

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

**ОТЧЕТ**

по практической подготовке

в рамках реализации проекта «Цифровая кафедра»

программы профессиональной переподготовки

«Искусственный интеллект и машинное обучение»

|  |  |  |  |
| --- | --- | --- | --- |
| Обучающийся \_Аваков Карен Ашотовиич\_\_\_\_\_  (Ф.И.О.) | | \_\_\_\_\_\_\_ВПР24\_\_\_\_\_  (группа) | |
| Направление подготовки  \_\_09.03.04, Программная инженерия\_\_\_\_\_\_\_\_  (шифр, наименование) | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ИиВТ\_\_\_\_\_\_\_\_\_\_\_  (факультет) | | |
| Наименование базы практической подготовки \_\_\_ ООО «Зазекс»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (название предприятия) | | | |
| Период прохождения практики с15.04.2024 г. по 11.05.2024 г. | | | |
| Индивидуальное/групповое (*подчеркнуть нужное*) задание:  Эмулятор вождения для оценки качества управления автомобилем \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | |
|  | | | |
| Практические результаты:  Была разработана и обучена модель нейронной многослойной сети, которая с минимальной\_\_\_\_\_ погрешностью способна предсказывать количественный значения для управления машиной\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | |

|  |  |  |
| --- | --- | --- |
| Обучающийся | \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_  (дата сдачи отчета) (подпись) | Аваков К. А  (Ф.И.О.) |
|  |  |  |
| Руководитель практической подготовки от ДГТУ | \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (оценка) (подпись, дата) | доцент, Скляренко А. А  (должность, ФИО.) |

|  |
| --- |
| Ростов-на-Дону  2024 г. |

ДНЕВНИК ПРОХОЖДЕНИЯ ПРАКТИКИ

|  |  |  |  |
| --- | --- | --- | --- |
| Дата | База практики | Выполняемые работы | Оценка руководителя |
| 15.04.2024 | ООО «Зазекс» | Знакомство с предприятием, прохождение вводного инструктажа |  |
| 16.04.2024 | ООО «Зазекс» | Получение индивидуального (группового) задания и постановка задачи. |  |
| 17.04.2024-  22.04.2024 | ООО «Зазекс» | Аналитический обзор предметной области |  |
| 23.04.2024 – 6.05.2024 | ООО «Зазекс» | Программная реализация |  |
| 7.05.2024 – 10.05.2024 | ООО «Зазекс» | Подготовка и оформление отчета по практике |  |
| 11.05.2024 | ООО «Зазекс» | Сдача итогового отчета |  |

Руководитель практической подготовки от ДГТУ:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Доцент |  |  |  | А.А. Скляренко |
|  |  | *подпись, дата* |  |  |

**ОПИСАНИЕ РАБОТЫ ПРАКТИКАНТА**

в соответствии с учебным планом

программы профессиональной переподготовки

**«Искусственный интеллект и машинное обучение»**

1. **Описание предприятия и полученного индивидуального задания**

ООО «Zuzex» — это компания, специализирующаяся на разработке программного обеспечения, которая была основана в 2008 году. Они предоставляют широкий спектр услуг, включая веб-разработку, разработку мобильных приложений, DevOps и автоматизацию, обработку данных и искусственный интеллект, а также дизайн продуктов и разработку десктопных приложений. Компания имеет богатый опыт в создании масштабируемых и надежных корпоративных веб-платформ, мобильных приложений, интеллектуальных чат-ботов и медицинских решений​

Эмулятор вождения играет ключевую роль в оценке и улучшении качества

управления автомобилем. Он предоставляет безопасную и контролируемую среду для анализа и обучения водителей, имитируя реальные дорожные условия и сценарии.

Важные аспекты и преимущества эмулятора включают:

1. Безопасность на дорогах: Эмулятор вождения может помочь в обучении водителей, оценке их навыков и выявлении ошибок, что потенциально снижает количество дорожно-транспортных происшествий.
2. Образование и тренировки: Симуляторы могут быть использованы в автошколах для обучения новых водителей в безопасной и контролируемой среде.
3. Исследования и разработки: Анализ данных о поведении водителей может предоставить ценную информацию для исследований в области безопасности дорожного движения и разработки новых технологий для автомобилей.
4. Профессиональное использование: Такие симуляторы могут использоваться для оценки и тренировки профессиональных водителей, что особенно важно для транспортных компаний и служб доставки.

Однако, на пути к реализации проекта могут возникнуть следующие проблемы и препятствия:

1. Технические сложности: Разработка высокореалистичного симулятора требует значительных технических ресурсов и знаний, особенно в области графики, физики и обработки данных.
2. Интеграция данных: Сбор и анализ данных о поведении водителей требуют эффективной системы хранения и обработки данных, что может быть вызовом.
3. Пользовательский интерфейс: Создание интуитивного и удобного интерфейса для пользователей требует внимательного подхода к дизайну и тестированию.

Эмулятор вождения имеет значительный потенциал для повышения безопасности на дорогах и улучшения качества обучения водителей, что делает проект актуальным и востребованным в современном мире.

1. **Описание выполнения индивидуального задания**
   1. **Аналитический обзор предметной области**

Мы рассматриваем задачу регрессии, которая включает в себя прогнозирование количественных значений на основе входных данных. Методы регрессии, используемые в различных приложениях, включают линейную регрессию, полиномиальную регрессию, регрессию на основе деревьев решений, методы регуляризации (например, ридж и лассо-регрессию) и более сложные подходы, такие как градиентный бустинг и нейронные сети.

Глубокое обучение представляет собой подход к машинному обучению, использующий многослойные нейронные сети с множеством скрытых слоев для извлечения высокоуровневых признаков из данных. Этот метод показывает отличные результаты в задачах регрессии благодаря способности моделей глубокого обучения выявлять сложные нелинейные зависимости между входными признаками и целевыми переменными. Преимущества глубокого обучения включают автоматическое изучение признаков из данных, способность к обучению на больших объемах данных и возможность улучшения результатов с увеличением количества обучающих примеров.

В контексте вождения и оценки качества управления автомобилем, методы

регрессии могут использоваться для:

1. Прогнозирования времени в пути.
2. Оценки расхода топлива.
3. Предсказания поведения водителя на основе различных параметров, таких как скорость, ускорение и углы поворота.
4. Анализа данных о движении для выявления опасных маневров и улучшения системы помощи водителю.

Применение проекта в различных областях нашей жизни:

1. Безопасность дорожного движения: Помощь в обучении водителей и улучшении их навыков, что снижает количество аварий.
2. Автошколы: Использование симуляторов для оценки и повышения навыков учеников.
3. Транспортные компании: Оценка и тренировка профессиональных водителей, что улучшает эффективность и безопасность перевозок.
4. Исследования: Сбор данных для анализа поведения водителей и разработки новых технологий для автомобилей.
   1. **Описание моделей**

Наша модель регрессии построена с использованием нейронной сети с

несколькими слоями. Каждый слой имеет специфическую функцию и назначение:

Первый слой:

* Тип: Полносвязный слой (Dense)
* Размерность: 512 нейронов
* Функция активации: SELU (Scaled Exponential Linear Unit)
* Назначение: Этот слой принимает входные данные, которые предварительно обработаны и представлены в виде многочленных признаков. Большое количество нейронов позволяет модели извлечь сложные зависимости из входных данных.

Второй слой:

* Тип: Полносвязный слой (Dense)
* Размерность: 256 нейронов
* Функция активации: SELU
* Назначение: Сокращение количества нейронов помогает модели фокусироваться на наиболее значимых признаках

Третий слой:

* Тип: Полносвязный слой (Dense)
* Размерность: 64 нейрона
* Функция активации: SELU
* Назначение: Дальнейшее сокращение размерности и фокусировка на ключевых признаках

Четвёртый слой:

* Тип: Полносвязный слой (Dense)
* Размерность: 64 нейрона
* Функция активации: SELU
* Назначение: Окончательное сокращение размерности

Выходной слой:

* Тип: Полносвязный слой (Dense)
* Размерность: 2 нейрона
* Функция активации: Линейная (linear)
* Назначение: Этот слой предсказывает конечные количественные значения на основе входных данных, соответствуя задаче регрессии.

Графики MAE (Mean Absolute Error) и loss (ошибка) используются для

мониторинга качества обучения модели. MAE (рисунок 2.1) показывает среднюю абсолютную ошибку предсказаний, что помогает оценить точность модели. График функции потерь (рисунок 2.2) отображает, как хорошо модель обучается на данных, с течением времени. Также представлен график корреляции, отражающий степень зависимости данных. (рисунок 2.3)

Изображение выглядит как текст, снимок экрана, дисплей, Прямоугольник

Автоматически созданное описание

Рисунок 2.1 – Средняя абсолютная ошибка (MAE)­

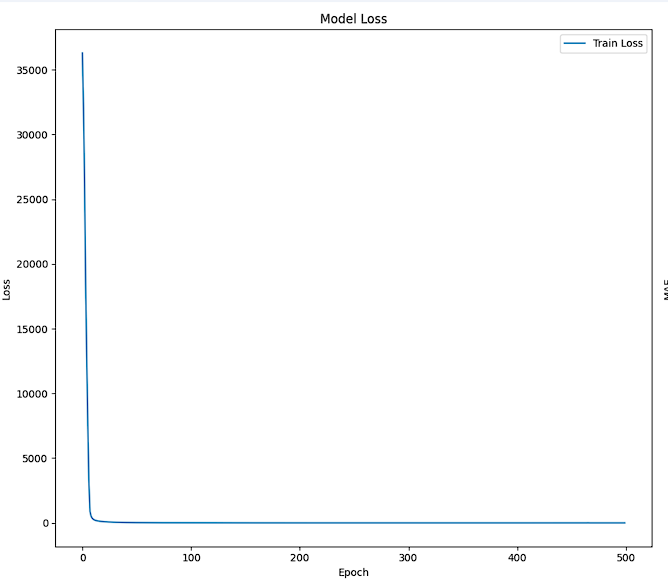


Рисунок 2.2 – График функции потерь (Loss)

Изображение выглядит как текст, снимок экрана, диаграмма, Прямоугольник

Автоматически созданное описание

Рисунок 2.3 – График корреляции

* 1. **Программная реализация**

Для реализации проекта использовался язык программирования Python, который предлагает богатый набор библиотек для машинного обучения и анализа данных, таких как TensorFlow и Keras. Эти библиотеки упрощают создание и обучение нейронных сетей. А библиотека PyGame позволяет реализовать визуальную составляющую проекта

Этапы реализации включали:

1. Сбор данных: Данные о поведении водителей собираются на основе информации получаемой после проезда трассы игроком. Эти данные включают скорость, углы поворота, время, координаты машины и направление движения.
2. Предварительная обработка данных: Данные нормализуются и преобразуются в многочленные признаки для улучшения обучения модели.
3. Создание модели: Использование библиотеки Keras для создания нейронной сети с несколькими слоями. Определение структуры модели и выбор функций активации.
4. Обучение модели: Обучение модели на подготовленных данных с использованием метода обратного распространения ошибки и оптимизатора Adam.
5. Оценка модели: Мониторинг метрик MAE и функции потерь для оценки производительности модели. Анализ результатов и настройка гиперпараметров при необходимости.
   1. **Демонстрация работы**

Демонстрация работы проекта включает в себя:

1. Сбор данных: во время поездки собираются данные о поведении водителя, такие как скорость, углы поворота, координаты, время, направление движения. (рисунок 2.4)

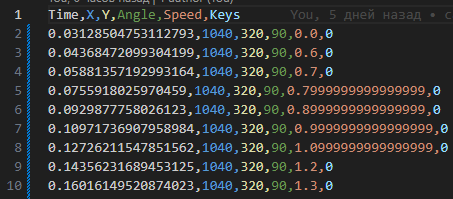


Рисунок 2.4

1. Анализ данных: Полученные данные анализируются с использованием обученной модели. Модель предсказывает 2 параметра: угол поворота и скорость машины.
2. Запуск симулятора: Пользователь запускает симулятор вождения, где имитируется реальная поездка с использованием заранее подготовленных маршрутов. Траектория ИИ (зеленая линия) и траектория игрока (красная линия) (рисунок 2.5)

Изображение выглядит как снимок экрана, текст, диаграмма, линия

Автоматически созданное описание

Изображение выглядит как снимок экрана, текст, диаграмма, линия

Автоматически созданное описаниеРисунок 2.5

1. Анализ результатов: Игрок может анализировать полученные результаты в виде графика траекторий (рисунок 2.6)

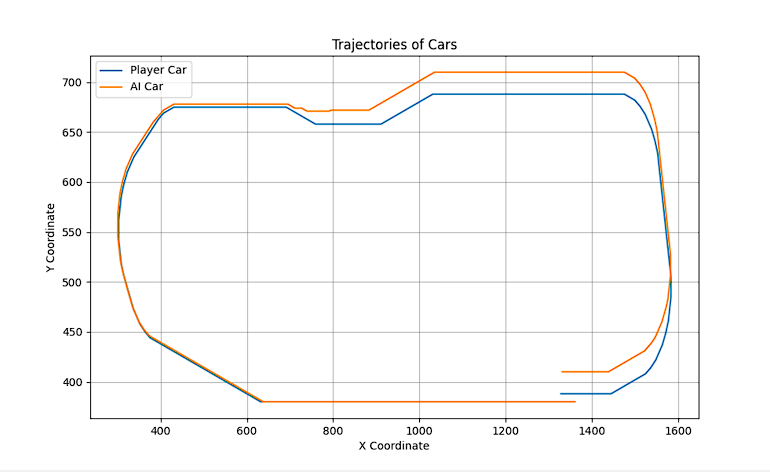


Рисунок 2.6

1. **Заключение**

В процессе выполнения задания была разработана и обучена модель нейронной сети для задачи регрессии, направленной на предсказание количественных значений в контексте управления автомобилем. Использование многослойной нейронной сети с функцией активации SELU позволило достичь высокой точности предсказаний.

Основные выводы:

* Разработанная модель нейронной сети с многослойной структурой успешно справляется с задачей регрессии, показывая хорошие результаты на различных наборах данных, собранных с симуляторов вождения.
* Применение функции активации SELU и последовательное уменьшение числа нейронов в слоях позволило модели эффективно извлекать значимые признаки и обобщать данные.
* Графики MAE и функции потерь демонстрируют, что модель хорошо обучается и демонстрирует стабильные результаты в процессе обучения.
* Полученные результаты подтверждают применимость моделей глубокого обучения для анализа и улучшения навыков вождения, что может быть полезно для безопасности дорожного движения и обучения водителей.

Эти выводы подчеркивают значимость использования нейронных сетей в области автомобильного транспорта для анализа и прогнозирования различных параметров, способствуя повышению безопасности и эффективности управления автомобилями.

1. **Перечень использованных информационных ресурсов**

1. Keras regression examples on GitHub [Электронный ресурс], URL: <https://github.com/keras-team/keras-io> (дата обращения: 05.06.2024г.)

2. Глубокое обучение (Deep Learning): обзор [Электронный ресурс], URL: <https://habr.com/ru/companies/otus/articles/459785/> (дата обращения: 07.06.2024г.)

*Приложение*

*к отчету**по практической подготовке*

**Листинг программы**

main-menu.py:

import pygame

import sys

from level import level\_menu, draw\_text

from game import run\_game

pygame.init()

pygame.display.set\_caption("Project Cars")

WHITE = (255, 255, 255)

BLACK = (0, 0, 0)

GRAY = (169, 169, 169)

DARK\_GRAY = (105, 105, 105)

numberOfLvl = 0

drive = False

def main\_menu(width, height):

    global drive, numberOfLvl

    WINDOW\_SIZE = (width, height)

    screen = pygame.display.set\_mode(WINDOW\_SIZE)

    image = pygame.image.load("main\_menu.jpg")

    image = pygame.transform.scale(image, (width, height))

    image\_rect = image.get\_rect()

    screen\_rect = screen.get\_rect()

    image\_rect.center = screen\_rect.center

    font = pygame.font.SysFont("DejaVuSans", 38, bold=True)

    button\_width = 300

    button\_height = 70

    border\_radius = 20

    while True:

        screen.fill(BLACK)

        screen.blit(image, image\_rect)

        mouse\_pos = pygame.mouse.get\_pos()

        button\_surface = pygame.Surface((button\_width, button\_height), pygame.SRCALPHA)

        button\_color = (\*BLACK, 150)  # Черный цвет с прозрачностью 150

        button\_1 = pygame.Rect(width // 2 - 150, height // 2 - 150, button\_width, button\_height)

        button\_2 = pygame.Rect(width // 2 - 150, height // 2 - 20, button\_width, button\_height)

        button\_3 = pygame.Rect(width // 2 - 150, height // 2 + 140, button\_width, button\_height)

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                sys.exit()

            elif event.type == pygame.VIDEORESIZE:  # Обработка изменения размера окна

                width, height = event.w, event.h

                screen = pygame.display.set\_mode((width, height))

                image = pygame.transform.scale(image, (width, height))

                image\_rect = image.get\_rect(center=screen\_rect.center)

            elif event.type == pygame.MOUSEBUTTONDOWN:

                if event.button == 1:  # Левая кнопка мыши

                    if button\_1.collidepoint(event.pos):

                        level = level\_menu(width, height)

                        if level is not None:

                            drive = False

                            numberOfLvl = level

                            with open('numberOfLvl\_value.txt', 'w') as f:

                                f.write(str(numberOfLvl))

                            with open('drive\_value.txt', 'w') as f:

                                f.write(str(drive))

                            run\_game(width, height, numberOfLvl)

                    elif button\_2.collidepoint(event.pos):

                        level = level\_menu(width, height)

                        if level is not None:

                            drive = True

                            numberOfLvl = level

                            with open('numberOfLvl\_value.txt', 'w') as f:

                                f.write(str(numberOfLvl))

                            with open('drive\_value.txt', 'w') as f:

                                f.write(str(drive))

                            run\_game(width, height, numberOfLvl)

                    elif button\_3.collidepoint(event.pos):

                        pygame.quit()

                        sys.exit()

        if button\_1.collidepoint(mouse\_pos):

            pygame.draw.rect(button\_surface, (\*DARK\_GRAY, 200), (0, 0, button\_width, button\_height), border\_radius=border\_radius)

        else:

            pygame.draw.rect(button\_surface, button\_color, (0, 0, button\_width, button\_height), border\_radius=border\_radius)

        screen.blit(button\_surface, button\_1.topleft)

        draw\_text('Начать обучение', font, WHITE, screen, width // 2, height // 2 - 115)

        if button\_2.collidepoint(mouse\_pos):

            pygame.draw.rect(button\_surface, (\*DARK\_GRAY, 200), (0, 0, button\_width, button\_height), border\_radius=border\_radius)

        else:

            pygame.draw.rect(button\_surface, button\_color, (0, 0, button\_width, button\_height), border\_radius=border\_radius)

        screen.blit(button\_surface, button\_2.topleft)

        draw\_text('Начать вождение', font, WHITE, screen, width // 2, height // 2 + 15)

        if button\_3.collidepoint(mouse\_pos):

            pygame.draw.rect(button\_surface, (\*DARK\_GRAY, 200), (0, 0, button\_width, button\_height), border\_radius=border\_radius)

        else:

            pygame.draw.rect(button\_surface, button\_color, (0, 0, button\_width, button\_height), border\_radius=border\_radius)

        screen.blit(button\_surface, button\_3.topleft)

        draw\_text('Выход', font, WHITE, screen, width // 2, height // 2 + 175)

        info\_font = pygame.font.SysFont("DejaVuSans", 24)

        info\_text = " - Created by Avakov Karen & Nikita Plotnikov. All rights reserved © - "

        info\_x = width // 2

        info\_y = height - 50

        draw\_text(info\_text, info\_font, WHITE, screen, info\_x, info\_y)

        pygame.display.update()

main\_menu(1920, 1080)

level.py:

import pygame

import sys

from pygame.locals import QUIT, VIDEORESIZE, MOUSEBUTTONDOWN

pygame.display.set\_caption("level choice")

WHITE = (255, 255, 255)

BLACK = (0, 0, 0)

GRAY = (169, 169, 169)

DARK\_GRAY = (105, 105, 105)

def draw\_text(text, font, color, surface, x, y):

    text\_obj = font.render(text, True, color)

    text\_rect = text\_obj.get\_rect()

    text\_rect.center = (x, y)

    surface.blit(text\_obj, text\_rect)

def level\_menu(width, height):

    WINDOW\_SIZE = (width, height)

    screen = pygame.display.set\_mode(WINDOW\_SIZE)

    title\_font = pygame.font.SysFont("DejaVuSans", 78, bold=True)

    background = pygame.image.load("level.jpg").convert()

    background = pygame.transform.scale(background, WINDOW\_SIZE)

    button\_images = [

        pygame.image.load("level1\_menu.jpg").convert\_alpha(),

        pygame.image.load("level2\_menu.jpg").convert\_alpha(),

        pygame.image.load("level3\_menu.jpg").convert\_alpha()

    ]

    button\_width = 400

    button\_height = 400

    button\_images = [pygame.transform.scale(img, (button\_width, button\_height)) for img in button\_images]

    highlight\_overlay = pygame.Surface((button\_width, button\_height), pygame.SRCALPHA)

    highlight\_overlay.fill((255, 255, 255, 50))

    main\_menu\_button\_width = 300

    main\_menu\_button\_height = 70

    main\_menu\_button\_rect = pygame.Rect(width // 2 - main\_menu\_button\_width // 2, height - 200, main\_menu\_button\_width, main\_menu\_button\_height)

    border\_radius = 20

    while True:

        screen.blit(background, (0, 0))  # Отображение фонового изображения

        draw\_text('Выберите уровень', title\_font, WHITE, screen, width // 2, height // 4)

        mouse\_pos = pygame.mouse.get\_pos()

        for i in range(3):

            button\_x = width // 4 \* (i + 1) - button\_width // 2

            button\_y = height // 2 - button\_height // 2

            button\_surface = pygame.Surface((button\_width, button\_height), pygame.SRCALPHA)

            button\_surface.blit(button\_images[i], (0, 0))

            button\_rect = pygame.Rect(button\_x, button\_y, button\_width, button\_height)

            if button\_rect.collidepoint(mouse\_pos):

                button\_surface.blit(highlight\_overlay, (0, 0))

                for event in pygame.event.get():

                    if event.type == MOUSEBUTTONDOWN and event.button == 1:

                        return i + 1

            screen.blit(button\_surface, (button\_x, button\_y))

        button\_surface = pygame.Surface((main\_menu\_button\_width, main\_menu\_button\_height), pygame.SRCALPHA)

        button\_color = (\*BLACK, 150)  # Черный цвет с прозрачностью 150

        if main\_menu\_button\_rect.collidepoint(mouse\_pos):

            pygame.draw.rect(button\_surface, (\*DARK\_GRAY, 200), (0, 0, main\_menu\_button\_width, main\_menu\_button\_height), border\_radius=border\_radius)

        else:

            pygame.draw.rect(button\_surface, button\_color, (0, 0, main\_menu\_button\_width, main\_menu\_button\_height), border\_radius=border\_radius)

        screen.blit(button\_surface, main\_menu\_button\_rect.topleft)

        draw\_text("Главное меню", pygame.font.SysFont("DejaVuSans", 28), WHITE, screen, main\_menu\_button\_rect.centerx, main\_menu\_button\_rect.centery)

        for event in pygame.event.get():

            if event.type == QUIT:

                pygame.quit()

                sys.exit()

            elif event.type == VIDEORESIZE:

                width = event.w

                height = event.h

                screen = pygame.display.set\_mode((width, height))

                background = pygame.transform.scale(background, (width, height))

            elif event.type == MOUSEBUTTONDOWN:

                if main\_menu\_button\_rect.collidepoint(event.pos):

                    return None

                for i in range(3):

                    button\_x = width // 4 \* (i + 1) - button\_width // 2

                    button\_y = height // 2 - button\_height // 2

                    button\_rect = pygame.Rect(button\_x, button\_y, button\_width, button\_height)

                    if button\_rect.collidepoint(event.pos):

                        return i + 1

        info\_font = pygame.font.SysFont("DejaVuSans", 24)

        info\_text = " - Created by Avakov Karen & Nikita Plotnikov. All rights reserved © - "

        info\_x = width // 2

        info\_y = height - 50

        draw\_text(info\_text, info\_font, WHITE, screen, info\_x, info\_y)

        pygame.display.update()

Car.py:

import pygame

import math

WHITE = (255, 255, 255)

BLACK = (0, 0, 0)

RED = (255, 0, 0)

GREEN = (0, 255, 0)

UP = 'up'

DOWN = 'down'

LEFT = 'left'

RIGHT = 'right'

class Car(pygame.sprite.Sprite):

    def \_\_init\_\_(self, x, y):

        super().\_\_init\_\_()

        self.original\_image = pygame.image.load('MyCar.png')

        self.original\_image = pygame.transform.scale(self.original\_image, (40, 80))

        self.image = self.original\_image.copy()

        self.rect = self.image.get\_rect(center=(x, y))

        self.angle = 0

        self.speed = 0

        self.maxForwardSpeed = 4

        self.forwardAcceleration = 0.1

        self.maxBackSpeed = -2

        self.backAcceleration = 0.05

        self.min\_turn\_speed = 1

        self.positions = []

        self.drawPositions = []

        self.angle\_ = []

        self.speed\_ = []

        self.keys\_recorded = []

        self.trail\_color = RED

        self.trail = []

        self.initial\_x = x

        self.initial\_y = y

    def update(self, keys):

        pressed\_key = None

        if keys[pygame.K\_UP]:

            self.speed += self.forwardAcceleration

            self.speed = min(self.speed, self.maxForwardSpeed)

            pressed\_key = 0  # UP

        elif keys[pygame.K\_DOWN]:

            self.speed -= self.backAcceleration

            self.speed = max(self.speed, self.maxBackSpeed)

            pressed\_key = 1  # DOWN

        else:

            if self.speed > 0:

                self.speed -= 4 \* self.backAcceleration

            if self.speed < 0:

                self.speed += 2 \* self.forwardAcceleration

        if abs(self.speed) > self.min\_turn\_speed:

            if keys[pygame.K\_LEFT]:

                self.angle += 2.5

                pressed\_key = 2  # LEFT

            elif keys[pygame.K\_RIGHT]:

                self.angle -= 2.5

                pressed\_key = 3  # RIGHT

        self.image = pygame.transform.rotate(self.original\_image, self.angle)

        self.rect = self.image.get\_rect(center=self.rect.center)

        dx = math.cos(math.radians(self.angle + 90)) \* self.speed

        dy = math.sin(math.radians(-self.angle - 90)) \* self.speed

        self.rect.x += dx

        self.rect.y += dy

        if self.rect.left < 0:

            self.rect.left = 0

        elif self.rect.right > 1920:

            self.rect.right = 1920

        if self.rect.top < 0:

            self.rect.top = 0

        elif self.rect.bottom > 1080:

            self.rect.bottom = 1080

        new\_position = (self.rect.centerx, self.rect.centery)

        self.drawPositions.append(new\_position)

        if not self.positions or self.positions[-1] != new\_position and self.speed != 0:

            self.positions.append(new\_position)

            self.angle\_.append(self.angle)

            self.speed\_.append(self.speed)

            self.trail.append(new\_position)

        if pressed\_key is not None:

            self.keys\_recorded.append(pressed\_key)

    def draw\_trail(self, screen):

        if len(self.trail) > 1:

            pygame.draw.lines(screen, self.trail\_color, False, self.trail, 2)

    def get\_keys\_recorded(self):

        return self.keys\_recorded

    def get\_angle\_recorded(self):

        return self.angle\_

    def get\_positions(self):

        return self.positions

    def get\_speed\_recorded(self):

        return self.speed\_

    def get\_positions(self):

        return self.drawPositions

    def reset\_keys\_recorded(self):

        self.keys\_recorded = []

    def reset\_positions(self):

        self.positions = []

AICar.py:

import pygame

from pygame.locals import \*

import math

import os

os.environ['TF\_ENABLE\_ONEDNN\_OPTS'] = '0'

import tensorflow as tf

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from tensorflow.keras.models import Sequential, load\_model

from tensorflow.keras.layers import Dense

from tensorflow.keras.optimizers import Adam

from sklearn.preprocessing import StandardScaler, PolynomialFeatures

import logging

WHITE = (255, 255, 255)

BLACK = (0, 0, 0)

RED = (255, 0, 0)

GREEN = (0, 255, 0)

UP = 'up'

DOWN = 'down'

LEFT = 'left'

RIGHT = 'right'

class AICar(pygame.sprite.Sprite):

    def \_\_init\_\_(self, x, y):

        super().\_\_init\_\_()

        self.original\_image = pygame.image.load('AICar.png')

        self.original\_image = pygame.transform.scale(self.original\_image, (40, 80))

        self.image = self.original\_image.copy()

        self.rect = self.image.get\_rect(center=(x, y))

        self.angle = 0

        self.speed = 0

        self.maxForwardSpeed = 4

        self.maxBackSpeed = -2

        self.forwardAcceleration = 0.1

        self.backAcceleration = 0.05

        self.min\_turn\_speed = 1

        self.positions = []

        self.drawPositions = []

        self.final\_model = None

        self.frames\_since\_last\_update = 0

        self.actions\_probabilities = []  # Массив для хранения предсказанных действий

        self.current\_action\_index = 0

        self.logger = logging.getLogger(\_\_name\_\_)

        handler = logging.FileHandler('car\_model.log', encoding='utf-8')

        handler.setFormatter(logging.Formatter('%(asctime)s - %(levelname)s - %(message)s'))

        self.logger.addHandler(handler)

        self.logger.setLevel(logging.INFO)

        self.trail\_color = GREEN

        self.trail = []

        self.X\_normalized = []

        self.model = None

        self.X\_poly = None

        self.history = None

        self.visible = False

    def create\_model(self):

        width = 1920

        height = 1080

        WINDOW\_SIZE = (width, height)

        screen = pygame.display.set\_mode(WINDOW\_SIZE)

        font = pygame.font.Font(None, 46)

        text = font.render("Обучение модели в процессе, ожидайте...", True, WHITE)

        text\_rect = text.get\_rect(center=(width // 2, height // 2))

        screen.blit(text, text\_rect)

        pygame.display.flip()

        df = pd.read\_csv('car\_data.csv')

        X = df[['Time', 'X', 'Y', 'Keys']].values

        y = df[['Speed', 'Angle']].values

        scaler = StandardScaler()

        X\_normalized = scaler.fit\_transform(X)

        poly = PolynomialFeatures(degree=5)

        self.X\_poly = poly.fit\_transform(X\_normalized)

        X\_train, y\_train = self.X\_poly, y

        model = Sequential()

        model.add(Dense(512, input\_dim=self.X\_poly.shape[1], activation='selu'))

        model.add(Dense(256, activation='selu'))

        model.add(Dense(128, activation='selu'))

        model.add(Dense(64, activation='selu'))

        model.add(Dense(2, activation='linear'))

        model.compile(optimizer=Adam(learning\_rate=0.0001), loss='mean\_squared\_error', metrics=['mae'])

        early\_stopping = tf.keras.callbacks.EarlyStopping(

            monitor='loss', patience=500, restore\_best\_weights=True, verbose=2

        )

        history = model.fit(X\_train, y\_train, epochs=500, batch\_size=32, callbacks=[early\_stopping])

        self.model = model

        self.history = history.history

        plt.figure(figsize=(18, 8))

        plt.subplot(1, 2, 1)

        plt.plot(self.history['loss'], label='Train Loss')

        plt.title('Model Loss')

        plt.xlabel('Epoch')

        plt.ylabel('Loss')

        plt.legend()

        plt.subplot(1, 2, 2)

        plt.plot(self.history['mae'], label='Train MAE')

        plt.title('Mean Absolute Error')

        plt.xlabel('Epoch')

        plt.ylabel('MAE')

        plt.legend()

        plt.tight\_layout()

        plt.show()

        model.save("car\_race.h5")

        model\_json = model.to\_json()

        with open("model\_architecture.json", "w") as json\_file:

            json\_file.write(model\_json)

        plt.figure(figsize=(10, 8))

        corr = df[['Time', 'X', 'Y', 'Keys']].corr()

        sns.heatmap(corr, annot=True, cmap='hot', vmin=-1, vmax=1)

        plt.title('Correlation Matrix of Input Features')

        plt.show()

        model.summary()

        return model

    def predict\_actions(self, drive):

        if drive == "False":

            self.actions\_probabilities = self.model.predict(self.X\_poly)

        else:

            with open('numberOfLvl\_value.txt', 'r') as f:

                numberOfLvl = int(f.read().strip())

            if numberOfLvl == 1:

                self.model = load\_model('car\_race\_1.h5')

                df = pd.read\_csv('car\_data\_1.csv')

                X = df[['Time', 'X', 'Y', 'Keys']].values

                scaler = StandardScaler()

                X\_normalized = scaler.fit\_transform(X)

                poly = PolynomialFeatures(degree=5)

                self.X\_poly = poly.fit\_transform(X\_normalized)

                self.actions\_probabilities = self.model.predict(self.X\_poly)

            if numberOfLvl == 2:

                self.model = load\_model('car\_race\_2.h5')

                df = pd.read\_csv('car\_data\_2.csv')

                X = df[['Time', 'X', 'Y', 'Keys']].values

                scaler = StandardScaler()

                X\_normalized = scaler.fit\_transform(X)

                poly = PolynomialFeatures(degree=5)

                self.X\_poly = poly.fit\_transform(X\_normalized)

                self.actions\_probabilities = self.model.predict(self.X\_poly)

            if numberOfLvl == 3:

                self.model = load\_model('car\_race\_3.h5')

                df = pd.read\_csv('car\_data\_3.csv')

                X = df[['Time', 'X', 'Y', 'Keys']].values

                scaler = StandardScaler()

                X\_normalized = scaler.fit\_transform(X)

                poly = PolynomialFeatures(degree=5)

                self.X\_poly = poly.fit\_transform(X\_normalized)

                self.actions\_probabilities = self.model.predict(self.X\_poly)

    def update(self, screen):

        if self.current\_action\_index < len(self.actions\_probabilities):

            predicted\_action = self.actions\_probabilities[self.current\_action\_index]

            predicted\_speed = predicted\_action[0]

            predicted\_angle = predicted\_action[1]

            self.speed = predicted\_speed

            self.angle = predicted\_angle

            self.current\_action\_index += 1

        else:

            if self.speed > 0:

                self.speed -= 4 \* self.backAcceleration

            elif self.speed < 0:

                self.speed += 2 \* self.forwardAcceleration

            return

        self.image = pygame.transform.rotate(self.original\_image, self.angle)

        self.rect = self.image.get\_rect(center=self.rect.center)

        dx = math.cos(math.radians(self.angle + 90)) \* self.speed

        dy = math.sin(math.radians(-self.angle - 90)) \* self.speed

        new\_position = (self.rect.centerx, self.rect.centery)

        self.drawPositions = (new\_position)

        self.trail.append(new\_position)

        new\_x = self.rect.x + dx

        new\_y = self.rect.y + dy

        if 0 <= new\_x <= screen.get\_width() - self.rect.width:

            self.rect.x = new\_x

        if 0 <= new\_y <= screen.get\_height() - self.rect.height:

            self.rect.y = new\_y

    def draw\_trail(self, screen):

        if len(self.trail) > 1:

            pygame.draw.lines(screen, self.trail\_color, False, self.trail, 2)

    def hide(self):

        self.visible = False

    def show(self):

        self.visible = True

    def get\_positions(self):

        return self.drawPositions

game.py:

from AiCar import AICar

from Car import Car

import pygame

import sys

import time

import pandas as pd

import matplotlib.pyplot as plt

pygame.display.set\_caption("game")

WHITE = (255, 255, 255)

BLACK = (0, 0, 0)

RED = (255, 0, 0)

GREEN = (0, 255, 0)

UP = 'up'

DOWN = 'down'

LEFT = 'left'

RIGHT = 'right'

class Level:

    def \_\_init\_\_(self, width, height, numberOfLvl):

        self.width = width

        self.height = height

        self.screen = pygame.display.set\_mode((self.width, self.height))

        self.track\_image, self.track\_x, self.track\_y = self.load\_track\_image(numberOfLvl)

        self.track\_mask = pygame.mask.from\_surface(self.track\_image)

        self.checkpoints = self.create\_checkpoints(10, 91, numberOfLvl)

        self.current\_checkpoint\_index = 0

        self.visited\_checkpoints = 0

    def load\_track\_image(self, numberOfLvl):

        track\_image = pygame.image.load(f"level{numberOfLvl}.png").convert\_alpha()

        track\_width = track\_image.get\_width()

        track\_height = track\_image.get\_height()

        screen\_center\_x = self.screen.get\_width() // 2

        screen\_center\_y = self.screen.get\_height() // 2

        track\_x = screen\_center\_x - track\_width // 2

        track\_y = screen\_center\_y - track\_height // 2

        return track\_image, track\_x, track\_y

    def create\_track\_mask(self):

        track\_mask = pygame.mask.from\_surface(self.track\_image[0])

        return track\_mask

    def check\_checkpoints(self, car\_rect):

        if not self.checkpoints:

            return None

        if self.current\_checkpoint\_index >= len(self.checkpoints):

            return None

        current\_checkpoint = self.checkpoints[self.current\_checkpoint\_index]

        if car\_rect.colliderect(current\_checkpoint):

            self.current\_checkpoint\_index = (self.current\_checkpoint\_index + 1) % len(self.checkpoints)

            self.visited\_checkpoints += 1

            return current\_checkpoint

        return None

    def check\_all\_checkpoints\_visited(self):

        return self.visited\_checkpoints == len(self.checkpoints)

    def create\_checkpoints(self, checkpoint\_width, checkpoint\_height, numberOfLvl):

        if numberOfLvl == 1:

            checkpoints = [

                pygame.Rect(self.width // 2 - 350, self.height // 2 - 218, checkpoint\_width, 120),

                pygame.Rect(self.width // 2 - 350, self.height // 2 + 98, checkpoint\_width, 120),

                pygame.Rect(self.width // 2 + 320, self.height // 2 + 98, checkpoint\_width, 120),

                pygame.Rect(self.width // 2 + 320, self.height // 2 - 218, checkpoint\_width, 120),

            ]

        elif numberOfLvl == 2:

            checkpoints = [

                pygame.Rect(500, 268, checkpoint\_width, checkpoint\_height),

                pygame.Rect(self.width // 2 - 400, self.height // 2 - 33, checkpoint\_width, checkpoint\_height),

                pygame.Rect(self.width // 2 + 400, self.height // 2 + 220, checkpoint\_width, checkpoint\_height),

                pygame.Rect(self.width // 2 + 450, self.height // 2 - 20, checkpoint\_width, checkpoint\_height),

                pygame.Rect(self.width // 2 + 65, 270, checkpoint\_width, checkpoint\_height),

            ]

        elif numberOfLvl == 3:

            checkpoints = [

                pygame.Rect(self.width // 2 + 361, self.height // 2 - 320, 88, 10),

                pygame.Rect(self.width // 2 - 97, self.height // 2 - 150, 88, 10),

                pygame.Rect(self.width // 2 - 300, self.height // 2 + 261, 10, 66),

                pygame.Rect(self.width // 2 + 187, self.height // 2 + 370, 87, 10),

                pygame.Rect(self.width // 2 + 361, self.height // 2 - 90, 88, 10),

            ]

        return checkpoints

    def draw(self, screen):

        screen.blit(self.track\_image, (self.track\_x, self.track\_y))

        if self.checkpoints:

            current\_checkpoint = self.checkpoints[self.current\_checkpoint\_index]

            pygame.draw.rect(screen, RED, current\_checkpoint)

    # def draw(self, screen):

    #     screen.blit(self.track\_image, (self.track\_x, self.track\_y))

    #     for checkpoint in self.checkpoints:

    #         pygame.draw.rect(screen, RED, checkpoint)

    def is\_on\_track(self, car\_surface, car\_rect):

        car\_mask = pygame.mask.from\_surface(car\_surface)

        offset\_x = car\_rect.x - self.track\_x

        offset\_y = car\_rect.y - self.track\_y

        return self.track\_mask.overlap\_area(car\_mask, (offset\_x, offset\_y)) > 0

    def draw\_button(self, text, rect):

        pygame.draw.rect(self.screen, BLACK, rect)

        font = pygame.font.Font(None, 36)

        text\_surf = font.render(text, True, WHITE)

        text\_rect = text\_surf.get\_rect(center=rect.center)

        self.screen.blit(text\_surf, text\_rect)

    def open\_main\_menu(self):

        import main\_menu

        main\_menu.main\_menu(1920, 1080)

    def start\_ai\_race(self):

        self.ai\_mode = True

def CheckAiInitialPos(ai\_initialX, ai\_initialY, ai\_NewX, ai\_NewY):

    if ai\_initialX != ai\_NewX or ai\_initialY != ai\_NewY:

        return True

    return False

def plot\_trajectories(car\_trail, ai\_trail):

    car\_x, car\_y = zip(\*car\_trail)

    ai\_x, ai\_y = zip(\*ai\_trail)

    plt.figure(figsize=(10, 6))

    plt.plot(car\_x, car\_y, label="Player Car")

    plt.plot(ai\_x, ai\_y, label="AI Car")

    plt.xlabel("X Coordinate")

    plt.ylabel("Y Coordinate")

    plt.title("Trajectories of Cars")

    plt.legend()

    plt.grid(True)

    plt.show()

def printRestart(numberOfLvl, screen, width,height):

    font = pygame.font.Font(None, 36)

    restart\_text = font.render("Нажмите 'Рестарт'!", True, RED)

    if numberOfLvl == 1 or numberOfLvl == 2:

        restart\_rect = restart\_text.get\_rect(center=(width // 2, height // 2))

    elif numberOfLvl == 3:

        restart\_rect = restart\_text.get\_rect(center=(width // 2 + 150, height // 2))

    screen.blit(restart\_text, restart\_rect)

def run\_game(width, height, numberOfLvl):

    pygame.init()

    WINDOW\_SIZE = (width, height)

    screen = pygame.display.set\_mode(WINDOW\_SIZE)

    clock = pygame.time.Clock()

    def initialize\_cars():

        positions = {

            1: (width // 2 + 400, height // 2 - 160),

            2: (width // 2 + 80, height // 2 - 220),

            3: (width // 2 + 410, height // 2)

        }

        car = Car(\*positions[numberOfLvl])

        ai\_car = AICar(\*positions[numberOfLvl])

        if numberOfLvl in [1, 2]:

            car.angle += 90

            ai\_car.angle += 90

        return car, ai\_car

    car, ai\_car = initialize\_cars()

    ai\_initialX, ai\_initialY = ai\_car.rect.x, ai\_car.rect.y

    off\_track\_counter\_ai, off\_track\_counter = 0, 0

    startAi, player\_started, show\_results = False, False, False

    penalty\_player, penalty\_ai = 0.0, 0.0

    game\_time\_player, game\_time\_ai = 0.0, 0.0

    last\_off\_track, lastAi\_off\_track = False, False

    ai\_disqualified, player\_disqualified = False, False

    game\_start\_time, ai\_last\_movement\_time = time.time(), time.time()

    ai\_mode, ai\_control, player\_control, pressed, ai\_training\_completed = False, False, True, True, False

    startAiTime, total\_time\_text\_ai, last\_checkpoint, lastAi\_checkpoint = None, None, None, None

    background = Level(width, height, numberOfLvl)

    font = pygame.font.Font(None, 36)

    button1, button2, button3 = pygame.Rect(5, 170, 200, 50), pygame.Rect(5, 230, 200, 50), pygame.Rect(5, 290, 200, 50)

    game\_time\_history = []

    def reset\_game():

        screen.fill(WHITE)

        run\_game(width, height, numberOfLvl)

    player\_started = False

    show\_results = False

    ai\_last\_position\_x = ai\_car.rect.x

    ai\_last\_position\_y = ai\_car.rect.y

    while True:

        all\_sprites = pygame.sprite.Group()

        all\_sprites.add(car)

        if ai\_car.visible:

            all\_sprites.add(ai\_car)

        screen.fill(WHITE)

        current\_time = time.time()

        elapsed\_time = current\_time - game\_start\_time

        game\_time\_history.append(elapsed\_time)

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                sys.exit()

            elif event.type == pygame.MOUSEBUTTONDOWN:

                mouse\_pos = event.pos

                if button1.collidepoint(mouse\_pos):

                    background.open\_main\_menu()

                elif button2.collidepoint(mouse\_pos) and pressed:

                    if background.current\_checkpoint\_index < len(background.checkpoints):

                        player\_disqualified = True

                    background.current\_checkpoint\_index = 0

                    background.create\_checkpoints(10, 91, numberOfLvl)

                    ai\_car.show()

                    ai\_mode = True

                    ai\_control = True

                    startAi = True

                    keys\_list = car.get\_keys\_recorded()

                    positions\_list = car.get\_positions()

                    angles\_list = car.get\_angle\_recorded()

                    speeds\_list = car.get\_speed\_recorded()

                    data = {'Time': [], 'X': [], 'Y': [], 'Angle': [], 'Speed': [], 'Keys': []}

                    max\_length = max(len(keys\_list), len(positions\_list), len(angles\_list), len(speeds\_list), len(game\_time\_history))

                    for i in range(max\_length):

                        data['X'].append(positions\_list[i][0] if i < len(positions\_list) else car.rect.x)

                        data['Y'].append(positions\_list[i][1] if i < len(positions\_list) else car.rect.y)

                        data['Angle'].append(angles\_list[i] if i < len(angles\_list) else car.angle)

                        data['Speed'].append(speeds\_list[i] if i < len(speeds\_list) else 0)

                        data['Keys'].append(keys\_list[i] if i < len(keys\_list) else 4)

                        data['Time'].append(game\_time\_history[i] if i < len(game\_time\_history) else data['Time'].append(time.time() - game\_start\_time))

                    df = pd.DataFrame(data)

                    df.to\_csv('car\_data.csv', index=False)

                    with open('drive\_value.txt', 'r') as f:

                        drive = f.read().strip()

                        if drive == 'True':

                            ai\_car.predict\_actions(drive)

                            ai\_training\_completed = True

                            startAiTime = time.time()

                        else:

                            ai\_car.model = ai\_car.create\_model()

                            ai\_car.predict\_actions(drive)

                            ai\_training\_completed = True

                            if ai\_training\_completed:

                                startAiTime = time.time()

                        background.create\_checkpoints(10, 91, numberOfLvl)

                        ai\_car.show()

                        pressed = False

                elif button3.collidepoint(mouse\_pos):

                    reset\_game()

        keys = pygame.key.get\_pressed()

        if not ai\_mode and player\_control:

            ai\_car.hide()

            car.update(keys)

            if keys[pygame.K\_LEFT] or keys[pygame.K\_RIGHT] or keys[pygame.K\_UP] or keys[pygame.K\_DOWN]:

                if player\_started == False:

                    player\_started = True

                    game\_start\_time = time.time()

        if ai\_mode and ai\_control:

            ai\_car.update(screen)

        if player\_control:

            checkpoint = background.check\_checkpoints(car.rect)

            if checkpoint and checkpoint != last\_checkpoint:

                last\_checkpoint = checkpoint

                if checkpoint == background.checkpoints[-1]:

                    player\_control = False

                    ai\_control = True

                    startAi = True

                    startAiTime = time.time()

        if CheckAiInitialPos(ai\_initialX, ai\_initialY, ai\_car.rect.x, ai\_car.rect.y):

            if startAi and startAiTime and ai\_mode and ai\_control:

                if ai\_control:

                    game\_time\_ai = time.time() - startAiTime + penalty\_ai

                    if ai\_car.rect.x == ai\_last\_position\_x and ai\_car.rect.y == ai\_last\_position\_y:

                        if current\_time - ai\_last\_movement\_time >= 2:

                            ai\_disqualified = True

                            printRestart(numberOfLvl, screen, width, height)

                    else:

                        ai\_last\_position\_x = ai\_car.rect.x

                        ai\_last\_position\_y = ai\_car.rect.y

                        ai\_last\_movement\_time = current\_time

                    if ai\_disqualified:

                        total\_time\_text\_ai = font.render(f"Время ИИ: дисквалифицирован", True, BLACK)

                    else:

                        total\_time\_text\_ai = font.render(f"Время ИИ: {game\_time\_ai:.2f} сек", True, BLACK)

                    screen.blit(total\_time\_text\_ai, (width - 400, 90))

                    off\_track\_textAi = font.render(f"Выездов за трассу: {off\_track\_counter\_ai}", True, BLACK)

                    screen.blit(off\_track\_textAi, (width - 400, 50))

                aiCheckpoints = background.check\_checkpoints(ai\_car.rect)

                if aiCheckpoints and aiCheckpoints != lastAi\_checkpoint:

                    lastAi\_checkpoint = aiCheckpoints

                    if background.checkpoints and lastAi\_checkpoint == background.checkpoints[-1]:

                        ai\_control = False

                        printRestart(numberOfLvl, screen, width, height)

                if background.is\_on\_track(ai\_car.image, ai\_car.rect):

                    if lastAi\_off\_track:

                        lastAi\_off\_track = False

                    ai\_text = font.render("ИИ на трассе", True, BLACK)

                else:

                    if not lastAi\_off\_track:

                        penalty\_ai += 5.0

                        off\_track\_counter\_ai += 1

                        lastAi\_off\_track = True

                    ai\_text = font.render("ИИ вне трассы - Штраф!", True, BLACK)

                screen.blit(ai\_text, ((width - ai\_text.get\_width()) // 2, 120))

                background.draw(screen)

            else:

                total\_time\_text\_ai = font.render(f"Время ИИ: {game\_time\_ai:.2f} сек", True, BLACK)

                screen.blit(total\_time\_text\_ai, (width - 300, 90))

                off\_track\_textAi = font.render(f"Выездов за трассу: {off\_track\_counter\_ai}", True, BLACK)

                screen.blit(off\_track\_textAi, (width - 300, 50))

                printRestart(numberOfLvl, screen, width, height)

        else:

            total\_time\_text\_ai = font.render(f"Время ИИ: {game\_time\_ai:.2f} сек", True, BLACK)

            total\_time\_text\_ai = font.render(f"Время ИИ: {game\_time\_ai:.2f} сек", True, BLACK)

            screen.blit(total\_time\_text\_ai, (width - 300, 90))

            off\_track\_textAi = font.render(f"Выездов за трассу: {off\_track\_counter\_ai}", True, BLACK)

            screen.blit(off\_track\_textAi, (width - 300, 50))

        if player\_control or ai\_control:

            if background.is\_on\_track(car.image, car.rect):

                if last\_off\_track:

                    last\_off\_track = False

                text = font.render(f"Машина на трассе", True, BLACK)

            else:

                if not last\_off\_track:

                    off\_track\_counter += 1

                    last\_off\_track = True

                    penalty\_player += 5.0

                text = font.render(f"Машина вне трассы - Штраф!", True, BLACK)

        else:

            background.draw(screen)

            all\_sprites.draw(screen)

            text\_finish = font.render("Финиш!", True, BLACK)

            text\_finish\_rect = text\_finish.get\_rect(center=((width // 2, 40)))

            screen.blit(text\_finish, text\_finish\_rect.topleft)

            printRestart(numberOfLvl, screen, width, height)

        background.draw(screen)

        all\_sprites.draw(screen)

        car.draw\_trail(screen)

        ai\_car.draw\_trail(screen)

        if text:

            screen.blit(text, ((width - text.get\_width()) // 2, 50))

        if player\_started and player\_control:

            game\_time\_player = time.time() - game\_start\_time + penalty\_player

            if player\_disqualified:

                total\_time\_text\_player = font.render(f"Время игрока: дисквалифицирован", True, BLACK)

            else:

                total\_time\_text\_player = font.render(f"Время игрока: {game\_time\_player:.2f} сек", True, BLACK)

            screen.blit(total\_time\_text\_player, (10, 90))

        elif player\_started:

            total\_time\_text\_player = font.render(f"Время игрока: {game\_time\_player:.2f} сек", True, BLACK)

            screen.blit(total\_time\_text\_player, (10, 90))

        else:

            total\_time\_text\_player = font.render(f"Время игрока: {game\_time\_player:.2f} сек", True, BLACK)

            screen.blit(total\_time\_text\_player, (10, 90))

        off\_track\_text = font.render(f"Выездов за трассу: {off\_track\_counter}", True, BLACK)

        screen.blit(off\_track\_text, (10, 50))

        background.draw\_button("Главное меню", button1)

        background.draw\_button("Проезд ИИ", button2)

        background.draw\_button("Рестарт", button3)

        fps = font.render(f"FPS: {int(clock.get\_fps())}", True, BLACK)

        fps\_rect = fps.get\_rect(center=(width // 2, height - 20))

        screen.blit(fps, fps\_rect)

        final\_results = None

        if not ai\_control and not player\_control:

            final\_results = [

            ["", "Время", "Выезды за трассу"],

            ["Игрок", f"{game\_time\_player:.2f} сек", off\_track\_counter],

            ["ИИ", f"{game\_time\_ai:.2f} сек", off\_track\_counter\_ai]

        ]

        cell\_width = 250

        cell\_height = 60

        row\_height = cell\_height

        if final\_results is not None:

            if numberOfLvl == 1:

                x\_offset = 500

                y\_offset = 100

            elif numberOfLvl == 2:

                x\_offset = 500

                y\_offset = 80

            elif numberOfLvl == 3:

                x\_offset = 70

                y\_offset = 350

            else:

                x\_offset = (width - cell\_width \* len(final\_results[0])) // 2

                y\_offset = 100

            for i, row in enumerate(final\_results):

                for j, item in enumerate(row):

                    pygame.draw.rect(screen, BLACK, (x\_offset + j \* cell\_width, y\_offset + i \* row\_height, cell\_width, cell\_height), 2)

                    if not item:

                        item = ""

                    text = font.render(str(item), True, BLACK)

                    text\_rect = text.get\_rect(center=(x\_offset + j \* cell\_width + cell\_width // 2, y\_offset + i \* row\_height + cell\_height // 2))

                    screen.blit(text, text\_rect)

            box\_rect = text\_finish.get\_rect(center=((width // 2, 60)))

            pygame.draw.rect(screen, WHITE, box\_rect)

        if not player\_control and not ai\_control and not show\_results:

            show\_results = True

            plot\_trajectories(car.trail, ai\_car.trail)

        pygame.display.update()

        clock.tick(60)