Министерство образования Республики Беларусь
Учреждение образования “Полоцкий государственный университет”

Факультет информационных технологий
Кафедра технологий программирования

Майнор «Технологии интернета вещей»
Отчет

Выполнил — студент гр. 21-ИТ-1, ФИТ
Ланцев Е. Н.

Проверил — ассистент кафедры ТП
Сергеев М. А.

Полоцк,
2023г.

MQTT ПРОЕКТ

Цель работы: разработать практический проект в сфере интернета вещей с использованием протокола MQTT и языка программирования Python.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Протокол MQTT (Message Queuing Telemetry Transport) представляет собой механизм обмена сообщениями, основанный на концепции "издатель-подписчик" (pub/sub). Этот протокол был впервые представлен в 1999 году Энди Стэнфорд-Кларком из IBM и Арленом Ниппером из Cirrus Link. Их целью было создание эффективного средства обмена данными между устройствами в сетях с ограниченной пропускной способностью или непредсказуемой связью.

MQTT призван обеспечить надежную связь в условиях, где обычные методы могут оказаться недостаточно эффективными. Это особенно важно в контексте сетей с переменной пропускной способностью или сетей, подверженных непостоянным условиям связи. Сначала протокол был применен для обеспечения связи между фрагментами нефтепровода и центральными звеньями с использованием спутников.

Одной из ключевых особенностей MQTT является его легковесность и минимальное потребление ресурсов, что делает его идеальным выбором для встроенных систем и устройств с ограниченными ресурсами. Принцип работы протокола "издатель-подписчик" позволяет эффективно передавать данные от издателя (устройства, инициирующего передачу) к подписчику (устройству, ожидающему получения данных) без необходимости установления прямого соединения между ними.[1]

Графическое представление функционирования протокола MQTT представлено на рисунке 1.

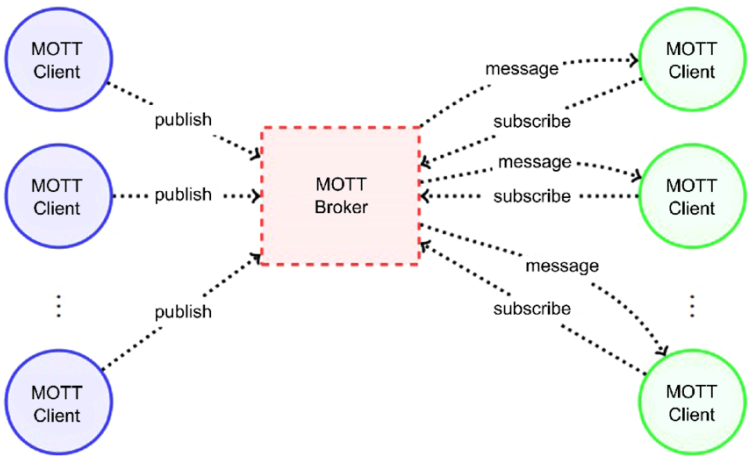

Рисунок 1 – Графический пример работы протокола MQTT [5]

Протокол MQTT представляет собой логически обоснованный выбор для разработчиков, стремящихся создать приложения с высокой степенью надежности и обширной совместимостью с подключенными к интернету устройствами и приложениями. Этот протокол особенно актуален в контексте разнообразных устройств, таких как браузеры, смартфоны и устройства Интернета вещей (IoT).

MQTT предоставляет эффективный механизм обмена данными между устройствами в распределенных средах, где требуется надежная и мгновенная передача информации. Его принцип "издатель-подписчик" обеспечивает устойчивую связь между различными устройствами, даже при изменяющихся условиях сети.

Этот протокол отлично подходит для приложений, ориентированных на взаимодействие с устройствами IoT, поскольку он позволяет эффективно передавать данные между узлами сети с минимальными затратами ресурсов. Благодаря своей легковесной структуре и оптимизированному процессу обмена сообщениями, MQTT обеспечивает отзывчивость и эффективность даже в условиях с ограниченной пропускной способностью.

Протокол MQTT, использующий модель «издатель-подписчик», представляет собой эффективный механизм обмена сообщениями в реальном времени. В контексте меж машинной связи, где необходимо обеспечить надежную передачу данных, MQTT выделяется среди прочих протоколов, таких как Web Application Messaging Protocol, Streaming Text-Oriented Messaging Protocol и Alternative Message Queueing Protocol.

В традиционной модели взаимодействия по сети клиенты и серверы соединяются напрямую, где клиенты запрашивают у сервера ресурсы, и сервер обрабатывает запрос, возвращая ответ. Однако MQTT переносит этот подход, применяя шаблон «издатель-подписчик». Здесь отправитель сообщения (издатель) и получатель (подписчик) разделены, и третий компонент, брокер сообщений, управляет взаимодействием между ними. [2]

Задачей брокера является фильтрация входящих сообщений от издателей и передача их соответствующим подписчикам. Этот механизм разделения достигается следующим образом:

1) Разделение в пространстве: Издатель и подписчик не обладают информацией о местоположении друг друга в сети и не обмениваются конфиденциальными данными, такими как IP-адрес и номер порта;

2) Разделение во времени: Издатель и подписчик не действуют и не подключаются к сети одновременно, что обеспечивает последовательность взаимодействия.

3) Раздельная синхронизация: Издатели и подписчики могут отправлять и получать сообщения независимо друг от друга, без необходимости ожидания ответа. Например, подписчик не требуется ожидать завершения отправки сообщения от издателя.

Такой подход не только повышает эффективность передачи данных, но и обеспечивает гибкость и масштабируемость в системах, где требуется управление потоком информации между устройствами.

Основные принципы протокола:

1) Модель "Издатель-Подписчик" (Pub/Sub): Протокол MQTT работает по принципу модели "издатель-подписчик", где устройства (издатели) публикуют сообщения, а другие устройства (подписчики) получают эти сообщения. Это

обеспечивает распределенную систему обмена данными, где устройства не обязательно должны знать друг о друге;

2) Брокер сообщений: Центральным элементом в архитектуре MQTT является брокер сообщений. Брокер принимает сообщения от издателей и маршрутизирует их к соответствующим подписчикам. Это позволяет изолировать устройства друг от друга, создавая более гибкую и масштабируемую среду обмена данными;

3) Разделение в пространстве: Издатели и подписчики в MQTT не взаимодействуют напрямую. Они разделены в пространстве, не зная о местоположении друг друга и не обмениваются информацией о сетевых адресах;

4) Разделение во времени: Издатели и подписчики не подключаются к сети одновременно. Это обеспечивает управление временными аспектами взаимодействия и позволяет более гибко управлять потоком данных;

5) Сохранение состояния сессии: MQTT поддерживает сохранение состояния сессии, что позволяет восстанавливать подписки и сообщения после переподключения клиента к брокеру. Это важно для обеспечения надежности и целостности передачи данных;

6) Легковесность и эффективность: Протокол разработан с учетом ограниченных ресурсов устройств, поэтому он легковесен и эффективен. Это особенно важно для применения в сценариях Интернета вещей (IoT) и других встроенных системах;

7) QoS (Quality of Service): MQTT поддерживает уровни обслуживания качества, которые позволяют определить уровень надежности доставки сообщений от издателя к подписчику [3]. Это включает в себя гарантированную доставку (QoS 1) и доставку с подтверждением (QoS 2);

Применение в различных областях:

1) Интернет вещей (IoT): MQTT является одним из ключевых протоколов в сфере IoT. Он обеспечивает надежную и эффективную связь между миллионами устройств, такими как датчики, умные устройства, и системы управления, что позволяет создавать умные города, умные дома и промышленные IoT-решения;

2) Мониторинг и управление зданиями (Building Automation): В системах умного дома и здания MQTT применяется для передачи данных о состоянии и управлении системами отопления, вентиляции, кондиционирования воздуха (HVAC), освещением и безопасностью;

3) Промышленная автоматизация и управление производством: MQTT используется в промышленности для передачи данных о состоянии оборудования, контроля качества, мониторинга производственных процессов и координации между различными устройствами;

4) Транспорт и логистика: В области транспорта MQTT применяется для мониторинга и отслеживания транспортных средств, управления логистикой, отслеживания грузов и обеспечения коммуникации в рамках цепочек поставок;

5) Здравоохранение: В медицинских системах MQTT используется для мониторинга пациентов, передачи данных от медицинского оборудования, а также для обмена информацией между различными медицинскими устройствами и системами;

6) Фермерство и сельское хозяйство (Precision Agriculture): MQTT применяется в сельском хозяйстве для мониторинга условий посевов, управления

системами полива, отслеживания сельскохозяйственной техники и обмена данными о погоде;

7) Финансовые услуги: В финансовых системах MQTT может использоваться для обмена данными между банками, финансовыми учреждениями и терминалами обслуживания клиентов;

8) Телекоммуникации: MQTT применяется в телекоммуникационных системах для мониторинга сетевого оборудования, управления трафиком, и обеспечения коммуникации между узлами сети;

Преимущества и недостатки данного протокола:

1) Преимущества: MQTT выделяется среди протоколов обмена сообщениями благодаря своей легковесности, модели "издатель-подписчик", использованию брокера сообщений, гарантированной доставке, возможности сохранения состояния сессии и простоте интеграции, что обеспечивает эффективное и надежное взаимодействие между устройствами в различных областях.

2) Недостатки: отсутствие встроенной защиты (например, шифрования) в стандартной реализации и потребность внимательного управления безопасностью для предотвращения угроз, таких как несанкционированный доступ и перехват данных.

Выполнение работы

Для реализации конкретного проекта, использующего протокол MQTT и язык программирования Python, была выбрана тема "Мультиплеерная игра Пинг-Понг". Проект основан на клиент-серверной архитектуре, где сервер выступает в роли главного игрока, а клиент - второго игрока. Коммуникация между клиентом и сервером осуществляется посредством протокола MQTT, обеспечивая эффективный обмен данными в реальном времени.

Данное решение представляет собой пример приложения, использующего MQTT для обеспечения взаимодействия между разными компонентами, что является типичным сценарием в сфере разработки многопользовательских игр и других распределенных приложений. Применение протокола MQTT позволяет реализовать асинхронное взаимодействие между клиентом и сервером, что особенно важно для игровых приложений, где требуется минимизация задержек и обеспечение плавного игрового процесса.

Архитектурное разделение на клиентскую и серверную части обеспечивает модульность и масштабируемость проекта. Клиент и сервер, оба написанные на Python, взаимодействуют посредством MQTT, что способствует согласованности и эффективной передаче игровых данных. Важно отметить, что выбор языка программирования Python и протокола MQTT в данном контексте обусловлен их широкой поддержкой и применимостью в сфере разработки игр и распределенных приложений.

Для разработки приложения клиента и сервера мною были использованы следующие библиотеки:

1) Paho-mqtt (библиотека для работы с протоколом MQTT).
2) Asyncio_mqtt (поддержка асинхронной реализации протокола MQTT)
3) Pygame (библиотека для разработки кросс платформенных графических приложений)

Для реализации проекта мною был разработан код, использующий протокол MQTT для подключения к брокеру и подписи на топик. Для подключения создается экземпляр клиента, который уже и подключается к брокеру и подписывается на топик, а так же проверку на ключевое слово, для отправки данных о нажатых клавишах серверу.

Листинг 1 – реализация передачи через протокол MQTT

```
async def send_player_speed(player_speed):
    async with Client(mqtt_hub) as client:
     await client.publish("minor/pong/player2_speed", player_speed)

async def send_player_pos(player_pos):
    print("send player pos: " + str(player_pos))
    async with Client(mqtt_hub) as client:
        await client.publish("minor/pong/player2_pos", player_pos)

 async def send_all():
     while True:
         await send_player_speed(player_speed)
         await send_player_pos(player2.y)
 thread = threading.Thread(target=asyncio.run, args=(send_all(),))
 thread.start()
```

В строчках кода 1 – 8 реализованы асинхронные функции отправки позиции и скорости игрока на сервер.

В строке 10 – 13 реализована функция вызова асинхронных функций отправки данных игрока на сервер.

В строках 14 – 15 создается отдельный поток отправки данных серверу для достижения быстрого отклика приложения.

Затем был реализован метод, который вызывается при получении данных с сервера. Данный метод также работает в отдельном потоке для уменьшения прерываний основного потока ввода/вывода. Данные, полученные с сервера проверяются на ключевые слова и присваиваются переменным для их последующего отображения в окне графического представления состояния игры. Пример кода представлен в листинге 2. Пример вывода информации представлен на рисунке 2.

Листинг 2 – получение данных с сервера и их присвоение состоянию клиента

```
1: client = mqtt.Client()
2: client.on_connect = on_connect
3: client.on_message = on_message
4:
5: def on_message(client, userdata, msg):
6:     if msg.topic == "minor/pong/ball_speed_x":
7:         global ball_speed_x
8:         ball_speed_x = int(msg.payload)
9:     if msg.topic == "minor/pong/ball_speed_y":
10:         global ball_speed_y
11:         ball_speed_y = int(msg.payload)
12:    if msg.topic == "minor/pong/ball_pos_x":
13:         ball.x = int(msg.payload)
14:    if msg.topic == "minor/pong/ball_pos_y":
15:         ball.y = int(msg.payload)
16:    if msg.topic == "minor/pong/player1_speed":
17:         global player1_speed
18:    if msg.topic == "minor/pong/player1_pos":
19:         player1.y = int(msg.payload)
20:
21:  pygame.draw.rect(screen, light_gray, player1)
22:  pygame.draw.rect(screen, light_gray, player2)
23:  pygame.draw.ellipse(screen, light_gray, ball)
24:  pygame.draw.aaline(screen, light_gray, (screen_width / 2,
25:                          0), (screen_width / 2, screen_height))
```

В строках 1 – 3 происходит подпись на события при подключении клиента к топику и при получении сообщения.

В строке 5 – 21 объявлена функция которая вызывается при получении сообщения. Данная функция играет главную роль в поддержании единого представления об состоянии игры у клиента и сервера.

В строках 6, 9, 12, 14, 16, 18 происходит проверка на соотношение топика с реплицированными данными.

В строках 21 – 25 происходит отображение в графическом приложении реплицированных данных с помощью Pygame API.

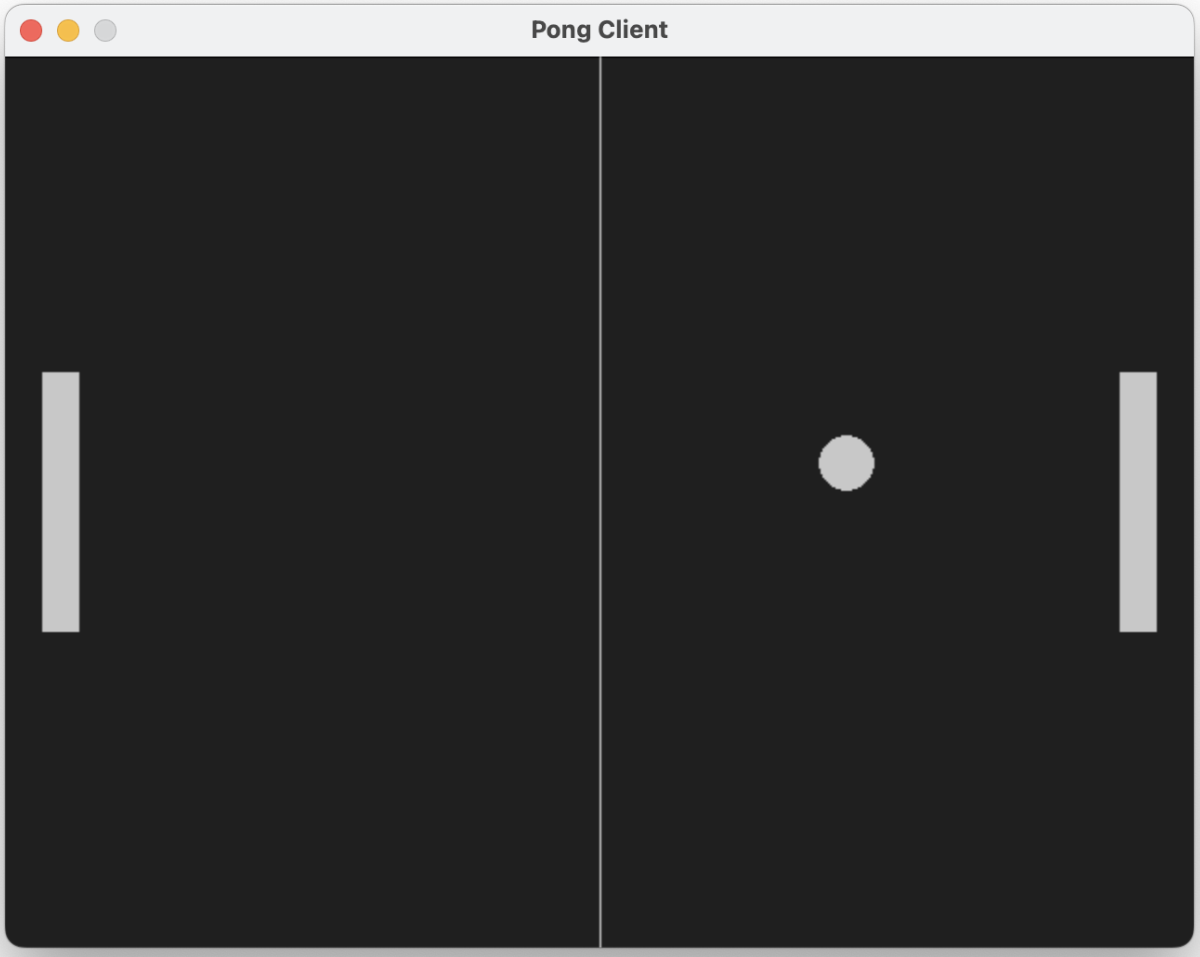

Рисунок 2 – Пример вывода окна клиента

Сервер в проекте являются центральной частью вычисления игровой логики. Клиент не имеет управления над позицией игровых объектов, вместо этого он лишь может управлять направлением движения своего игрока. В свою очередь сервер владеет всеми объектами в мире и управляет ими, реплицируя на остальных клиентов. Данный подход позволяет уменьшить шанс взлома приложения и разработки для него вредоносного программного обеспечения для получения преимущества над другими игроками.

Листинг 3 – отправка данных с сервера клиентам

```
async def send_all():
    while True:
        await send_ball_speed_x(ball_speed_x)
        await send_ball_speed_y(ball_speed_y)
        await send_ball_pos_x(ball.x)
        await send_ball_pos_y(ball.y)
        await send_player_speed(player_speed)
        await send_player_pos(player1.y)

async def send_ball_speed_x(ball_speed_x):
    async with Client(mqtt_hub) as client:
        await client.publish("minor/jejikeh/pong/ball_speed_x",
ball_speed_x)
```

Продолжение листинга 3 – отправка данных с сервера клиентам

```
13.  async def send_ball_pos_x(ball_x):
14.      async with Client(mqtt_hub) as client:
15.          await client.publish("minor/pong/ball_pos_x", ball_x)
16.
17.
18.  async def send_ball_pos_y(ball_y):
19.      async with Client(mqtt_hub) as client:
20.          await client.publish("minor/pong/ball_pos_y", ball_y)
21.
22.
23.  async def send_ball_speed_y(ball_speed_y):
24.      async with Client(mqtt_hub) as client:
25.                   await  client.publish("minor/pong/ball_speed_y",
  ball_speed_y)
26.
27.
28.  async def send_player_speed(player_speed):
29.      async with Client(mqtt_hub) as client:
30.                  await  client.publish("minor/pong/player1_speed",
  player_speed)
31.
32.
33.  async def send_player_pos(player_pos):
34.      print("send player pos: " + str(player_pos))
35.      async with Client(mqtt_hub) as client:
36.                    await  client.publish("minor/pong/player1_pos",
  player_pos)
37.
38.
39.
40.  thread = threading.Thread(target=asyncio.run, args=(send_all(),))
41.  thread.start()
```

В строке 1 определяется асинхронный метод с бесконечным циклом который отправляет все данные на MQTT сервер в определенные топики.

В строках 10 – 36 объявлены асинхронные функции которые отправляют данные клиенту. Каждая из этих функций отвечает за свой собственное ключевое слово и запукается параллельно относительно других функций благодаря возможностям асинхронного программирования в языке Python и библиотеки Asyncio.

В строках 40 – 41 создается новый поток для отправки сообщений клиенту для предотвращения прерываний в главном потоке, что может привести к потерям данных

и рассинхрону клиента и сервера.

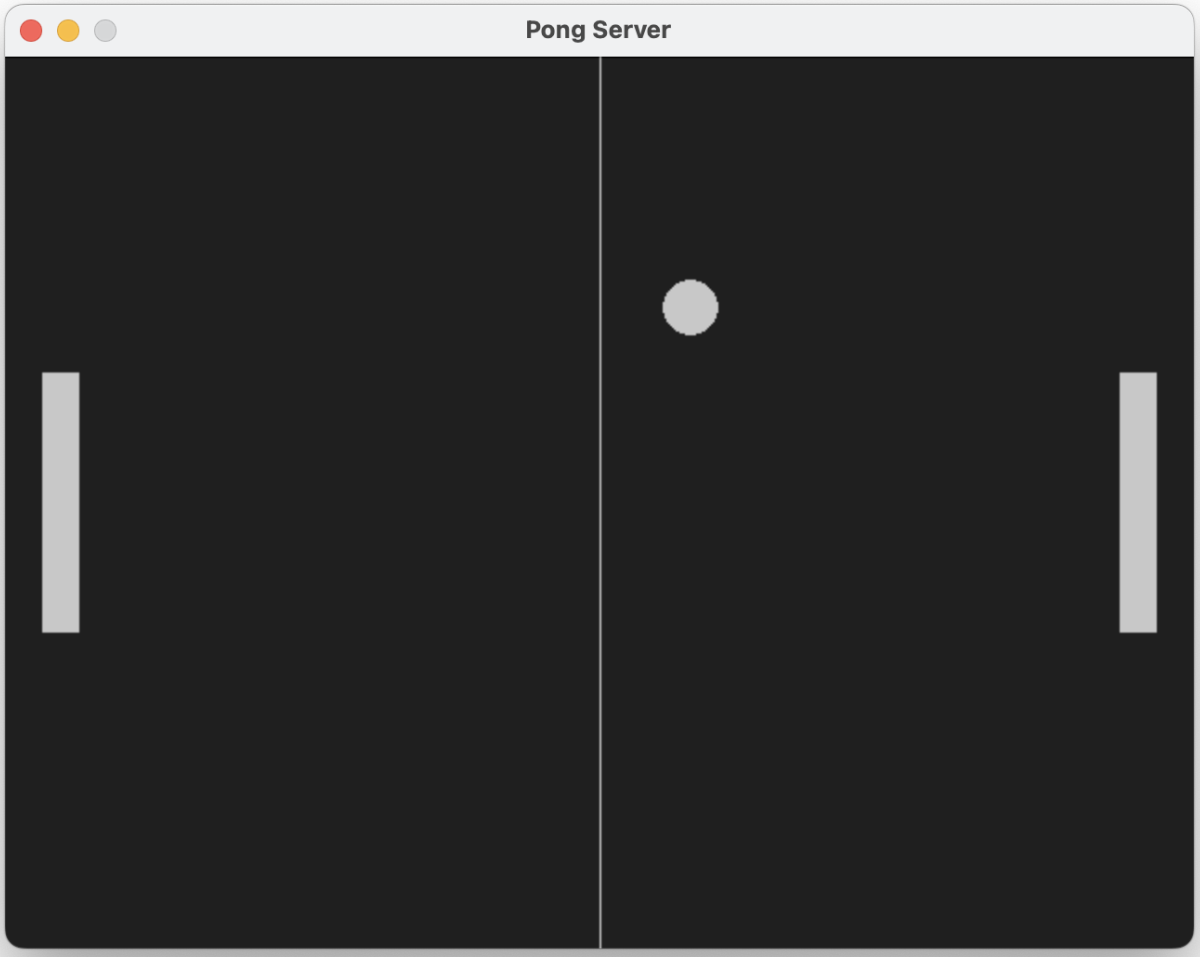

Рисунок 3 – пример окна сервера

Визуально клиент и сервер не отличаются, однако вся игровая логика считается на стороне сервера для последующей репликации клиентам. Игровая логика включает в себя перемещение игроков и игрового мяча, обработка коллизий. Пример кода представлен в листинге 4.

Листинг 4 – игровая логика реализованная на сервере

```
screen_width = 640
screen_height = 480

global ball_speed_x
ball_speed_x = 7

global ball_speed_y
ball_speed_y = 7

 player_speed = 0
 sync_time = 100

 screen = pygame.display.set_mode((screen_width, screen_height))
 pygame.display.set_caption('Pong Server')

```

Продолжение листинга 4 – игровая логика реализованная на сервере

```
16:  ball = pygame.Rect(screen_width/2 - 15, screen_height/2 - 15, 30,
   30)
17:  player1 = pygame.Rect(screen_width - 40, screen_height/2 - 70,
   20, 140)
18:  player2 = pygame.Rect(20, screen_height/2 - 70, 20, 140)
19:
20:  bg_color = pygame.Color('grey12')
21:  light_gray = (200, 200, 200)
22:
23:  pygame.init()
24:  clock = pygame.time.Clock()
25:
26:  iframe = 0
27:
28:  ball.x += ball_speed_x
29:  ball.y += ball_speed_y
30:  player1.y += player_speed
31:  if ball.top <= 0 or ball.bottom >= screen_height:
32:      ball_speed_y *= -1
33:  if ball.left <= 0 or ball.right >= screen_width:
34:      ball_speed_x *= -1
35:  if ball.colliderect(player1) or ball.colliderect(player2):
36:      ball_speed_x *= -1
37:
38:  for event in pygame.event.get():
39:          if event.type == pygame.QUIT:
40:              pygame.quit()
41:              sys.exit()
42:          if event.type == pygame.KEYDOWN:
43:              if event.key == pygame.K_DOWN:
44:                  player_speed += 7
45:              if event.key == pygame.K_UP:
46:                  player_speed -= 7
47:
48:          if event.type == pygame.KEYUP:
49:              if event.key == pygame.K_DOWN:
50:                  player_speed -= 7
51:
52:              if event.key == pygame.K_UP:
53:                  player_speed += 7
```

В строках 1 – 2 устанавливаются ширина и высота игрового экрана.

В строках 4 – 8 устанавливаются начальные горизонтальная и вертикальная скорости мяча.

В строках 13 – 14 создается игровое окно с использованием библиотеки Pygame и устанавливает заголовок окна.

В строках 16 – 18 создаются объекты игры, такие как мяч и две ракетки для игроков.

В строках 23 – 24 запускается Pygame и создается объект часов для управления временем в игре.

В строках 26 – 36 обновляет позиции мяча и игрока, а также проверяет столкновения мяча с верхней, нижней, левой и правой границами экрана, а также с ракетками игроков.

В строках 38 – 53 обрабатываются события Pygame, такие как нажатие и отпускание клавиш. Например, изменение скорости движения игрока при нажатии клавиш вверх и вниз.

Листинг 4 представляет собой основные элементы игровой логики сервера для мультиплеерной игры Pong. Он определяет параметры игрового окна, объекты, и обрабатывает события для управления игровыми элементами.

ВЫВОД

Использование протокола MQTT предоставляет эффективный механизм обмена данными между сервером и клиентами, обеспечивая асинхронное взаимодействие и поддерживая многопользовательский опыт. Реализован сервер и клиент для мультиплеерной игры Pong, использующий библиотеку Pygame и протокол MQTT. Игра имеет клиент-серверную архитектуру, где сервер управляет движением мяча и ракеток, а клиенты могут управлять направлением своих игроков. Код игрового сервера, основанного на Pygame и протоколе MQTT, демонстрирует эффективное использование MQTT для обеспечения взаимодействия между сервером и клиентами в многопользовательском приложении.

Протокол MQTT обеспечивает эффективный и легковесный механизм обмена сообщениями между сервером и клиентами. Это особенно важно в игровых приложениях, где требуется быстрая передача данных для обеспечения плавного игрового процесса. Использование протокола MQTT поддерживает гибкое асинхронное взаимодействие между сервером и клиентами. Это позволяет обрабатывать события в реальном времени, что является ключевым аспектом в многопользовательских играх. MQTT обеспечивает гарантированную доставку сообщений, что особенно важно для игр, где необходимо избегать потерь данных. Это обеспечивает стабильное и надежное взаимодействие между сервером и клиентами.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1 METANIT [Электронный ресурс]. – Режим доступа: https://metanit.com. Дата доступа: 06.10.2023;

2 GitHub [Электронный ресурс]. – Режим доступа: https://github.com. Дата доступа: 10.11.2023;

3 Amazon Web Services [Электронный ресурс]. – Режим доступа https://aws.amazon.com/ru/what-is/mqtt/. Дата доступа: 26.09.2023;

4 Twilio [Электронный ресурс]. – Режим доступа https://www.twilio.com/blog/what-is-mqtt. Дата доступа: 21.09.2023.

5 MQTT [Электронный ресурс]. – Режим доступа https://mqtt.org. Дата доступа: 21.09.2023.

6 HiveMQ [Электронный ресурс]. – Режим доступа https://www.hivemq.com/mqtt/. Дата доступа: 20.11.2023.