(2, activation='sigmoid'))
10	#model.add(keras.layers.ReLU(max_value= 10000))

```
11        return model
```

SetParametersTraining - естанавливает параметры для обучения модели на основе данных из `KartingData.ParametersTraining`.

```
1   def SetParametersTraining(self, data: KartingData.ParametersTraining):
2
3         self.isTraining = True
4         self.population_size = data.population_size
5         self.n_parents_from_population = data.n_parents_from_population
6         self.countLightMutationPopulation = data.countLightMutationPopulation
7         self.countHardMutationPopulation = data.countHardMutationPopulation
8         self.countNewPopulation = data.countNewPopulation
9
10        return
```

GetGuids - возвращает список GUID всех агентов в текущей популяции.

```
1   def GetGuids(self):
2
3         rewards = []
4         for agent in self.population:
5            print(agent)
6         for agent in self.population:
7            rewards.append(KartingData.RewardData(guidNN= agent.guidNN, reward=
8   agent.reward))
9
10
11        guids = KartingData.RewardsData(rewards=rewards)
12        return guids
```

FitModels - запускает эпоху тренировки, создавая новую популяцию агентов с мутациями и новыми агентами.

```
1   def FitModels(self):
2
3         parentPopulation = self.population[:self.n_parents_from_population]
4         for agent in parentPopulation:
5            print(agent.reward)
6         newPopulation = self.CreateInitPopulation(self.countNewPopulation)
7         mutationPopulation = []
8
9         guids: list[str] = []
10        models: list[keras.Model] = []
11        i = 0
12
13        j = 0
```

```
14        for i in range(self.countLightMutationPopulation):
15
16            new_guid = uuid.uuid4()
17            guids.append(str(new_guid))
18            modelParent                                              =
19  keras.models.load_model(f"{self.basePath}{self.nameModel}/model")
20
21  modelParent.load_weights(self.GetPathToModel(parentPopulation[j].guidNN))
22            weightsPerson = self.mutation_weights(modelParent.get_weights())
23
24            model                                                    =
25  keras.models.load_model(f"{self.basePath}{self.nameModel}/model")
26            model.set_weights(weightsPerson)
27            model.save_weights(self.GetPathToModel(new_guid))
28            models.append(model)
29
30            j += 1
31            if j >= len(parentPopulation):
32                j = 0
33
34
35        j = 0
36        for i in range(self.countHardMutationPopulation):
37
38            new_guid = uuid.uuid4()
39            guids.append(str(new_guid))
40            modelParent                                              =
41  keras.models.load_model(f"{self.basePath}{self.nameModel}/model")
42
43  modelParent.load_weights(self.GetPathToModel(parentPopulation[j].guidNN))
44            weightsPerson = self.mutation_weights(modelParent.get_weights(), 0.85, 0.95)
45
46            model                                                    =
47  keras.models.load_model(f"{self.basePath}{self.nameModel}/model")
48            model.set_weights(weightsPerson)
49            model.save_weights(self.GetPathToModel(new_guid))
50            models.append(model)
51
52            j += 1
53            if j >= self.n_parents_from_population:
54                j = 0
55
56
57        Karting.Converter.processs_models(self.nameModel, guids)
58
59        i = 0
60        for i in range(len(guids)):
61            agent     =     TrainAgent(guidNN     =     guids[i],     pathToInterpreter     =
62  self.GetPathToInterpreter(guids[i]))
```

```
63            mutationPopulation.append(agent)
64
65
66        population = []
67        population += (parentPopulation)
68        population += (newPopulation)
69        population += (mutationPopulation)
70        self.population = population
71
72        return
```

UpdateEnviromnet - обновляет окружение для агентов на основе данных из `KartingData.UpdateEnviromentDatas`.

```
1   def UpdateEnviromnet(self, envs: KartingData.UpdateEnviromentDatas):
2
3       angelesList = []
4       for agent in self.population:
5           for env in envs.data:
6               if (env.guidNN == agent.guidNN):
7                   result = agent.UpdateEnv(env)
8                   angelesList.append(result)
9
10      print(len(angelesList))
11      angeles = KartingData.MoveStepAngeles(angeles = angelesList)
12      #print(angeles)
13      return angeles
```

UpdateReward - обновляет награды для агентов на основе данных из `KartingData.RewardsData`. Сортирует популяцию по наградам и сохраняет историю наград.

```
1   def UpdateReward(self, data: KartingData.RewardsData):
2       print("+++++++++++++++++++++++++++++++++++++++++++++++++++")
3       print(len(self.population))
4       print(data)
5       for agent in self.population:
6           for reward in data.rewards:
7               if (reward.guidNN == agent.guidNN):
8                   agent.SetReward(reward)
9       self.rewardsHistory.append(data)
10      self.population = sorted(self.population, key=lambda agent: agent.reward,
11  reverse=True)
12      for agent in self.population:
13          print(agent.reward)
14      DrawData.plot_rewards(self.rewardsHistory)
15      return
```

GetTrainings - возвращает список всех существующих тренировок моделей.

```
1  def GetTrainings(self):
2      items = os.listdir(self.basePath)
3      folders = [item for item in items if os.path.isdir(os.path.join(self.basePath, item))]
4      print(folders)
5      return folders
```

GetCurrentTraining - возвращает текущее имя модели тренировки.

```
1  def GetCurrentTraining(self):
2      return self.nameModel
```

InitModel - инициализирует модель: либо создает новую тренировку, либо загружает существующую.

```
1  def InitModel(self, nameModel):
2      if nameModel not in self.namesModels:
3          self.CreateTraining(nameModel)
4      else:
5          self.LoadTraining(nameModel)
6      return
```

CreateTraining - создает новую тренировку с заданным именем и сохраняет начальную популяцию агентов.

```
1  def CreateTraining(self, name):
2
3      self.namesModels.append(name)
4      self.nameModel = name
5      #os.mkdir(f"{self.basePath}{name}/model")
6      self.model = self.getModel()
7      self.model.save(f"{self.basePath}{name}/model")
8      self.population = self.CreateInitPopulation(self.n_parents_from_population)
9      print(len(self.population))
10     self.Save()
11     return
```

CreateInitPopulation - создает начальную популяцию агентов с полностью случайными весами.

```
1  def CreateInitPopulation(self, count):
2
3      guids: list[str] = []
```

4	models: list[keras.Model] = []
5	i = 0
6	for i in range(count):
7	new_guid = uuid.uuid4()
8	guids.append(str(new_guid))
9	model =
10	keras.models.load_model(f"{self.basePath}{self.nameModel}/model")
11	model.set_weights(self.fully_random_weight_set(model))
12	model.save_weights(self.GetPathToModel(new_guid))
13	models.append(model)
14	agents: list[TrainAgent] = []
15	Karting.Converter.processs_models(self.nameModel, guids)
16	i = 0
17	for i in range(count):
18	agent = TrainAgent(guidNN = guids[i], pathToInterpreter =
19	self.GetPathToInterpreter(guids[i]))
20	agents.append(agent)
21	
22	return agents

Save - сохраняет текущее состояние наград агентов в файл.

1	def Save(self):
2	
3	rewards = []
4	for agent in self.population:
5	reward = KartingData.RewardData(guidNN = agent.guidNN, reward =
6	agent.reward)
7	rewards.append(reward)
8	
9	data = KartingData.RewardsData(rewards = rewards)
10	
11	json_data = data.model_dump_json()
12	
13	with open(f"{self.basePath}{self.nameModel}/Last.rewards", 'w', encoding='utf-
14	8') as f:
15	f.write(json_data)
16	f.close()
17	return

LoadTraining - загружает тренировку с заданным именем из файла и восстанавливает состояние агентов.

1	def LoadTraining(self, name):
2	self.nameModel = name
3	with open(f"{self.basePath}{self.nameModel}/Last.rewards", 'r', encoding='utf-
4	8') as f:

```
5            save = f.read()
6        f.close()
7        loaded_data = KartingData.RewardsData.model_validate(from_json(save))
8        print(loaded_data)
9        guids: list[str] = []
10       models: list[keras.Model] = []
11       rewards: list[float] = []
12       i = 0
13       for reward in loaded_data.rewards:
14           guids.append(reward.guidNN)
15           rewards.append(reward.reward)
16           model                                                              =
17   keras.models.load_model(f"{self.basePath}{self.nameModel}/model")
18           model.load_weights(self.GetPathToModel(reward.guidNN))
19           models.append(model)
20       agents: list[TrainAgent] = []
21       i = 0
22       for i in range(len(loaded_data.rewards)):
23           agent    =    TrainAgent(guidNN    =    guids[i],    pathToInterpreter    =
24   self.GetPathToInterpreter(guids[i]))
25           agent.reward = rewards[i]
26           agents.append(agent)
27
28       self.population = agents
29       return
```

DeleteTraining - удаляет тренировку с заданным именем и обновляет список существующих тренировок.

```
1  def DeleteTraining(self, name):
2      self.namesModels = self.GetTrainings()
3      if name in self.namesModels:
4          if os.path.isdir(f"{self.basePath}{name}"):
5              shutil.rmtree(f"{self.basePath}{name}")
6      self.namesModels = self.GetTrainings()
7      return
```

GetPathToModel - возвращает путь к файлу модели для заданного GUID.

```
1  def GetPathToModel(self, guidNN):
2      return f"{self.basePath}{self.nameModel}/{guidNN}.h5"
```

GetPathToInterpreter - возвращает путь к файлу интерпретатора для заданного GUID.

```
1  def GetPathToInterpreter(self, guidNN):
2          return f"{self.basePath}{self.nameModel}/{guidNN}.tflite"
```

fully_random_weight_set - возвращает набор полностью случайных весов для модели.

```
1   def fully_random_weight_set(self, mother):
2           offspring_weights = []
3           weights = mother.get_weights()
4           for m in weights:
5               shape = m.shape
6   
7               # easier to traverse them as flattened arrays
8               m = m.flatten()
9               result = []
10              for i in range(m.shape[0]):
11                  result.append(np.random.normal(0, 10))
12  
13              offspring_weights.append(np.array(result).reshape(shape))
14  
15          return offspring_weights
```

mutation_weights - возвращает мутированные веса для модели на основе заданных вероятностей.

```
1   def     mutation_weights(self,     weights,     keep_probabilityMinTemp     =     0.5,
2   keep_probabilityMaxTemp = 0.85):
3   
4           offspring_weights = []
5           #print(parent.weights)
6           for w in weights:
7               shape = w.shape
8               w = w.flatten()
9               result = []
10              keep_probability_shape      =      random.uniform(keep_probabilityMinTemp,
11  keep_probabilityMaxTemp)
12              if (keep_probability_shape > random.random()):
13                  keep_probability        =        random.uniform(keep_probabilityMinTemp,
14  keep_probabilityMaxTemp)
15                  #print(f"keep_probability: {keep_probability:.5f} ")
16                  for i in range(w.shape[0]):
17                      if keep_probability > random.random():
18                          result.append(np.random.normal(0, 10))
19                          continue
20                      result.append(w[i])
21              else:
22                  result = w
23  
```

```
24            offspring_weights.append(np.array(result).reshape(shape))
25        return offspring_weights
```

TrainAgent.__init__ - инициализирует объект агента, загружает интерпретатор модели и получает детали ввода и вывода.

```
1   def __init__(
2           self,
3           guidNN,
4           pathToInterpreter
5       ):
6           #print("Create")
7           self.guidNN = guidNN
8
9           with open(pathToInterpreter, "rb") as f:
10              tflite_model = f.read()
11          interpreter = tf.lite.Interpreter(model_content=tflite_model)
12          interpreter.allocate_tensors()
13
14          self.interpreter = interpreter
15          self.input_details = self.interpreter.get_input_details()
16          self.output_details = self.interpreter.get_output_details()
```

TrainAgent.UpdateEnv - обновляет окружение для агента и возвращает действие на основе данных окружения.

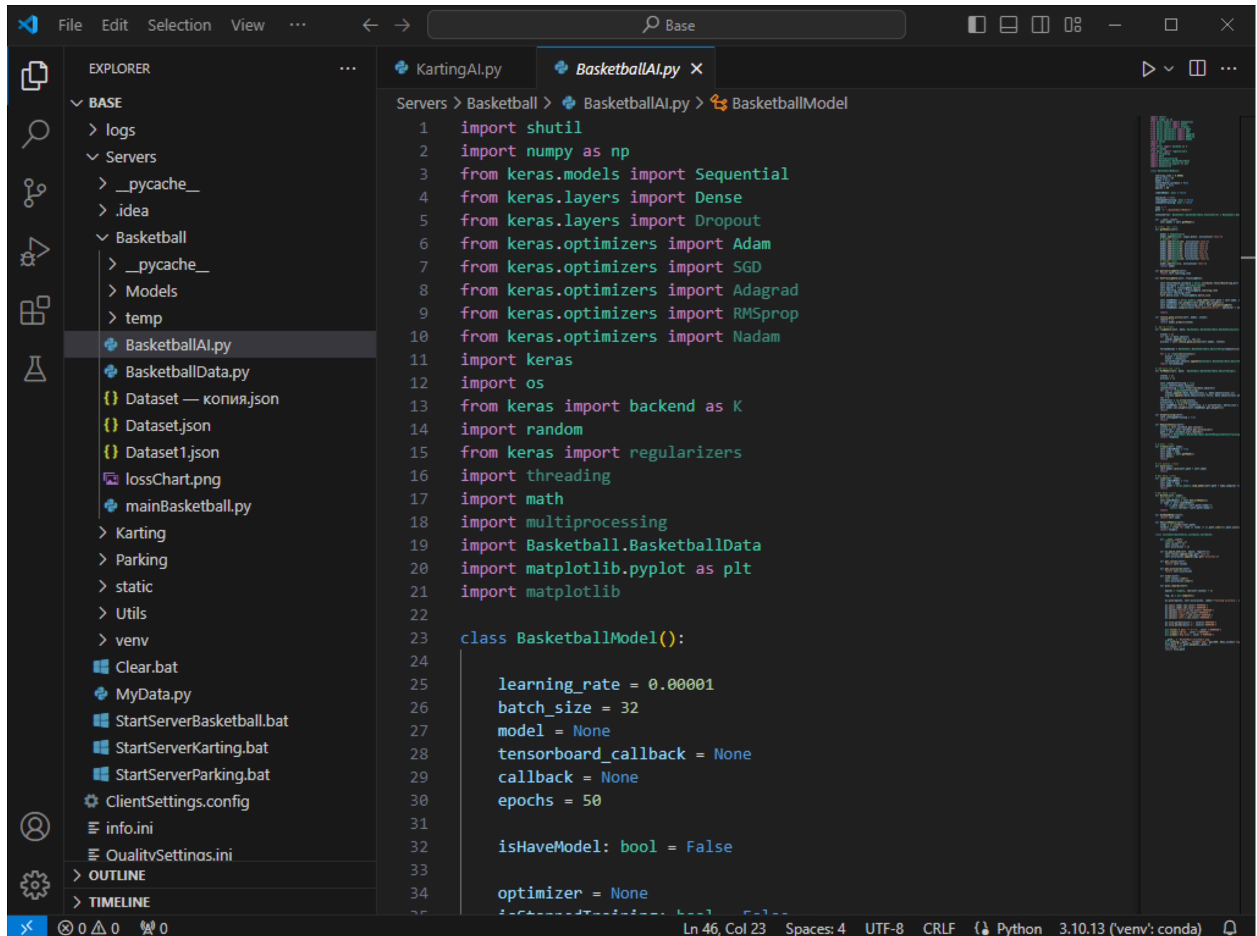
```
1   def UpdateEnv(self, env: KartingData.EnviromentData):
2
3           self.interpreter.set_tensor(self.input_details[0]['index'],
4   np.expand_dims(np.float32(env.rays), axis=0))
5           self.interpreter.invoke()
6           action = self.interpreter.get_tensor(self.output_details[0]['index'])
7           #action = action / 10000
8           angel  =  KartingData.MoveStepAngle(guidEnv  =  env.guidEnv,  guidNN  =
9   env.guidNN, angle = round(action[0][0], 3), speed = round(action[0][1], 3))
10          print(angel)
11          return angel
```

TrainAgent.SetReward - устанавливает награду для агента на основе данных из `KartingData.RewardData`.

```
1   def SetReward(self, reward: KartingData.RewardData):
2           self.reward = reward.reward
```

Описание сервера модуля Попадание в обруч

Рассмотрим код сервера модуля **Попадание в обруч**, а именно файл *BasketballAI.py*.



Сервер управления

__init__(self) - инициализирует экземпляр класса `BasketballModel` и вызывает метод `getModel()` для создания модели нейросети.

```
1  def __init__(self):
2      self.model = self.getModel()
```

getModel(self) - создает и возвращает последовательную модель нейросети (Sequential) с несколькими полносвязными слоями (Dense).

```
1  def getModel(self):
2
3      model = Sequential()
4      model.add(Dense(2, input_dim=2, activation='relu'))
5      # Hidden layers
6      model.add(Dense(128, activation='relu'))
7      model.add(Dense(32, activation='relu'))
8      model.add(Dense(32, activation='relu'))
9      model.add(Dense(64, activation='relu'))
10     model.add(Dense(64, activation='elu'))
11     model.add(Dense(32, activation='relu'))
12     model.add(Dense(16, activation='relu'))
13     model.add(Dense(128, activation='relu'))
```

```
14        # Output layer
15        model.add(Dense(2, activation='relu'))
16        return model
```

getLearningRate(self) - возвращает текущую скорость обучения (learning_rate).

```
1  def getLearningRate(self):
2        return self.learning_rate
```

SetTrainigData(self, trainingData) - устанавливает данные для обучения, инициализирует обратные вызовы (callbacks) и компилирует временную модель с оптимизатором Adam.

```
1   def SetTrainigData(self, trainingData):
2
3         self.tensorboard_callback = keras.callbacks.TensorBoard(log_dir= self.path +
4   self.name +"/logs", update_freq = 50)
5         self.callback = self.CustomCallback()
6         self.epochs = trainingData.epochs
7         self.learning_rate = trainingData.learning_rate
8         print(self.learning_rate)
9         self.batch_size = trainingData.batch_size
10
11        self.tempModel = keras.models.load_model(self.path + self.name, compile=
12  False)
13        self.tempModel.set_weights(self.model.get_weights())
14        self.optimizer = Adam(learning_rate = self.getLearningRate)
15        self.tempModel.compile(loss='mean_absolute_error', optimizer = self.optimizer,
16  metrics=['accuracy'])
17
18        return
```

choose_good_action(self, model, state) - делает предсказание состояния модели и возвращает результаты.

```
1  def choose_good_action(self, model, state):
2        reward = []
3        return model.predict(state)
```

ringShots(self, data: Basketball.BasketballData.BasketDistances) - выполняет предсказание силы и угла броска для каждого объекта на основе текущего состояния данных и возвращает результаты.

1	def ringShots(self, data: Basketball.BasketballData.BasketDistances):
2	
3	states = []
4	for val in data.objects:
5	states.append((val.x, val.z))
6	actions = self.choose_good_action(self.model, states)
7	
8	
9	forcesValues = Basketball.BasketballData.ResultsDrops(objects=[])
10	
11	for i in range(len(actions)):
12	action = actions[i]
13	state = states[i]
14	forcesValues.objects.append(Basketball.BasketballData.ResultDrop(force =
15	action[0], zAngle = action[1], x = state[0], z = state[1]))
16	return forcesValues

FitModel(self, data: Basketball.BasketballData.ResultsDrops) - обучает модель на основе предоставленных данных, обновляет веса модели и запускает обучение.

1	def FitModel(self, data: Basketball.BasketballData.ResultsDrops):
2	
3	states = []
4	actions = []
5	
6	self.isEnableTraining = True
7	random.shuffle(data.objects)
8	countTraining = math.floor(len(data.objects))
9	for val in range(countTraining):
10	states.append((data.objects[val].x, data.objects[val].z))
11	actions.append((data.objects[val].force, data.objects[val].zAngle))
12	val = 0
13	stateTrain = np.array(states)
14	actionTrain = np.array(actions)
15	self.tempModel.fit(x = stateTrain, y = actionTrain, batch_size = self.batch_size,
16	epochs=1 , verbose=1, callbacks = [self.callback])
17	self.model.set_weights(self.tempModel.get_weights())
18	return

StopTraining(self) - останавливает процесс обучения, устанавливая флаг \`isStoppedTraining\` в \`True\`.

1	def StopTraining(self):
2	self.isStoppedTraining = True
3	return

RequestStatus(self) - возвращает текущие значения потерь и точности модели на основе обратного вызова `CustomCallback`.

```
1  def RequestStatus(self):
2      losses = self.callback.get_losses()
3      accuracies = self.callback.get_accuracies()
4      path = self.callback.plot_rewards()
5      response = Basketball.BasketballData.BasketResponseStatusTraining(accurancy =
6  accuracies[len(accuracies) - 1], loss = losses[len(losses)- 1])
7      return response
```

Create(self, name) - создает новую модель с заданным именем, сохраняет ее и устанавливает флаг `isHaveModel` в `True`.

```
1  def Create(self, name):
2      self.isHaveModel = True
3      self.name = name
4      self.model = self.getModel()
5      self.Save()
6      return
```

Save(self) - сохраняет текущую модель в указанное местоположение на диске.

```
1  def Save(self):
2      self.model.save(self.path + self.name)
3      return
```

Load(self, name) - загружает модель по указанному имени из заданного пути.

```
1  def Load(self, name):
2      self.isHaveModel = True
3      self.name = name
4      self.model = keras.models.load_model(self.path + name,compile= False)
5      return
```

Delete(self, name) - удаляет модель с указанным именем, если она существует в списке моделей.

```
1  def Delete(self, name):
2      #if name != self.name:
3      self.namesModels = self.GetListModels()
4      if name in self.namesModels:
5          if os.path.isdir(f"{self.path}{name}"):
6              shutil.rmtree(f"{self.path}{name}")
```

```
7 |        return
```

GetNameModel(self) - возвращает имя текущей модели.

```
1 | def GetNameModel(self):
2 |        return self.name
```

GetListModels(self) - возвращает список всех доступных моделей в указанной директории.

```
1 | def GetListModels(self):
2 |        items = os.listdir(self.path)
3 |        folders = [item for item in items if os.path.isdir(os.path.join(self.path, item))]
4 |        return folders
```

Вложенный класс `CustomCallback`

__init__(self) - инициализирует экземпляр класса `CustomCallback`, создавая списки для хранения потерь и точностей.

```
1 | def __init__(self):
2 |            super().__init__()
3 |            self.losses = []
4 |            self.accuracies = []
```

on_epoch_end(self, epoch, logs=None) - вызывается в конце каждой эпохи обучения, добавляет значения потерь и точностей в соответствующие списки.

```
1 | def on_epoch_end(self, epoch, logs=None):
2 |            self.losses.append(logs.get('loss'))
3 |            self.accuracies.append(logs.get('accuracy'))
```

get_losses(self) - возвращает список потерь.

```
1 | def get_losses(self):
2 |            return self.losses
```

get_accuracies(self) - возвращает список точностей.

```
1 | def get_accuracies(self):
2 |            return self.accuracies
```

Clear(self) - очищает списки потерь и точностей.

```
1  def Clear(self):
2          self.losses.clear()
3          self.accuracies.clear()
```

plot_rewards(self) - строит график точности обучения по эпохам, сохраняет его в файл и возвращает путь к сохраненному изображению.

```
1  def plot_rewards(self):
2
3          epochs = range(1, len(self.losses) + 1)
4
5          fig, ax = plt.subplots()
6
7          ax.plot(epochs, self.accuracies, label='Training accuracy', color='#0095CD')
8
9          ax.xaxis.label.set_color('#8995AB')
10         ax.yaxis.label.set_color('#8995AB')
11         ax.spines['bottom'].set_color('#8995AB')
12         ax.spines['top'].set_color('#8995AB')
13         ax.spines['right'].set_color('#8995AB')
14         ax.spines['left'].set_color('#8995AB')
15
16         ax.tick_params(axis='x', colors='#8995AB')
17         ax.tick_params(axis='y', colors='#8995AB')
18
19         plt.title('График точности', color ='#8995AB')
20         plt.xlabel('Эпоха', color ='#8995AB')
21         plt.ylabel('Значение', color ='#8995AB')
22
23         __path__ = 'BasketBall/lossChart.png'
24         plt.savefig(__path__,      format='png',      dpi=600,      bbox_inches='tight',
25  transparent=True)
26         file_path = os.path.abspath(__path__)
27         plt.close('all')
28         return file_path
```

Sk Resident

ВИРТУАЛЬНЫЕ ЛАБОРАТОРИИ
ТРЕНАЖЕРЫ - СИМУЛЯТОРЫ
ИНТЕРАКТИВНЫЕ МАКЕТЫ
ЛАБОРАТОРНЫЕ СТЕНДЫ
ЦИФРОВЫЕ ДВОЙНИКИ
VR И AR КОМПЛЕКСЫ

ООО “ПРОГРАМЛАБ” INFO@PL-LLC.RU PLLLC.RU 8 (800) 550 89 72 +7 (499) 685 80 00