|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**  **«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ**  **(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»** | | | | |
| **Журнал практики** | | | | |
| Институт № 3 | «Системы управления, информатика и электроэнергетика» | | | |
|  |  | | | |
| Кафедра | \_\_№304\_ | | Учебная группа | М3О-309Б-21 |
|  |  | |  |  |
| ФИО обучающегося | | \_\_\_Поцепай Владимир Николаевич\_\_\_\_ | | |
|  | |  | | |
| Направление подготовки/  специальность | | 09.03.01 «Информатика и вычислительная техника» | | |
|  | | *шифр, наименование направления подготовки/специальности* | | |
|  | |  | | |
| Вид практики | | \_\_\_Производственная\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | |
|  | | *учебная, производственная, преддипломная или другой вид практики* | | |
| Оценка за практику | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Звонарева Галина Александровна | | |

Москва

2024

|  |  |  |
| --- | --- | --- |
| 1. **Место и сроки проведения практики:** | | |
| Наименование организации: | \_\_АО «Концерн радиостроения «Вега»\_\_\_ | |
| Сроки проведения практики |  | |
| дата начала практики: | \_28.06.2024\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | |
| дата окончания практики: | \_25.07.2024\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | |
| 1. **Инструктаж по технике безопасности:** | | |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | /\_\_\_\_\_\_Лунев С.А.\_\_\_\_\_/ | \_28\_\_ \_\_\_июня\_\_\_ 2024г. |
| *подпись проводившего* | *расшифровка подписи* | *дата проведения* |
| 1. **Индивидуальное задание обучающегося:** | | |
| \_Создание графического приложения для моделирования и анализа траекторий самолетов. Данные траекторий получаются от сервера по TCP протоколу \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | |
| 1. **План выполнения индивидуального задания обучающегося:** | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **№ п/п** | **Место проведения** | **Тема** | **Период выполнения** |
|  | отдел №10 | Знакомство с предприятием. Инструктаж по технике безопасности. | 28.06.2024 |
|  | отдел №10 | Исследование предметной области. | 03.07.2024 |
|  | отдел №10 | Изучение передачи данных с использованием TCP протоколов. | 05.07.2024 |
|  | отдел №10 | Изучение основ библиотеки PyGame. | 12.07.2024 |
|  | отдел №10 | Разработка графического приложения для приема, моделирования, анализа и отображения данных траекторий от сервера по TCP протоколу. | 17.07.2024 |
|  | отдел №10 | Отладка и тестирование ПО. | 20.07.2024 |
|  | отдел №10 | Анализ результатов. | 22.07.2024 |
|  | отдел №10 | Оформление и сдача отчета. | 25.07.2024 |

|  |  |  |
| --- | --- | --- |
| **Утверждаю** | | |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | /\_\_\_\_Звонарева Г.А.\_\_\_/ | \_28\_\_ \_\_\_июня\_\_\_ 2024г. |
| *подпись руководителя от МАИ* | *расшифровка подписи* | *дата утверждения\** |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | /\_\_\_\_\_\_Одегов Р.А.\_\_\_\_\_/ | \_28\_\_ \_\_\_июня\_\_\_ 2024г. |
| *подпись руководителя от организации/предприятия* | *расшифровка подписи* | *дата утверждения\** |
| **Ознакомлен** | | |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | /\_\_\_\_Поцепай В.Н.\_\_\_/ | \_28\_\_ \_\_\_июня\_\_\_ 2024г. |
| *подпись обучающегося* | *расшифровка подписи* | *дата ознакомления\** |

*\*Дата утверждения и ознакомления – дата начала практики*

|  |  |  |
| --- | --- | --- |
| 1. **Отзыв руководителя практики от организации/предприятия:** | | |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | /\_\_\_\_\_\_Одегов Р.А.\_\_\_\_\_/ | \_\_\_ \_\_\_\_\_\_ 2024г. |
| *подпись руководителя от организации/предприятии* | *расшифровка подписи* | *дата* |

|  |
| --- |
| 1. **Отчет обучающего по практике:** |

**Содержание**

Исследование предметной области............................................................................5

Изучение передачи данных с использованием TCP протоколов.............................7

Изучение основ библиотеки PyGame.......................................................................10

Разработка графического приложения для приема, моделирования, анализа и отображения данных траекторий от сервера по TCP протоколу……..…………..13

Отладка и тестирование ПО......................................................................................19

Анализ результатов....................................................................................................22

Список литературы………………………………...………………………….....…23

Приложение. Полный код программы.....................................................................25

**Исследование предметной области**

**Архитектура «клиент-сервер»**

«Клиент — сервер» — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Фактически клиент и сервер — это программное обеспечение. Обычно программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов, но они могут быть расположены также и на одной машине. Программы-серверы ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных (например, передача файлов посредством HTTP, FTP, BitTorrent, потоковое мультимедиа или работа с базами данных) или в виде сервисных функций (например, работа с электронной почтой, общение посредством систем мгновенного обмена сообщениями или просмотр web-страниц во всемирной паутине). Поскольку одна программа-сервер может выполнять запросы от множества программ-клиентов, её размещают на специально выделенной вычислительной машине, настроенной особым образом, как правило, совместно с другими программами-серверами, поэтому производительность этой машины должна быть высокой. Из-за особой роли такой машины в сети, специфики её оборудования и программного обеспечения, её также называют сервером, а машины, выполняющие клиентские программы, соответственно, клиентами.

Характеристика клиент-сервер описывает отношения взаимодействующих программ в приложении. Серверный компонент предоставляет функцию или услугу одному или нескольким клиентам, которые инициируют запросы на такие услуги. Серверы классифицируются по предоставляемым ими услугам. Например, веб-сервер обслуживает веб-страницы, а файловый сервер обслуживает компьютерные файлы. Общий ресурс может быть любой из программного обеспечения и электронных компонентов компьютера — сервера, от программ и данных в процессорах и запоминающих устройств. Совместное использование ресурсов сервера представляет собой услугу.

Является ли компьютер клиентом, сервером или и тем, и другим, определяется характером приложения, которому требуются сервисные функции. Например, на одном компьютере могут одновременно работать веб-серверы и программное обеспечение файлового сервера, чтобы обслуживать разные данные для клиентов, отправляющих различные типы запросов. Клиентское программное обеспечение также может взаимодействовать с серверным программным обеспечением на том же компьютере. Связь между серверами, например, для синхронизации данных, иногда называется межсерверной.

Архитектуры «клиент-сервер» - один из основных принципов работы сети Интернет. Любой веб- сайт, или приложение в Интернет работает на сервере, а его пользователи являются клиентами. Социальные сети, сайты электронной коммерции, мобильные приложения, устройства Интернета вещей (умные колонки или смарт-часы) работают на основе клиент-серверной архитектуры.

Хорошим примером работы системы «клиент-сервер» является автомобильный навигатор. Приложение навигации на сервере собирает данные с многих смартфонов пользователей, на которых установлены клиенты приложения. Кроме того, приложение навигации использует ещё и данные с сервера базы данных – геоинформационной системы, который предоставляет данные, например, о текущих ремонтах дорог, о появлении новых дорог и пр. Данные со многих клиентов (местоположение, скорость) обрабатывается сервером навигации и выдаётся на смартфоны пользователей в виде информации о средней скорости движения по тому или иному участку маршрута.

Практически любая корпоративная сеть или ИТ-система предприятия, как правило, строится по архитектуре «клиент-сервер». В небольших сетях (3-5 компьютеров в компании) функции сервера может выполнять один из рабочих компьютеров. Если число машин в организации более 10, то лучше сделать выделенный сервер (почтовый сервер, приложений, баз данных и пр.), который будет заниматься обслуживанием клиентов – компьютеров и телефонов сотрудников организации.

В домашних сетях архитектура «клиент-сервер» тоже используется довольно часто. Например, в домашнюю сеть могут быть объединены компьютеры членов семьи, один из которых выполняет функции сервера. В домашнюю сеть также могут быть включены такие устройства, как умные колонки, умные домашние устройства, а также «умные» счётчики (вода, электричество) и т.д. Тогда в системе управления сервера, будут видны все параметры, данные и медиафайлы (музыка, видео, фото), а также «умные устройства».

**Изучение передачи данных с использованием TCP протоколов**

Работу по обеспечению для трафика датаграмм надежности соединения между приложениями выполняет протокол TCP, который реализуется на каждом из конечных хостов. Поверх протокола TCP реализованы службы WWW, регистрации с терминала, пересылки файлов и обработки электронной почты.

TCP можно рассматривать как средство обеспечения запросов данных (data call) по аналогии с обычными телефонными звонками. Вызывающая сторона указывает точку назначения, а на другом конце слушающее приложение реагирует на поступающие вызовы и устанавливает соединение. Производится обмен данными между двумя концами соединения, а но завершении обмена оба партнера говорят: "До свидания" и вешают трубки.

TCP естественным образом интегрируется в окружение клиент/сервер. Серверное приложение прослушивает (listen) поступающие запросы на соединение. Например, службы WWW, пересылки файлов или доступа с терминала прослушивают запросы, поступающие от клиентов. Коммуникации в TCP запускаются соответствующими подпрограммами, которые и инициализируют соединение с сервером.

**Порты приложения**

Клиент должен идентифицировать службу, к которой он хочет получить доступ. Это выполняется через спецификацию IP-адреса службы хоста и его номера порта TCP. Как и для UDP, номера портов TCP находятся в диапазоне от 0 до 65 535. Порты в диапазоне от 0 до 1023 называются общеизвестными (well-known) и используются для доступа к стандартным службам. комбинация.

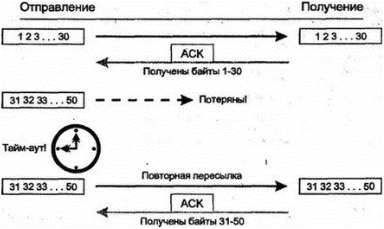
**Адреса socket**

IP-адреса и порта для коммуникации называется адресом socket. Соединение TCP полностью идентифицируется адресом socket на каждом конце данного соединения. Заголовок каждой датаграммы содержит IP-адреса источника и назначения. Номера портов источника и назначения указываются в заголовке сегмента TCP. Обычно сервер способен одновременно управлять несколькими клиентами. Уникальные адреса socket сервера присваиваются одновременно всем его клиентам. Поскольку датаграмма содержит сегмент соединения TCP, идентифицирующийся IP-адресами и портами, серверу очень просто отслеживать несколько соединений с клиентами. Для обеспечения надежной пересылки данных в TCP используются нумерация (numbering) и подтверждение (acknowledgment — ACK). Схема нумерации TCP несколько необычна: каждый пересылаемый по соединению октет рассматривается как имеющий порядковый номер. Заголовок сегмента TCP содержит порядковый номер первого октета данных этого сегмента.

От приемника требуется подтверждение получения данных. Если ACK не приходит за интервал тайм-аута, данные передаются повторно. Этот способ называется позитивным подтверждением с ретрансляцией (positive acknowledgment with retransmission).

Получатель данных TCP проводит строгий контроль входящих порядковых номеров, чтобы проверить последовательность получения данных и отсутствие потерянных частей. Поскольку ACK случайным образом может быть потерян или задержан, к получателю могут поступить дублированные сегменты. Порядковые номера позволяют определить дублирование данных, которые далее отбрасываются.

На рис. 2.1 показан упрощенный взгляд на тайм-аут и повторную пересылку в TCP.



*Рис. 2.1. Тайм-аут и повторная пересылка в TCP*

**Установка соединения**

Перед коммуникацией каждое из них вызывает подпрограмму для формирования блока памяти, который будет использован для хранения параметров TCP и IP данного соединения, например адресов socket, текущего порядкового номера, начального значения времени жизни и т.д.

Серверное приложение ожидает появления клиента, который, желая получить доступ к серверу, выдает запрос на соединение (connect), идентифицирующий IP-адрес и порт сервера.

Существует одна техническая особенность. Каждая сторона начинает нумерацию каждого байта не с единицы, а со случайного порядкового номера (далее мы узнаем, для чего это делается). Исходная спецификация дает совет: начальный порядковый номер генерировать на основе 32-разрядного внешнего таймера, увеличивающего значения примерно каждые 4 мкс.

**Сценарий соединения**

Процедуру соединения часто называют тройным рукопожатием (three-way handshake), поскольку для установки соединения производится обмен тремя сообщениями — SYN, SYN и ACK.

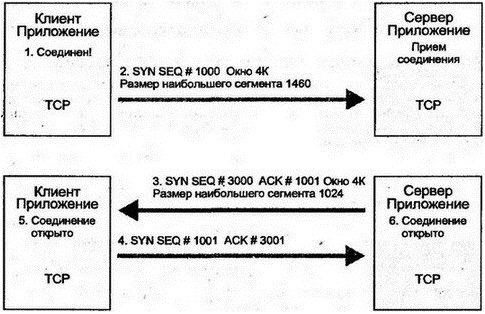
Во время установки соединения партнеры обмениваются тремя важными порциями информации:

1. Объем буферного пространства для приема данных
2. Максимальное количество данных, переносимое во входящем сегменте
3. Начальный порядковый номер, используемый для исходящих данных

Структура памяти персонального компьютера может ограничивать поступающие порции данных 1 Кбайт, а суперкомпьютер управляется с большими сегментами.

Способность управлять тем, как посылает данные другая сторона, является важным свойством, обеспечивающим масштабируемость TCP/IP.

На рис. 2.2 показан пример сценария соединения.

  
 *Рис. 2.2. Установление соединения*

Выполняются следующие операции:

1. Сервер инициализируется и становится готовым к соединению с клиентами (это состояние называется пассивным открытием — passive open).
2. Клиент запрашивает у TCP открытие соединения с сервером по указанному IP-адресу и порту (это состояние называется активным открытием — active open).
3. Клиентская TCP получает начальный порядковый номер (в данном примере — 1000) и посылает *сегмент синхронизации* (synchronize segment — SYN). В этом сегменте пересылается порядковый номер, размер приемного окна (4 К) и размер наибольшего сегмента, который может принять клиент (1460 байт).
4. Когда поступает SYN, серверная TCP получает *свой* начальный порядковый номер (3000). Она посылает сегмент SYN, содержащий начальный порядковый номер (3000), ACK 1001 (что означает нумерацию первого посланного клиентом байта как 1001), размер приемного окна (4 К) и размер наибольшего сегмента, который сможет получить сервер (1024 байта).
5. Клиентская TCP, получив от сервера сообщение SYN/ACK, отсылает обратно ACK 3001 (первый байт посланных сервером данных должен нумероваться как 3001).
6. Клиентская TCP указывает своему приложению на открытие соединения.
7. Серверная TCP, получив от клиентской TCP сообщение ACK, информирует свое приложение об открытии соединения.

Клиент и сервер анонсируют свои правила для принимаемых данных, синхронизируют свои порядковые номера и становятся готовыми к обмену данными. Спецификация TCP разрешает и другой сценарий (не слишком удачный), когда равные между собой приложения одновременно выполняют активное открытие друг друга.

**Изучение основ библиотеки PyGame**

Библиотека PyGame является мощным инструментом для разработки игр и мультимедийных приложений на языке Python. Она предоставляет удобные средства для работы с графикой, звуком и взаимодействием с пользователем. В данной главе мы рассмотрим основные возможности PyGame, включая рисование различных фигур, управление игровым циклом и обработку событий.

**Главный цикл программы**

Главный цикл программы — это основа любого приложения на PyGame. Он отвечает за обновление состояния игры, обработку событий и отрисовку графики. Обычно он выглядит следующим образом:

*python*

*running = True*

***while*** *running:*

***for*** *event* ***in*** *pygame.event.get():*

***if*** *event.type == pygame.QUIT:*

*running = False*

*# Логика игры и отрисовка*

*screen.fill(WHITE) # Очистка экрана*

*# Здесь можно рисовать фигуры*

*pygame.display.flip() # Обновление экрана*

**FPS и разрешение окна**

FPS (Frames Per Second) — это количество кадров, отображаемых на экране в секунду. Для управления FPS в PyGame используется объект pygame.time.Clock().

*python*

*clock = pygame.time.Clock()*

***while*** *running:*

*# Обработка событий и логика игры*

*clock.tick(60) # Ограничение FPS до 60*

Разрешение окна задается при его создании с помощью pygame.display.set\_mode(), где параметры — это ширина и высота окна.

*# Создание окна screen = pygame.display.set\_mode((800, 600))*

**Рисование фигур**

*Прямоугольник*

Для рисования прямоугольника в PyGame используется функция pygame.draw.rect(). Эта функция принимает несколько параметров, включая поверхность, цвет, координаты и размеры прямоугольника.

*python*

***import*** *pygame*

*# Инициализация PyGame*

*pygame.init()*

*# Создание окна*

*screen = pygame.display.set\_mode((800, 600))*

*# Цвета*

*WHITE = (255, 255, 255)*

*BLUE = (0, 0, 255)*

*# Рисование прямоугольника*

*pygame.draw.rect(screen, BLUE, (50, 50, 100, 50))*

*# Обновление экрана*

*pygame.display.flip()*

*Круг*

Для рисования круга используется функция pygame.draw.circle(). Она позволяет задавать центр круга, радиус и цвет.

*python*

*# Рисование круга*

*pygame.draw.circle(screen, BLUE, (200, 200), 50)*

*pygame.display.flip()*

*Прямая*

Для рисования прямой линии применяется функция pygame.draw.line(), которая требует указания начальной и конечной точек, а также цвета и толщины линии.

*python*

*# Рисование линии*

*pygame.draw.line(screen, BLUE, (300, 300), (400, 400), 5)*

*pygame.display.flip()*

*Прямоугольник со скругленными углами*

Для создания прямоугольника со скругленными углами используется функция pygame.draw.rect(), но с дополнительным параметром border\_radius.

*python*

*# Рисование прямоугольника со скругленными углами*

*pygame.draw.rect(screen, BLUE, (450, 50, 150, 100), border\_radius=20)*

*pygame.display.flip()*

*Обводка фигур*

Чтобы добавить обводку к фигурам, необходимо указать толщину линии в качестве последнего параметра при рисовании. Например, для прямоугольника:

*python*

*# Рисование прямоугольника с обводкой*

*pygame.draw.rect(screen, BLUE, (50, 150, 100, 50), 5) # 5 - толщина обводки*

*pygame.display.flip()*

**Обработка событий**

PyGame предоставляет мощные инструменты для обработки событий, таких как нажатия клавиш и клики мыши. Все события обрабатываются в главном цикле программы.

*Нажатия клавиш*

Для обработки нажатий клавиш используется событие pygame.KEYDOWN. Например, чтобы реагировать на нажатие клавиши "ESC":

*python*

***for*** *event* ***in*** *pygame.event.get():*

***if*** *event.type == pygame.KEYDOWN:*

***if*** *event.key == pygame.K\_ESCAPE:*

*running = False*

*Клики мыши*

События мыши обрабатываются с помощью pygame.MOUSEBUTTONDOWN. Например, для обработки клика левой кнопкой мыши:

*python*

***if*** *event.type == pygame.MOUSEBUTTONDOWN:*

***if*** *event.button == 1: # Левый клик*

*pos = pygame.mouse.get\_pos() # Получение позиции курсора*

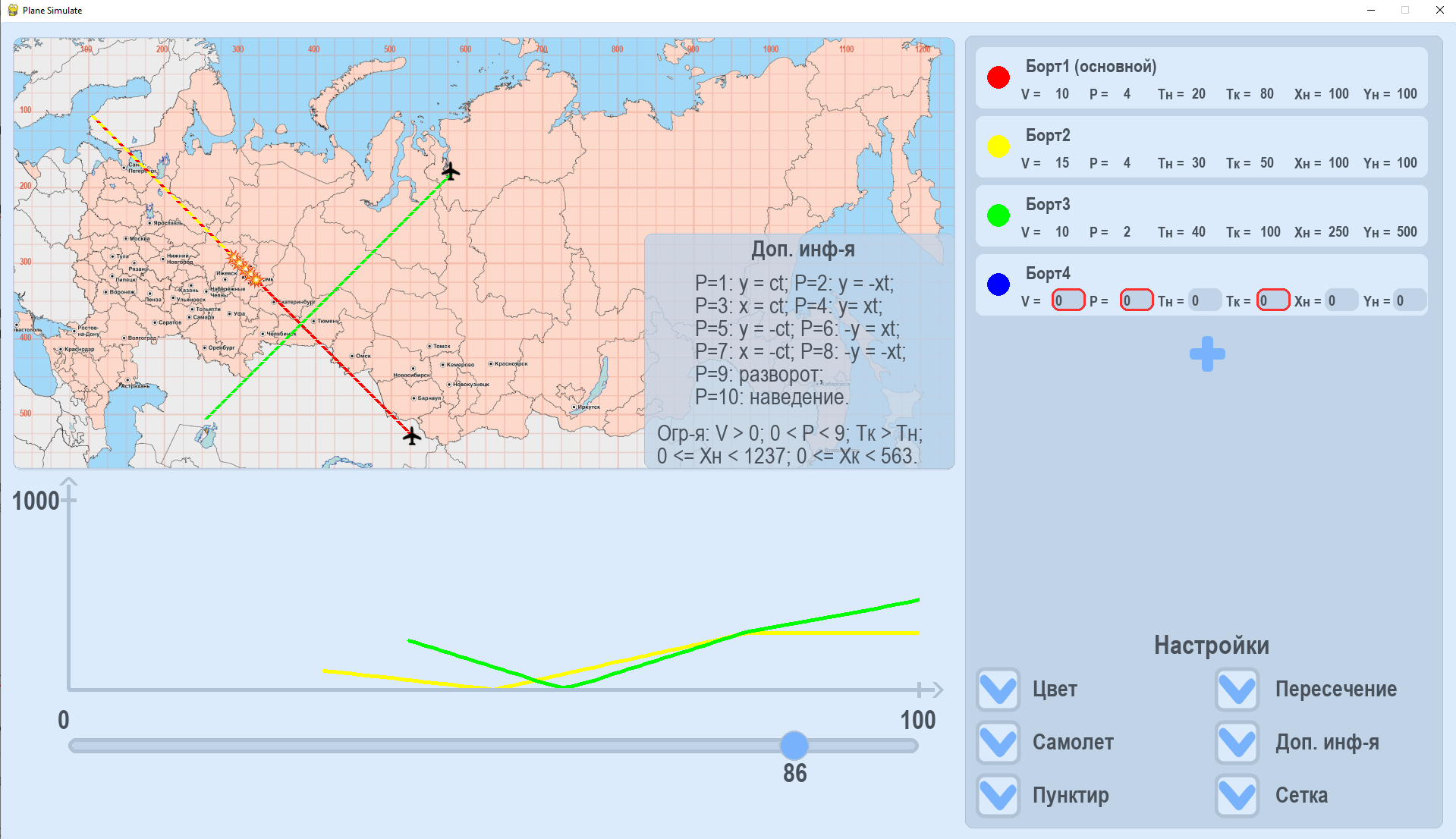
**Разработка графического приложения для приема, моделирования, анализа и отображения данных траекторий от сервера по TCP протоколу**

**Дизайн клиента**

Для разработки графического приложения клиента мы используем библиотеку PyGame, которая предоставляет удобные средства для создания окна, рисования графики и обработки событий. Дизайн клиента включает в себя:

* Основное окно приложения с полем для отображения траекторий
* Панель управления с полями для ввода данных (inputBox) и кнопками, обрабатывающими введенные данные и отправляющие запрос на сервер с введенными параметрами (знак “+”)
* Меню настроек, состоящее из chackBox’ов для работы с bool переменными, за счет которых включаются и выключаются отдельные доп составляющие визуализации траекторий
* Информационное поля для вывода данных
* Панель анализа взаимного расположения авиа-бортов. Она состоит из графика, где по 0Y отложено расстояние между i-м и основным авиа-бортом, а по 0X отсчитывается модельное время
* Слайдер отвечающий за смену модельного времени

Как выглядит дизайн клиента можно видеть на Рис 4.1:



*Рис. 4.1 Дизайн клиента*

**Выбор стека**

Для разработки данного приложения были выбраны язык программирования Python и библиотека PyGame по следующим причинам:

1. **Python** является высокоуровневым языком с простым и понятным синтаксисом, что позволяет быстро разрабатывать прототипы и реализовывать функциональность.
2. **PyGame** хорошо подходит для создания 2D-графических приложений, предоставляя удобные функции для рисования, управления окном и обработки событий.
3. **Python** имеет обширную экосистему библиотек, включая модули для работы с сокетами, математикой и анализом данных, что позволяет расширять функциональность приложения.
4. **Python** и **PyGame** кроссплатформенны и могут работать на различных операционных системах, что важно для обеспечения совместимости приложения.

Так же в процессе разработки приложения были использованы библиотеки:

* Os
* Sys
* Math
* Json
* Socet
* pyinstaller

**Архитектура**

Архитектура приложения состоит из двух основных частей: клиента и сервера. Клиент отвечает за отображение графического интерфейса и взаимодействие клиента с приложением, а также прием данных от сервера и визуализацию траекторий. Сервер по полученным параметрам в запросе генерирует и отправляет данные траекторий клиенту по TCP протоколу.

*Клиентская часть*

Клиентская часть реализована в модулях client.py , plus.py, bort.py, inputbox.py, checkbox.py, slider.py, cells.py, goToServer.py, const.py:

* В  client.py  представлено: создание окна игры, основной игровой цикл приложения, связь с другими модулями, инициализация объектов классов и сам процесс отрисовки элементов интерфейса
* В plus.py реализован класс Plus, который отвечает за создание нового борта и получение его траектории при корректно введенных параметрах и переход к инициализации следующего авиа-борта. Элемент этого класса и вызов его функций происходит в client.py
* В bort.py реализован класс Bort, главной задачей которого является отрисовка авиа-борта. Создание элементов этого класса и вызов его функций происходит в классе Plus в файле plus.py
* В inputbox.py реализован класс InputBox, отвечающий за рисования текстового поля и отслеживания его редактирования, с соответствующими изменениями дизайна панели: выбран и редактируется – синяя обводка, некорректные данные – красная обводка, доступно редактирование – серая панель, редактирование не доступно – отсутствие панели вокруг текста. Создание элементов этого класса и вызов его функций происходит в классе Plus в файле plus.py
* В checkbox.py реализован класс ChackBox, отвечающий за рисования панели для выбора bool значения переменной. Наличие галочки в боксе – True, отсутствие – False. Функции рисования вызываются в client.py, и соответствуют настройкам таких особенностей визуализации, как цвет траектории, фигурка самолета, пунктирная траектория, иконка пересечения (столкновения), вывод дополнительной информации, сетка на карте для простоты расчета и понимания представленной информации
* В slider.py, реализован класс Slider, который имеет функцию Draw, рисующую область с ползунком, для изменения значения моментального времени в диапазоне с 0 до Tmax (максимальное время системы- время когда все самолеты прекратили своё движение). Также в этой функции рисуется и координатная плоскость для вывода графиков взаимного расположения авиа-бортов
* В cells.py, реализован класс Cells, основная цель которого рисование клетки поверх карты
* В GoToSerwer.py, реализована функция GoServer, в которой параметры, полученные от inputbox’ов (v - скорость самолета, p - тип траектории, Tн - время вылета самолета, Tк – время конца полета, Xн – начальная координата x, Yн – начальная координата y) и значения траектории основного самолета преобразуются в json файл и передаются на сервер по TCP протоколу, полученный ответ от сервера переводится из json назад в list of list of float и возвращает этот результат
* В const.py, реализованы различные функции такие, как DashedLine, InterpolatePoints, FindMaxInColumn, ChackConect, draw\_rect\_alpha, а также инициализированы отдельные переменные от значения которых зависят форматы выводов данных, размеры окон, толщина линий и т. п.

*Серверная часть*

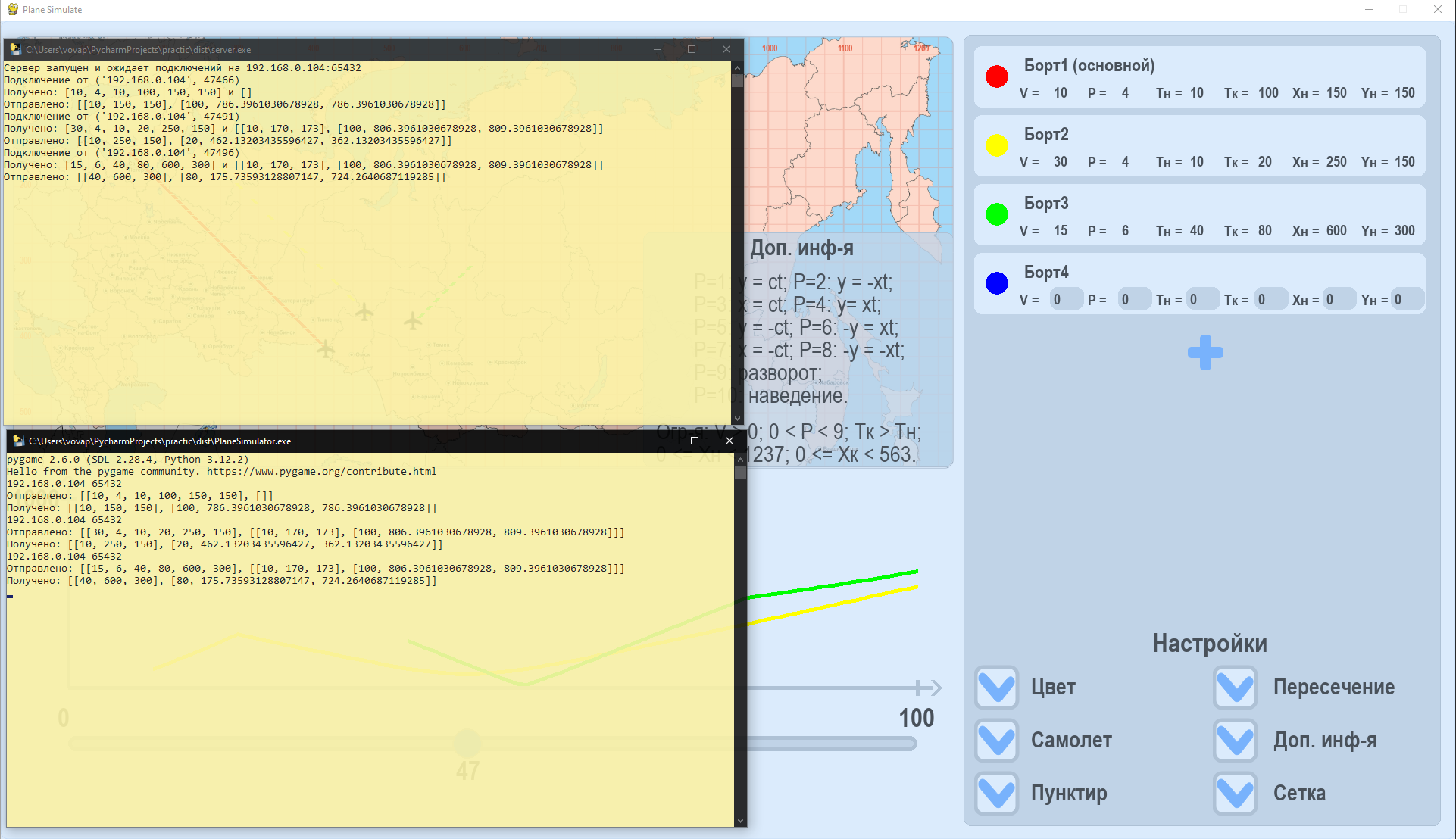
Серверная часть реализована в модуле server.py и отвечает за генерацию и отправку данных траекторий клиенту. Она включает в себя:

* Функция start\_server, отвечающий за создание TCP-сервера и обработку подключений клиентов
* Функция handle\_client\_request, отвечающая за получение данных, их обработку, генерацию результата и его передачу назад клиенту

**Соединение клиента с сервером**

Для установления соединения между клиентом и сервером используются сокеты. Клиент создает TCP-сокет, подключается к серверу по указанному IP-адресу и порту, а затем ожидает получения данных от сервера. Сервер также создает TCP-сокет, привязывается к определенному IP-адресу и порту и ожидает входящих подключений клиентов. При установлении соединения клиент отправляет параметры траектории нового авиа-борта на сервер, он в свою очередь обрабатывает полученные данные и отправляет дискретную функцию траектории клиенту, который обрабатывает и визуализирует её.

Результат работы можно наблюдать на Рис 4.2.



*Рис. 4.2 Результат работы соединения клиента и сервера*

Таким образом, разработанное приложение позволяет клиенту подключаться к серверу, получать данные траекторий, моделировать и анализировать их, а также визуализировать на экране с использованием библиотеки PyGame.

**Отладка и тестирование ПО**

После написания приложения клиент-сервер, была проведена отладка различными способами

**Логирование**

Использованы логирования для отслеживания действий приложения и выявления ошибок. Важно записывать ключевые события, такие как успешные подключения, получение данных от сервера и возникающие исключения. Это поможет в диагностике проблем.

**Отладка с помощью инструментов**

Применение отладчиков (например, встроенного в IDE) для пошагового выполнения кода, проверки значений переменных и анализа потока выполнения. Это позволяет выявлять логические ошибки и проблемы с подключением.

**Модульное тестирование**

Разработка модульных тестов для отдельных компонентов приложения, таких как функции обработки данных и классы, отвечающие за визуализацию. Это позволяет убедиться, что каждый модуль работает корректно в изоляции.

**Интеграционное тестирование**

Проверка взаимодействия между клиентом и сервером. Это включает в себя тестирование обмена данными, чтобы убедиться, что клиент правильно обрабатывает данные, полученные от сервера.

**Тестирование пользовательского интерфейса**

Проверка корректности работы графического интерфейса, включая обработку событий, таких как нажатия кнопок и взаимодействие с элементами управления. Это важно для обеспечения удобства использования приложения.

**Стресс-тестирование**

Проверка устойчивости приложения при высоких нагрузках, например, при одновременных подключениях нескольких клиентов. Это поможет выявить проблемы с производительностью и стабильностью приложения.

**Исключения**

Обработка исключений в коде, чтобы предотвратить аварийное завершение приложения. Важно правильно обрабатывать ошибки соединения, тайм-ауты и другие возможные проблемы.

**Пользовательские уведомления**

Создание системы уведомлений для пользователей о возникших ошибках или проблемах с соединением. Это улучшает взаимодействие с пользователем и помогает ему понять, что происходит.

**Документация**

Составление отчетов о проведенных тестах, включая выявленные ошибки и способы их устранения. Это поможет в дальнейшем развитии приложения и улучшении его качества.

**Соответствие требованиям**

Оценка того, насколько разработанное приложение соответствует изначальным требованиям и целям проекта. Были ли реализованы все необходимые функции для приема, моделирования, анализа и отображения данных траекторий.

**Удобство использования**

Анализ удобства и интуитивности графического интерфейса клиента. Насколько легко пользователю взаимодействовать с приложением, понятны ли элементы управления и отображаемая информация.

**Скорость передачи данных**

Оценка скорости передачи данных траекторий от сервера к клиенту. Насколько быстро клиент получает и обрабатывает поступающие данные.

**Нагрузочное тестирование**

Проверка работоспособности приложения при одновременном подключении нескольких клиентов. Насколько стабильно приложение работает под нагрузкой.

**Обработка ошибок**

Анализ того, насколько хорошо приложение справляется с обработкой ошибок, таких как потеря соединения, некорректные данные от сервера или клиента. Насколько устойчиво приложение к сбоям.

**Стабильность**

Оценка стабильности работы приложения в течение длительного времени. Насколько часто возникают сбои или зависания.

**Качество отображения**

Анализ качества визуализации траекторий на экране клиента. Насколько четко и наглядно отображаются траектории, понятны ли пользователю представленные данные.

**Интерактивность**

Оценка интерактивности визуализации, таких как возможность масштабирования, вращения или выделения отдельных траекторий. Насколько удобно пользователю работать с визуализацией.

**Улучшения и расширения**

Определение возможных направлений для улучшения и расширения функциональности приложения, основываясь на полученных результатах. Какие аспекты можно доработать или добавить новые возможности.

**Перспективы применения**

Анализ перспектив применения разработанного приложения в реальных условиях. Для каких задач оно может быть полезно и в каких областях может найти применение. Таким образом, можно всесторонне оценить эффективность разработанного клиент-серверного приложения, выявить его сильные и слабые стороны, а также определить направления для дальнейшего совершенствования.

**Анализ результатов**

В процессе выполнения работы были реализованы следующие задачи: были изучены основы архитектуры «клиент-сервер», протокола TCP, языка python в частности библиотеки PyGame. Было разработано descktop приложение, состоящее из двух частей - клиент и сервер, на языке программирования Python.

В разработанной библиотеке были реализованы алгоритмы для передачи данных со стороны клиента на сервер и обратно, а также алгоритмы визуализации полученной информации.

При тестировании приложения была проверена передача различных параметров. В результате анализа работы приложения, алгоритмы работают достоверно и точно.

**Список литературы**

1. **"Изучаем Python"** - Мартин Чик. Книга охватывает основы языка Python, включая его синтаксис, структуры данных и принципы объектно-ориентированного программирования, что является важным для разработки приложений.
2. **"Python. Книга для профессионалов"** - Дэниел Б. К. К. Эта книга предоставляет углубленное понимание языка Python, включая продвинутые темы, такие как работа с модулями, обработка исключений и ООП.
3. **"Глубокое погружение в Python"** - Ребекка Р. К. В этой книге рассматриваются более сложные аспекты Python, включая архитектурные паттерны и лучшие практики разработки, что полезно для проектирования клиент-серверных приложений.
4. **"Программирование на Python. Основы и практика"** - Игорь С. К. Книга предлагает практические примеры и объясняет, как использовать Python для разработки различных приложений, включая клиент-серверные.
5. **"TCP/IP Illustrated, Volume 1: The Protocols"** - W. Richard Stevens. Эта книга является классическим источником по протоколам TCP/IP и предоставляет глубокое понимание работы сетевых протоколов, что важно для разработки клиент-серверных приложений.
6. **"Python Network Programming Cookbook"** - M. O. K. Эта книга предлагает рецепты по работе с сетевым программированием на Python, включая создание серверов и клиентов, что будет полезно для реализации TCP-соединений.
7. **"Learning Python"** - Mark Lutz. Книга охватывает все аспекты языка Python, включая ООП, и является отличным ресурсом для глубокого изучения языка.
8. **"Fluent Python"** - Luciano Ramalho. Эта книга подходит для более опытных программистов и охватывает продвинутые концепции Python, включая использование ООП и функционального программирования.
9. **"Python для профессионалов"** - Дэвид Бизли. Книга предлагает углубленный анализ Python и его возможностей, включая создание сложных приложений и архитектурные паттерны.
10. **"Computer Networking: A Top-Down Approach"** - James Kurose, Keith Ross. Книга предоставляет обширное введение в компьютерные сети, включая протоколы и архитектуры, что полезно для понимания клиент-серверного взаимодействия.
11. Статья "Python сокеты, сетевое программирование: TCP сервер и клиент" на сайте SnakeProject.ru. В статье рассматривается реализация сокетов в Python на примере протокола TCP.
12. Статья "Удалённое исполнение системных команд по запросу через сокеты на Python" на Хабре. Описывается разработка простого протокола поверх TCP для передачи данных между сервером и клиентом.
13. Статья "Руководство по программированию сокетов на Python. Протокол и сообщения" на Хабре. Объясняется, как работать с сокетами в Python, в том числе с использованием протокола TCP.
14. Ответ на вопрос "Как передавать любые файлы, используя протокол TCP\IP и Python ServerSocket" на Stack Overflow. Приводятся примеры кода сервера и клиента для передачи файлов по TCP.
15. Статья "Самоучитель по Python для начинающих. Часть 21: Основы разработки игр на Pygame" на Proglib.io. Дается введение в разработку игр с использованием библиотеки PyGame.

**Приложение. Полный код программы**

**Client.py**

from const import \*

from chackbox import ChackBox

from slider import Slider

from plus import Plus

from goToServer import GoServer

from cells import Cells

#-------------oткрытие окна игры--------------

pygame.init()

pygame.mixer.init()

pygame.font.init()

screen = pygame.display.set\_mode((screen\_width, screen\_height))

pygame.display.set\_caption("Plane Simulate")

input\_text=""

#-------------инициализация-------------------

fontBold = pygame.font.SysFont(font\_name, 35, bold=True)

fontNormal = pygame.font.SysFont(font\_name, 30, bold=True)

fontNormalNoBold = pygame.font.SysFont(font\_name, 30, )

SettingSurface = fontBold.render("Настройки", True, textCollor)

SettingRect = SettingSurface.get\_rect(topleft=(1520, 800), size=(35, 180))

ColorSurface = fontNormal.render("Цвет", True, textCollor)

ColorRect = SettingSurface.get\_rect(topleft=(1360, 860), size=(35, 180))

PlainSurface = fontNormal.render("Самолет", True, textCollor)

PlainRect = SettingSurface.get\_rect(topleft=(1360, 930), size=(35, 180))

PunctirSurface = fontNormal.render("Пунктир", True, textCollor)

PunctirRect = SettingSurface.get\_rect(topleft=(1360, 1000), size=(35, 180))

ConnectSurface = fontNormal.render("Пересечение", True, textCollor)

ConnectRect = SettingSurface.get\_rect(topleft=(1680, 860), size=(35, 180))

InfoSurface = fontNormal.render("Доп. инф-я", True, textCollor)

InfoRect = SettingSurface.get\_rect(topleft=(1680, 930), size=(35, 180))

NoiseSurface = fontNormal.render("Сетка", True, textCollor)

NoiseRect = SettingSurface.get\_rect(topleft=(1680, 1000), size=(35, 180))

CB1 = ChackBox(1285, 850, screen)

CB2 = ChackBox(1285, 920, screen)

CB3 = ChackBox(1285, 990, screen)

CB4 = ChackBox(1600, 850, screen)

CB5 = ChackBox(1600, 920, screen)

CB6 = ChackBox(1600, 990, screen)

listCB = [CB1.condition,CB2.condition,CB3.condition,CB4.condition,CB5.condition,CB6.condition]

listParam = [[]]

ListCord = []

ListConect = []

cordFirst = []

maxT = openT

Slider = Slider(screen)

Plus = Plus(screen)

Cell = Cells(screen)

"""настройка звука"""

#pygame.mixer.music.load(os.path.join(BASE, "media/music/main.mp3"))

#pygame.mixer.music.set\_volume(0.2)

#pygame.mixer.music.play(-1)

#click = pygame.mixer.Sound(os.path.join(BASE, "media/music/click.mp3"))

#click.set\_volume(0.2)

running = True

while running: #основной цикл программы

pygame.time.Clock().tick(FPS)

pygame.draw.rect(screen, mainCollor, (0, 0, screen\_width, screen\_height)) #подложка

screen.blit(map, (15, 18)) #карта

if listCB[5]:

Cell.Draw() #клетка

k = 0

for bortCords in ListCord: #проход по всем бортам

l = -1

for i in range(len(bortCords) - 1): #проход по всем координатам точек

if bortCords[i][0] <= Slider.condition - stepCord: # and bortCords[i][1]<xMax and bortCords[i][2]<yMax:

#координата раньше чем ограничивающая отметка времени

color = (0, 0, 0)

if listCB[0]: #цвет включен

color = Plus.borts[0].color[k]

else: #цвет выключен

color = textCollor

if listCB[2]: #пунктир включен

DashedLine(screen, (bortCords[i][1], bortCords[i][2]), (bortCords[i + 1][1], bortCords[i + 1][2]),

color, lineWidht, lineDash, lineGap)

else: #пунктир выключен

pygame.draw.line(screen, color, (bortCords[i][1], bortCords[i][2]),

(bortCords[i + 1][1], bortCords[i + 1][2]), lineWidht)

l = i

if listCB[1] and l>-1:# and bortCords[len(bortCords) - 1][1] < xMax and bortCords[len(bortCords) - 1][2] < yMax:

#самолет включен и нарисовано хоть что-то из траетории

screen.blit(plain, (bortCords[l+1][1] - 15, bortCords[l+1][2] - 15))

k += 1

if len(ListCord)>1 :

ListConect = ChackConect(ListCord)

for i in range(len(ListConect)):

for j in range(len(ListConect[i])):

if j+1 < len(ListConect[i]):

if ListConect[i][j+1][0]<=Slider.condition and ListConect[i][j+1][3] < conectEps and listCB[3]:

#пересечение включено, пересечение было до отметки ограничения времени,

#борты на растоянии меньше conectEps в одно и тоже время

screen.blit(conect, (ListConect[i][j][1]-15, ListConect[i][j][2]-15)) #пересечение

if listCB[4]: #доп инф-я включена

draw\_rect\_alpha(screen, (dopCollorAlfa), (848, 279, 410, 310) )#доп область

pygame.draw.rect(screen, borderCollor, (848, 279, 410, 310), 1, border\_radius=borderRad) # обводка

DopSurface = fontNormal.render("Доп. инф-я", True, textCollor)

DopRect = DopSurface.get\_rect(topleft=(989, 280), size=(35, 110))

screen.blit(DopSurface, DopRect)

"""Вывод инф-и о параметре P"""

DopSurface3 = fontNormalNoBold.render("""P=1: y = ct; P=2: y = -xt;""",True, textCollor)

DopRect3 = DopSurface.get\_rect(topleft=(865+50, 325), size=(35, 110))

screen.blit(DopSurface3, DopRect3)

DopSurface4 = fontNormalNoBold.render("""P=3: x = ct; P=4: y= xt;""",True, textCollor)

DopRect4 = DopSurface.get\_rect(topleft=(865+50, 355), size=(35, 110))

screen.blit(DopSurface4, DopRect4)

DopSurface5 = fontNormalNoBold.render("""P=5: y = -ct; P=6: -y = xt;""",True, textCollor)

DopRect5 = DopSurface.get\_rect(topleft=(865+50, 385), size=(35, 110))

screen.blit(DopSurface5, DopRect5)

DopSurface6 = fontNormalNoBold.render("""P=7: x = -ct; P=8: -y = -xt;""",True, textCollor)

DopRect6 = DopSurface.get\_rect(topleft=(865+50, 415), size=(35, 110))

screen.blit(DopSurface6, DopRect6)

DopSurface7 = fontNormalNoBold.render("""P=9: разворот;""",True, textCollor)

DopRect7 = DopSurface.get\_rect(topleft=(865+50, 445), size=(35, 110))

screen.blit(DopSurface7, DopRect7)

DopSurface8 = fontNormalNoBold.render("""P=10: наведение.""",True, textCollor)

DopRect8 = DopSurface.get\_rect(topleft=(865+50, 475), size=(35, 110))

screen.blit(DopSurface8, DopRect8)

"""Вывод информации о органичениях по параметрам"""

DopSurface1 = fontNormalNoBold.render(f"Огр-я: V > 0; 0 < P < {pMax}; Tк > Tн;", True, textCollor)

DopRect1 = DopSurface.get\_rect(topleft=(865, 523), size=(75, 355))

screen.blit(DopSurface1, DopRect1)

DopSurface2 = fontNormalNoBold.render(f"0 <= Xн < {xMax}; 0 <= Xк < {yMax}.", True, textCollor)

DopRect2 = DopSurface.get\_rect(topleft=(865, 553), size=(75, 355))

screen.blit(DopSurface2, DopRect2)

screen.blit(main, (0, 0)) #фон

if len(ListCord) > 1:

ListConect = ChackConect(ListCord)

for i in range(len(ListConect)):

"""параметры для рисования графика"""

x = 89

y = 880

MaxD = 1000

deltaY = 250 / max(100, MaxD)

deltaX = Slider.w / Slider.max

color = Plus.borts[0].color[i + 1]

for j in range(len(ListConect[i])):

if j + 1 < len(ListConect[i]):

if MaxD>ListConect[i][j][3]:

# рисуем графики

pygame.draw.line(screen, color, (x+(ListConect[i][j][0])\*deltaX, y-ListConect[i][j][3]\*deltaY),

(x+(ListConect[i][j+1][0])\*deltaX, y-ListConect[i][j+1][3]\*deltaY), lineWidht)

pygame.draw.rect(screen, dopCollor, (1271, 18, 630, 1045), border\_radius = borderRad) #правая область

pygame.draw.rect(screen, borderCollor, (1271, 18, 630, 1045), 1, border\_radius = borderRad) #обводка

""" Рисуем chackboxs """

CB1.Draw()

CB2.Draw()

CB3.Draw()

CB4.Draw()

CB5.Draw()

CB6.Draw()

""" Рисуем текст для chackboxs """

screen.blit(SettingSurface, SettingRect)

screen.blit(ColorSurface, ColorRect)

screen.blit(PlainSurface, PlainRect)

screen.blit(PunctirSurface, PunctirRect)

screen.blit(ConnectSurface, ConnectRect)

screen.blit(InfoSurface, InfoRect)

screen.blit(NoiseSurface, NoiseRect)

Slider.Draw() #слайдер

Plus.Draw() #плюс

pygame.display.flip() #переворот экрана

""" Отслеживание событий (нажатий мыши и кнопок) """

for event in pygame.event.get():

if event.type == pygame.QUIT: #крест (закрытие окна)

running = False

elif event.type == pygame.MOUSEBUTTONDOWN: #нажатие ПКМ

CB1.ChangeCB(event.pos)

CB2.ChangeCB(event.pos)

CB3.ChangeCB(event.pos)

CB4.ChangeCB(event.pos)

CB5.ChangeCB(event.pos)

CB6.ChangeCB(event.pos)

listCB = [CB1.condition, CB2.condition, CB3.condition, CB4.condition, CB5.condition, CB6.condition]

Slider.Change(event.pos)

newParamBort = Plus.Change(event.pos)

if newParamBort != [[]] and newParamBort != None:

listParam = Plus.listParam

try:

maxT = FindMaxInColumn(listParam, 3)

except:

print("maxT", maxT)

Slider.ImportMaxT(maxT)

Slider.Change([Slider.posPoint, Slider.y + 5])

if ListCord ==[]:

newListCord = GoServer(newParamBort)

cordFirst = newListCord

else:

newListCord = GoServer(newParamBort,cordFirst)

ListCord.append(newListCord)

for i in range(len(ListCord)):

ListCord[i]=InterpolatePoints(ListCord[i],stepCord,maxT)

for bort in Plus.borts:

input\_text = bort.Change(event.pos)

elif event.type == pygame.KEYDOWN: #нажатие кнопок клавиатуры

if event.key == pygame.K\_RETURN:

input\_text = Plus.NewText(input\_text)

for bort in Plus.borts:

bort.StopChange()

elif event.key == pygame.K\_BACKSPACE:

input\_text = input\_text[:-1]

input\_text = Plus.NewText(input\_text)

else:

try:

input\_text += str(event.unicode)

except:

print()

input\_text = Plus.NewText(input\_text)

#------------end running------------------

pygame.quit()

print(listCB)

print(listParam)

**Plus.py**

import pygame

from const import \*

from bort import Bort

class Plus():

def \_\_init\_\_(self,sc):

self.condition = 1 #кол-во прорисованных бортов

self.screen = sc

self.delta = 91

self.x = 1567

self.y = 50+91

self.h = 50

self.w = 50

self.borts = [Bort(1285,33,1,self.screen)] #массив бортов

self.uncorrect =False #флаг корректности параметров борта

self.listParam=[] #данные всех бортов

self.lastListParam=[] #данные последнего борта

self.errorParam = [False,False,False,False,False,False] #флаги ошибок в корректности параметра True=ошибка

self.oversteck =False #флаг переполнения количества бортов True=переполнение

def Draw(self):

"""Прорисовывыем плюс и все доступные борта"""

if (self.condition<8) and self.oversteck==False:

self.screen.blit(plus, (self.x, self.y)) # плюс

for el in self.borts:

el.Draw()

def Change(self,mosPos):

""" Проверяем нажатие плюса и в случае корректных параметров борта и отсутствие переполнения

добавляем новый борт и отправляем данные на сервер для создания траектории по параметрам борта"""

if self.condition>0:

self.CheckCorrect()

if (self.x <= mosPos[0] <= self.x + self.w and self.y <= mosPos[1] <= self.y + self.h) and self.condition<8 :

#плюс нажат и нет переполнения

if self.uncorrect==False:# or self.condition==0:

#параметры корректны

self.y +=self.delta

if len(self.borts) >= 1: #существует хотябы 1 борт

self.borts[len(self.borts) - 1].UnActive()

if self.condition<7: #не последний борт

self.condition += 1

newBort = Bort(1285,33+self.delta\*(self.condition-1),self.condition,self.screen)

self.borts.append(newBort)

else: #последний борт

self.oversteck = True

for i in range(6): #обнуление флагов ошибки

self.borts[self.condition-2].IBs[i].error = False

if self.lastListParam!=[]: #список не пуст

self.listParam.append(self.lastListParam)

if self.condition != 1: # не нулевой борт (пока без параметров)

return(self.lastListParam)#GoServer(self.lastListParam))

else: #параметры некорректны

for i in range(6):

self.borts[self.condition - 1].IBs[i].changed = False

if self.errorParam[i]:

self.borts[self.condition-1].IBs[i].error = True

else:

self.borts[self.condition-1].IBs[i].error = False

return([[]])

#self.CheckCorrect()

def NewText (self,newText):

"""

Обновляет текст для всех связанных элементов.

Параметры:

newText (str): Новый текст, который нужно установить.

Возвращает:

str: Обновленный текст.

"""

newText = str(self.toInt(newText))[:4]

for bort in self.borts:

for ib in bort.IBs:

ib.NewText(newText)

return newText

def toInt(self,string):

"""

Преобразует строку в целое число.

Параметры:

string (str): Входная строка.

Возвращает:

int: Преобразованное целое число.

"""

nums=[0]

try:

for i in string:

try:

nums.append(int(i))

except:

print()

except:

print()

res = ""

for num in nums:

res = res + str(num)

return int(res)

def CheckCorrect(self):

"""

Проверяет корректность параметров и обновляет флаги ошибок.

Возвращает:

bool: True, если параметры не корректны, False - если нет ошибки.

"""

self.UpdateListParam()

self.uncorrect = False

self.errorParam = [False,False,False,False,False,False]

if self.lastListParam[0] ==0:

self.errorParam[0] = True

if self.lastListParam[1] ==0 or self.lastListParam[1] > pMax:

self.errorParam[1] = True

if self.lastListParam[1] >9 and self.condition == 1:

self.errorParam[1] = True

if self.lastListParam[2] <0:

self.errorParam[2] = True

if self.lastListParam[3] <=self.lastListParam[2]:

self.errorParam[3] = True

if self.lastListParam[4] <0 or self.lastListParam[4] > xMax:

self.errorParam[4] = True

if self.lastListParam[5] <0 or self.lastListParam[5] > yMax:

self.errorParam[5] = True

for i in range(6):

self.uncorrect = self.uncorrect or self.errorParam[i]

def UpdateListParam(self):

"""Обновляет список параметров из связанных элементов"""

self.lastListParam = []

bort = self.borts[self.condition - 1]

for ib in bort.IBs:

self.lastListParam.append(int(ib.inputText))

**Bort.py**

import pygame

from const import \*

from inputbox import InputBox

class Bort():

def \_\_init\_\_(self,x,y,i,sc):

self.condition = i #номер борта в массиве

self.screen = sc

self.x = x

self.y = y

self.w = 60

self.h = 60

self.color = bortsColor

self.textBorts = ["Борт1 (основной)","Борт2","Борт3","Борт4","Борт5","Борт6","Борт7","Борт8"]

self.textParam = ["V =", "P =", "Tн =", "Tк =", "Xн =", "Yн ="]

self.active = True #флаг, True=изменяются параметры, False=параметры обработаны и неизменны

self.fontBold = pygame.font.SysFont(font\_name, 24, bold=True)

self.fontNormal = pygame.font.SysFont(font\_name, 20, bold=True)

self.inputText = "0"

self.IBs = [InputBox(self.x+100,self.y+45,self.screen),InputBox(self.x+190,self.y+45,self.screen),

InputBox(self.x+280,self.y+45,self.screen),InputBox(self.x+370,self.y+45,self.screen),

InputBox(self.x+460,self.y+45,self.screen),InputBox(self.x+550,self.y+45,self.screen)]

#список inputboxs конкретного борта

def Draw(self):

""" Пририсовываем конкретный борт """

pygame.draw.rect(self.screen, mainCollor, (1285, self.y, 596, 81), border\_radius=borderRad) #борт

pygame.draw.circle(self.screen, self.color[self.condition-1], (self.x+30, self.y+40), 15) #цвет борта

self.Surface = self.fontBold.render(self.textBorts[self.condition-1], True, textCollor)

self.Rect = self.Surface.get\_rect(topleft=(self.x + 66, self.y +10), size=(35, 180))

self.screen.blit(self.Surface, self.Rect)

for i in range (len(self.textParam)):

self.SurfaceI = self.fontNormal.render(self.textParam[i], True, textCollor)

self.RectI = self.SurfaceI.get\_rect(topleft=(self.x + 60 + 90\*i, self.y +50), size=(30, 45))

self.screen.blit(self.SurfaceI, self.RectI)

for ib in self.IBs:

ib.Draw() #рисуем inputboxs

def StopChange(self):

""" Сбрасывает состояние inputboxs в не активное """

for ib in self.IBs:

ib.OffChenged()

def Change(self,mosPos):

""" Обновляет данные введенных значений в inputbox """

for ib in self.IBs:

if ib.Change(mosPos):

self.StopChange()

text = ib.OnChenged()

return text

return

def UnActive(self):

""" Останавливает возможность изменение параметров борта """

for ib in self.IBs:

ib.UnActive()

**Inputbox.py**

import pygame

from const import \*

class InputBox():

def \_\_init\_\_(self,x,y,sc):

self.condition = True #флаг доступности IB True=возможно активировать, False=невозможно активировать

self.changed = False #флаг активированного IB True=активный

self.error = False #флаг некорректного значения True=ошибка

self.screen = sc

self.x = x

self.y = y

self.w = 45

self.h = 30

self.backColor = dopCollor

self.textColor = textCollor

self.fontBold = pygame.font.SysFont(font\_name, 24, bold=True)

self.fontNormal = pygame.font.SysFont(font\_name, 20, bold=True)

self.inputText = "0"

def Draw(self):

""" Рисуем IB, его обводку и текст в нем"""

if self.condition:

self.backColor = dopCollor

if self.changed:

self.textColor = activeCollor

else:

self.textColor = textCollor

else:

self.backColor = mainCollor

self.textColor = textCollor

pygame.draw.rect(self.screen, self.backColor, (self.x, self.y, self.w, self.h), border\_radius=borderRad)

if self.error:

pygame.draw.rect(self.screen, errorCollor, (self.x, self.y, self.w, self.h), 3, border\_radius=borderRad)

if self.changed:

pygame.draw.rect(self.screen, borderCollor, (self.x, self.y, self.w, self.h), 3, border\_radius=borderRad)

text\_surface = self.fontNormal.render(self.inputText, True, self.textColor)

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

text\_rect.midleft = (self.x + 5, self.y + self.h // 2)

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

def Change(self,mosPos):

""" При нажатии на IB активируем его и меняем содержимое"""

if self.x <= mosPos[0] <= self.x + self.w and self.y <= mosPos[1] <= self.y + self.h and self.condition:

text = self.OnChenged()

return True

return False

def UnActive(self):

""" Блокируем IB делаем его недоступным """

self.condition = False

self.OffChenged()

def OffChenged(self):

""" Дизактивируем IB """

self.changed = False

def OnChenged(self):

""" Aктивируем IB """

self.changed = True

return self.inputText

def NewText(self,newText):

""" Меняем содержимое IB """

if self.changed:

self.inputText = newText

**Checkbox.py**

import pygame

from const import \*

class ChackBox():

def \_\_init\_\_(self,x,y,sc):

self.condition = True #флаг, True=галочка, False=отсутствие галочки

self.screen = sc

self.x = x

self.y = y

self.w = 60

self.h = 60

def Draw(self):

""" Рисуем CB """

self.screen.blit(chackbox, (self.x, self.y)) # бокc

if self.condition:

self.screen.blit(true, (self.x, self.y)) # галочка

def ChangeCB(self,mosPos):

""" При нажатии меняем значение CB """

if self.x <= mosPos[0] <= self.x + self.w and self.y <= mosPos[1] <= self.y + self.h:

self.condition = not(self.condition)

**Cells.py**

import pygame

from const import \*

class Cells():

def \_\_init\_\_(self,sc):

self.condition = 1 #масштаб клетки (не реализован)

self.screen = sc

self.x = 15

self.y = 18

self.w =1242

self.h = 571

self.fontNormal = pygame.font.SysFont(font\_name, 12, bold=True)

self.textCollor = (230,100,75)

def Draw(self):

""" Рисуем клетку и подписи координат """

self.screen.blit(cels, (self.x, self.y)) # клетка

for i in range(0,self.condition \* self.w, 100):

Surface = self.fontNormal.render(str(i), True, self.textCollor)

Rect = Surface.get\_rect(topleft=(self.x + i - 10, self.y + 10), size=(20, 20))

self.screen.blit(Surface, Rect)

for i in range(0,self.condition \* self.h, 100):

Surface = self.fontNormal.render(str(i), True, self.textCollor)

Rect = Surface.get\_rect(topleft=(self.x + 10, self.y + i - 10), size=(20, 20))

self.screen.blit(Surface, Rect)

#print(i,self.x + i - 10,self.y + self.h - 50)

#def ChangeCB(self,mosPos):

# if self.x <= mosPos[0] <= self.x + self.w and self.y <= mosPos[1] <= self.y + self.h:

# self.condition = not(self.condition)

**Slider.py**

import pygame

from const import \*

class Slider():

def \_\_init\_\_(self,sc):

self.condition = 30 #значение slider

self.screen = sc

self.min = 0

self.max = 100

self.posPoint = 350 #позиция точки ролика

self.x = 89

self.y = 944

self.h = 30

self.w = 1121

self.fontNormal = pygame.font.SysFont(font\_name, 35, bold=True)

def Draw(self):

""" Рисуем слайдер, ролик и значения времени и ось координат для графиков """

pygame.draw.rect(self.screen, dopCollor, (89, 944, 1121, 20),

border\_radius=borderRad) # слайдер

pygame.draw.rect(self.screen, borderCollor, (89, 944, 1121, 20), 5,

border\_radius=borderRad) # слайдер

pygame.draw.circle(self.screen, ellsCollor, (self.posPoint, 954), 20) # точка

pygame.draw.circle(self.screen, borderCollor, (self.posPoint, 954), 20, 2) # точка

self.Surface1 = self.fontNormal.render("0", True, textCollor)

self.Rect1 = self.Surface1.get\_rect(topleft=(self.x-14, self.y - 45), size=(20, 20))

self.screen.blit(self.Surface1, self.Rect1)

self.Surface2 = self.fontNormal.render(str(self.max), True, textCollor)

self.Rect2 = self.Surface2.get\_rect(topleft=(self.x + self.w - 25, self.y - 45), size=(50, 20))

self.screen.blit(self.Surface2, self.Rect2)

self.Surface3 = self.fontNormal.render(str(self.condition), True, textCollor)

self.Rect3 = self.Surface3.get\_rect(topleft=(self.posPoint -15, self.y + 25), size=(50, 20))

self.screen.blit(self.Surface3, self.Rect3)

"""Ось координат графика"""

#0Y

pygame.draw.line(self.screen, borderCollor, (self.x, 610),

(self.x, 880), 5)

pygame.draw.line(self.screen, borderCollor, (self.x, 600),

(self.x-10, 610), 5)

pygame.draw.line(self.screen, borderCollor, (self.x, 600),

(self.x+10, 610), 5)

#подпись оси y

self.Surface4 = self.fontNormal.render(str(1000), True, textCollor)

self.Rect4 = self.Surface4.get\_rect(topleft=(self.x - 75, 610), size=(50, 20))

self.screen.blit(self.Surface4, self.Rect4)

pygame.draw.line(self.screen, borderCollor, (self.x-10, 630),

(self.x+10, 630), 5)

#0X

pygame.draw.line(self.screen, borderCollor, (self.x, 880),

(self.x + self.w+20, 880), 5)

pygame.draw.line(self.screen, borderCollor, (self.x + self.w +30, 880),

(self.x + self.w+20, 870), 5)

pygame.draw.line(self.screen, borderCollor, (self.x + self.w +30, 880),

(self.x + self.w+20, 890), 5)

pygame.draw.line(self.screen, borderCollor, (self.x+self.w, 880+10),

(self.x+self.w, 880-10), 5)

def Change(self,mosPos):

""" При нажатии на слайдер меняем положение ролика и значение состояния времени """

if self.x <= mosPos[0] <= self.x + self.w and self.y <= mosPos[1] <= self.y + self.h:

self.posPoint = mosPos[0]

self.condition = int((self.posPoint-79)\*self.max/self.w)

#print("T:",self.condition)

def ImportMaxT(self,T):

""" Обновляем максимальное значение времени """

self.max = T

**Consts.py**

import os

import pygame

import sys

import math

HOST = '192.168.0.106' # IP-адрес сервера

PORT = 65432 # Порт для подключения

# Основные цвета

mainCollor = (219, 235, 252) # Основной цвет, вероятно, для фона или основных элементов

dopCollor = (192, 211, 231) # Дополнительный цвет, возможно, для дополнительных элементов или подсветки

dopCollorAlfa = (192, 211, 231,200) # Дополнительный цвет, возможно, для дополнительных элементов или подсветки

borderCollor = (174, 193, 213) # Цвет для границ или разделителей

errorCollor = (250, 50, 50) # Цвет для ошибок или проблемных состояний

textCollor = (73, 81, 90) # Цвет для текстовых элементов

activeCollor = (20, 40, 150) # Цвет для активных или выбранных состояний

ellsCollor = (120, 178, 252) # Цвет для некоторых специфических элементов, возможно, связанных с эллипсами или другими фигурами

# Список для хранения цветов авиа-бортов

bortsColor = [(255,0,0),(255,255,0),(0,255,0),(0,0,255),(0,255,255),(255,0,255),(255,100,50),(50,100,150)]

# Максимальные координаты

xMax = 1237

yMax = 563

pMax = 9

# Параметры связи

conectEps = 10 # Максимальное расстояние между двумя точками, чтобы считать их связанными

# Параметры отображения

FPS = 30 # Целевая частота кадров в секунду

# Другие параметры

openT = 100 # Начальное или стандартное "открытое время" для некоторой функциональности

stepCord = 1 # Шаг изменения координат

# Размеры окна

screen\_width = 1920

screen\_height = 1080

# Параметры линий

lineWidht = 5 # Ширина линий

lineGap = 5 # Расстояние между штрихами линий

lineDash = 5 # Длина штрихов линий

# Параметры границ

borderRad = 10 # Радиус скругления границ

# Получаем путь к текущей директории

current\_dir = os.path.dirname(\_\_file\_\_)

# Указываем путь к папке media относительно текущей директории

media\_dir = os.path.join(current\_dir, 'media')

#BASE = 'C:/Users/vovap/PycharmProjects/practic/media/'

# Загрузка медиафайлов

map = pygame.image.load(os.path.join(media\_dir, "map.png"))

cels = pygame.image.load(os.path.join(media\_dir, "map (1).png"))

plus = pygame.image.load(os.path.join(media\_dir, "plus.png"))

plain = pygame.image.load(os.path.join(media\_dir, "plane (1).png"))

main = pygame.image.load(os.path.join(media\_dir, "main.png"))

conect = pygame.image.load(os.path.join(media\_dir, "conect.png"))

chackbox = pygame.image.load(os.path.join(media\_dir, "CheckboxFalse.png"))

true = pygame.image.load(os.path.join(media\_dir, "True.png"))

# Шрифт по умолчанию

font\_name = "Arial"

def DashedLine(screen, start\_pos, end\_pos, line\_color, line\_width=2, dash\_length=10, gap\_length=5):

"""

Рисует пунктирную линию на заданном экране.

Параметры:

screen (pygame.Surface): Экран, на котором будет нарисована линия.

start\_pos (tuple): Координаты начальной точки линии (x, y).

end\_pos (tuple): Координаты конечной точки линии (x, y).

line\_color (tuple): Цвет линии в формате RGB (r, g, b).

line\_width (int): Ширина линии (по умолчанию 2).

dash\_length (int): Длина штриха (по умолчанию 10).

gap\_length (int): Длина промежутка между штрихами (по умолчанию 5).

"""

x1, y1 = start\_pos

x2, y2 = end\_pos

dx = x2 - x1

dy = y2 - y1

distance = max(abs(dx), abs(dy))

try:

dx = dx / distance

dy = dy / distance

except:

dx=0

dy=0

x, y = start\_pos

count = 0

while distance >= 0.5:

if count % 2 == 0:

pygame.draw.line(screen, line\_color, (x, y), (x + dx \* dash\_length, y + dy \* dash\_length), line\_width)

x += dx \* (dash\_length + gap\_length)

y += dy \* (dash\_length + gap\_length)

distance -= dash\_length + gap\_length

count += 1

def InterpolatePoints(mas,step,mT):

"""

Интерполирует точки между соседними элементами в списке `mas`.

Параметры:

mas (list of list of float): Входной список, содержащий элементы вида `[t, x, y]`, где `t` - время, `x` и `y` - координаты.

step (int): Шаг интерполяции. Определяет, через сколько единиц времени будут вычисляться промежуточные точки.

mT (float): Максимальное время, до которого нужно интерполировать точки.

Возвращает:

list of list of float: Новый список, в котором каждый элемент содержит интерполированные значения `[t, x, y]`.

"""

masRes = []

for i in range(len(mas) - 1):

t1 = mas[i][0]

x1, y1 = mas[i][1:]

t2 = mas[i+1][0]

x2, y2 = mas[i+1][1:]

if t1!=t2:

num\_steps = (t2 - t1) // step

for j in range(num\_steps + 1):

t = t1 + j \* step

x = x1 + int((x2 - x1) \* j / num\_steps)

y = y1 + int((y2 - y1) \* j / num\_steps)

masRes.append([t, x, y])

if i == len(mas) - 2 and mas[i+1][0]<mT:

for j in range (mas[i+1][0]+step,mT+1,step):

masRes.append([j, x, y])

return masRes

def FindMaxInColumn(matrix, column\_index):

"""

Находит максимальное значение в указанном столбце двумерного массива.

Параметры:

matrix (list of lists): Двумерный массив, в котором нужно найти максимальное значение.

column\_index (int): Индекс столбца, в котором нужно найти максимальное значение.

Возвращает:

int: Максимальное значение в указанном столбце.

"""

max\_value = matrix[0][column\_index]

for row in matrix:

if row[column\_index] > max\_value:

max\_value = row[column\_index]

return max\_value

def ChackConect(list):

"""

Находит точки пересечения между соседними элементами в списке `list`.

Параметры:

list (list of list of list of float): Входной список, содержащий вложенные списки вида `[[t, x, y], [t, x, y], ...]`.

Возвращает:

list of list of float: Новый список, в котором каждый элемент содержит время `t`, координаты `x` и `y` точки пересечения, а также расстояние `d` между точками.

"""

res=[]

i=0

for j in range(i+1,len(list)):

res.append([])

for k in range(len(list[i])):

for m in range(len(list[j])):

if list[i][k][0]==list[j][m][0]:

x1 = list[i][k][1]

y1 = list[i][k][2]

x2 = list[j][m][1]

y2 = list[j][m][2]

d = math.sqrt((x2 - x1)\*\*2 + (y2 - y1)\*\*2)

res[j-1].append([list[i][k][0],list[i][k][1],list[i][k][2],d])

#if list[i][k] == list[j][m]:

# res.append(list[i][k])

#print(res)

return res

def draw\_rect\_alpha(surface, color, rect):

shape\_surf = pygame.Surface(pygame.Rect(rect).size, pygame.SRCALPHA)

pygame.draw.rect(shape\_surf, color, shape\_surf.get\_rect(),border\_radius=borderRad)

surface.blit(shape\_surf, rect)

**GoToServer.py**

import pygame

import socket

import ast

import json

from const import \*

def GoServer(params,cordFirst = []):

"""

Отправляет данные на сервер, получает ответ и корректирует координаты.

Параметры:

params (list of float): Список параметров, включающий `V`, `P`, `T1`, `T2`, `X1`, `Y1`.

Возвращает:

list of list of float: Скорректированные координаты в виде списка вложенных списков `[[t, x, y], [t, x, y], ...]`.

"""

cord = SendDataToServer(params,cordFirst)

for i in range(len(cord)):

cord[i][1] = 20 + cord[i][1] # min(max(20,el[1]+20), 1240)

cord[i][2] = 23 + cord[i][2] # min(max(23,el[1]+23), 570)

return cord

def SendDataToServer(params,cordFirst):

"""

Отправляет данные на сервер и получает ответ.

Параметры:

params (list of float): параменты iго самолета

V (float): Значение V.

P (float): Значение P.

T1 (float): Значение T1.

T2 (float): Значение T2.

X1 (float): Значение X1.

Y1 (float): Значение Y1.

cordFirst (list of list of float): Значение координат главного самолета.

Возвращает:

list of list of float: Ответ сервера в виде списка вложенных списков `[[t, x, y], [t, x, y], ...]`.

"""

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

HOST = GetLocalIp()

print(HOST, PORT)

s.connect((HOST, PORT))

data = [params,cordFirst]

#data = f"{V},{P},{T1},{T2},{X1},{Y1}"

#s.sendall(data.encode())

try:

s.sendall(json.dumps(data).encode())

print(f"Отправлено: {data}")

except:

print("Ошибка передачи на сервер")

dataRes = s.recv(1024)

#result = ast.literal\_eval(dataRes.decode())

#dataRes =b''

#while True:

# chunk = s.recv(1024)

# if not chunk:

# break

# dataRes += chunk

try:

result = []

response = json.loads(dataRes.decode())

for item in response:

result.append([item[0], item[1], item[2]])

print(f"Получено: {result}")

except (json.JSONDecodeError, ValueError) as e:

print(f"Ошибка при обработке ответа: {e}")

#print(f"Получено: {', '.join(map(str, result))}")

return(result)

def GetLocalIp():

"""

Получает локальный IP-адрес компьютера.

Возвращает:

str: Локальный IP-адрес компьютера.

"""

hostname = socket.gethostname()

local\_ip = socket.gethostbyname(hostname)

return local\_ip

**Server.py**

import socket

import math

import json

HOST = '192.168.0.106' # IP-адрес сервера

PORT = 65432 # Порт для подключения

def handle\_client\_request(conn, addr):

print(f"Подключение от {addr}")

params=[]

# Получение данных от клиента

data = conn.recv(1024)

try:

response = json.loads(data.decode())

params = response[0]

cordsFirst = response[1]

print(f"Получено: {params} и {cordsFirst}")

except (json.JSONDecodeError, ValueError) as e:

print(f"Ошибка при обработке ответа: {e}")

#print("Получено:", data)

#params = [int(x) for x in data.decode().split(',')]

# Формирование ответа

cord = [[0,0,0],[100,100,100]]

if int(params[1]) == 1:

cord = [[int(params[2]), int(params[4]), int(params[5])],

[int(params[3]), int(params[4]),

int(params[5]) - int(params[0]) \* (int(params[3]) - int(params[2]))]]

if int(params[1]) == 2:

cord = [[int(params[2]), int(params[4]), int(params[5])],

[int(params[3]), int(params[4]) + math.sqrt(((int(params[0])\*\*2)/2)) \* (int(params[3]) - int(params[2])),

int(params[5]) - math.sqrt(((int(params[0])\*\*2)/2)) \* (int(params[3]) - int(params[2]))]]

if int(params[1]) == 3:

cord = [[int(params[2]), int(params[4]), int(params[5])],

[int(params[3]), int(params[4]) + int(params[0])\* (int(params[3]) - int(params[2])),

int(params[5])]]

if int(params[1]) == 4:

cord = [[int(params[2]), int(params[4]), int(params[5])],

[int(params[3]), int(params[4]) + math.sqrt(((int(params[0])\*\*2)/2)) \* (int(params[3]) - int(params[2])),

int(params[5]) + math.sqrt(((int(params[0])\*\*2)/2)) \* (int(params[3]) - int(params[2]))]]

if int(params[1]) == 5:

cord = [[int(params[2]), int(params[4]), int(params[5])],

[int(params[3]), int(params[4]),

int(params[5]) + int(params[0]) \* (int(params[3]) - int(params[2]))]]

if int(params[1]) == 6:

cord = [[int(params[2]), int(params[4]), int(params[5])],

[int(params[3]), int(params[4]) - math.sqrt(((int(params[0])\*\*2)/2)) \* (int(params[3]) - int(params[2])),

int(params[5]) + math.sqrt(((int(params[0])\*\*2)/2)) \* (int(params[3]) - int(params[2]))]]

if int(params[1]) == 7:

cord = [[int(params[2]), int(params[4]), int(params[5])],

[int(params[3]), int(params[4]) - (int(params[0])) \* (int(params[3]) - int(params[2])),

int(params[5])]]

if int(params[1]) == 8:

cord = [[int(params[2]), int(params[4]), int(params[5])],

[int(params[3]), int(params[4]) - math.sqrt(((int(params[0])\*\*2)/2)) \* (int(params[3]) - int(params[2])),

int(params[5]) - math.sqrt(((int(params[0])\*\*2)/2)) \* (int(params[3]) - int(params[2]))]]

#response = str(cord).replace('[', '').replace(']', '').replace('\'', '')

response = cord

# Отправка ответа клиенту

#conn.sendall(response.encode())

conn.sendall(json.dumps(response).encode())

print(f"Отправлено: {response}")

def start\_server():

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

try:

HOST = get\_local\_ip()

s.bind((HOST, PORT))

s.listen()

print(f"Сервер запущен и ожидает подключений на {HOST}:{PORT}")

while True:

conn, addr = s.accept()

handle\_client\_request(conn, addr)

except OSError as e:

print(f"Ошибка: {e}")

def get\_local\_ip():

hostname = socket.gethostname()

local\_ip = socket.gethostbyname(hostname)

return local\_ip

if \_\_name\_\_ == "\_\_main\_\_":

start\_server()

|  |  |  |
| --- | --- | --- |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | /\_\_\_\_Поцепай В.Н.\_\_\_/ | \_\_\_ \_\_\_\_\_\_ 2024г. |
| *подпись обучающегося* | *расшифровка подписи* | *дата* |