РЕФЕРАТ

Отчет 30 с., 10 рис., 21 лист., 2 источн.

СЕРВЕР, ОНЛАЙН ИГРА, PYGAME, SOCKET, IP-АДРЕСА, ПОРТЫ, ЯЗЫК ПРОГРАММИРОВАНИЯ, РАЗРАБОТКА, PYTHON

Объект исследования: алгоритмы разработки серверной части онлайн игры.

Цель работы: изучить методы и алгоритмы создания серверной части онлайн игры. Написать функционирующий код программы сервера.

Методы: анализ, моделирование, логический метод, синтез, гипотетический метод, изучение и обобщение.

В результате проведенной работы были изучены методы и алгоритмы создания серверной части онлайн игры, была реализована программа этой серверной части на языке python с использованием модулей pygame и socket.

Область применения: программирование клиент-серверных приложений.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность: разработка серверной части того или иного приложения является довольно актуальной задачей, поскольку при изучении данной темы познается принцип работы клиент-серверного программного продукта (ПП), а также различные способы коммуникации клиентской и серверной частей.

Цель: изучение алгоритмов и методов создания серверной части онлайн-приложения и непосредственно разработка программного продукта.

Объект разработки: многопользовательская онлайн-игра.

Предмет разработки: серверная часть программного продукта.

1 Анализ предметной области

В процессе разработки серверной части проекта был использован функционал библиотек pygame и socket языка программирования python.

1.1 Библиотека pygame

1.1.1 Определение

Pygame — это «игровая библиотека», набор инструментов, помогающих программистам создавать игры [1]. К ним относятся:

— Графика и анимация

— Звук (включая музыку)

— Управление/обработка событий от пользователя (мышь, клавиатура, геймпад и так далее)

1.1.2 Принцип работы

В ходе разработки серверной части продукта данная библиотека использовалась для обеспечения наглядности тех или иных функций, добавляемых в код программы, в процессе его написания и для простоты визуализации работы программы [3].

Таким образом, принцип работы следующий:

— подключение библиотеки.

— инициализация игры и игрового окна.

— в основном игровом цикле на каждой итерации контроль FPS (frames per second).

— в основном игровом цикле обработка событий (пользовательского ввода) на каждой итерации при необходимости. В случае работы серверной части в качестве источника изменения данных об игровом поле служат данные, получаемые сервером от клиентов, поэтому пользовательского ввода в коде серверной части не присутствует, как правило.

— в основном игровом цикле обновление данных игрового поля на каждой итерации.

— в основном игровом цикле отрисовка графики/рендеринг на каждой итерации.

1.1.3 Основной функционал библиотеки pygame

1. pygame.Color – представление цвета в библиотеке [1]

Класс Color представляет собой значение цвета в палитре RGBA (red, green, blue, alpha), используя диапазон значений от 0 до 255 включительно. Он позволяет выполнять основные арифметические операции — двоичные операции +, -, *, //, %, и унарная операция ~ — для создания новых цветов, поддерживает преобразования в другие цветовые пространства, такие как HSV или HSL, и позволяет настраивать отдельные цветовые каналы. Альфа-значение по умолчанию равно 255 (полностью непрозрачное), если оно не задано (Рисунок 1).

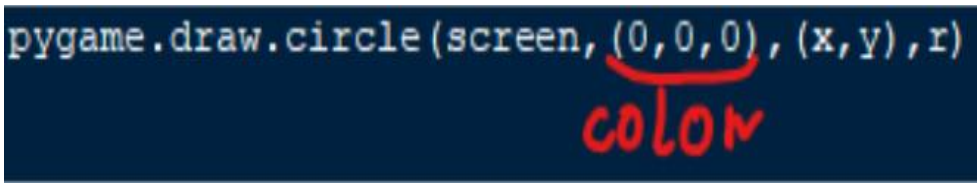

Рисунок 1 - Пример использования объекта color

2. pygame.display – модуль контроля окна дисплея и экрана [1]

Этот модуль обеспечивает управление дисплеем pygame. Pygame имеет единственную поверхность отображения, которая либо содержится в окне, либо работает во весь экран. Как только вы создаете дисплей, вы рассматриваете его как обычную поверхность. Изменения не сразу видны на экране; вы должны выбрать одну из двух функций переключения, чтобы обновить фактическое отображение. Начало отображения, где x = 0 и y = 0, находится в левом верхнем углу экрана. Обе оси положительно увеличиваются по направлению к нижней правой части экрана.

Дисплей pygame фактически может быть инициализирован в одном из нескольких режимов. По умолчанию дисплей представляет собой базовый программный буфер кадров. Вы можете запросить специальные модули, такие

как автоматическое масштабирование или поддержка OpenGL. Они управляются флагами, переданными в метод «.set_mode()» (Рисунок 3).

Одновременно pygame может использовать только один дисплей, а создание нового закроет предыдущий.

Инициализация дисплея происходит при помощи метода «.init()» (Рисунок 3).

Заполнение экрана некоторым цветом вызывается функцией «.fill(color)», color – цвет, которым будет заполняться поверхность. Данный метод вызывается в самом начале этапа отрисовки, чтобы не заполнять экран цветом поверх других объектов (Рисунок 2).

Обновление состояния игрового дисплея вызывается методом «.update()» (Рисунок 4).

Чтобы деинициализировать дисплей необходимо вызвать метод «.quit()» (Рисунок 4).

```
screen.fill('grey25')
```

Рисунок 2 - Заполнение экрана серверного монитора оттенком серого цвета – “grey25”

```
pygame.init()
screen = pygame.display.set_mode((WIDTH_SERVER_WINDOW, HEIGHT_SERVER_WINDOW))
```

Рисунок 3 - Пример создания дисплея ширины «WIDTH_SERVER_WINDOW» и высоты «WIDTH_SERVER_WINDOW»

```
    pygame.display.update();

pygame.display.quit();
```

Рисунок 4 - Обновление дисплея в игровом цикле и закрытие его при выходе из цикла

3. pygame.draw – модуль отрисовки графики [1]

Этот модуль будет работать для рендеринга в любом формате surface. Рендеринг на аппаратных поверхностях будет происходить медленнее, чем на обычных программных поверхностях. Большинство функций принимают аргумент width для представления размера обводки (толщины) по краю

фигуры. Если задана ширина 0, фигура будет заполнена (сплошная). Все функции рисования учитывают область обрезки поверхности и будут ограничены этой областью. Функции возвращают прямоугольник, представляющий ограничивающую область измененных пикселей. Этот ограничивающий прямоугольник является "минимальной" ограничительной рамкой, которая охватывает затронутую область. Все функции рисования принимают аргумент цвета, который может быть одним из следующих форматов:

1. Объект класса pygame.Color
2. Тройка чисел задающих код цвета в RGB-модели
3. Четверка чисел, задающих код цвета в RGBA-модели
4. Целое значение, которое было представлено в формате пикселей поверхности при помощи: pygame.Surface.map_rgb() - преобразует цвет в отображаемое значение цвета, и pygame.Surface.unmap_rgb() - преобразует отображенное целочисленное значение цвета в цвет).

Прозрачность цвета будет записана непосредственно на поверхность (если поверхность содержит пиксельные альфа-значения), но функция рисования не будет рисовать прозрачно. Эти функции временно блокируют поверхность, на которой они работают. Многие последовательные вызовы рисования можно ускорить, блокируя и разблокируя объект Surface вокруг вызовов рисования: pygame.Surface.lock() - блокирует память Surface для доступа к пикселям, и pygame.Surface.unlock() - разблокировывает память Surface с помощью доступа к пикселям).

В разработке серверной части приложения для удобства необходим был всего лишь один метод данного модуля, а именно – «.circle(surface, color, center, radius)» , позволяющий отрисовывать окружность на поверхности surface, цвета color (любой из представленных выше форматов), с центром в координатах center (кортеж координат (x,y)), радиусом radius.

1.2 Библиотека socket

1.2.1 Определение

Сокет - название программного интерфейса для обеспечения обмена данными между процессами. Процессы при таком обмене могут исполняться как на одной ЭВМ, так и на различных ЭВМ, связанных между собой сетью. Сокет - абстрактный объект, представляющий конечную точку соединения [2][4].

Следует различать **клиентские** и **серверные сокеты**. Клиентские сокеты грубо можно сравнить с конечными аппаратами телефонной сети, а серверные - с коммутаторами. Клиентское приложение (например, браузер) использует только клиентские сокеты, а серверное (например, веб-сервер, которому браузер посылает запросы) - как клиентские, так и серверные сокеты [2][4].

Для взаимодействия между машинами с помощью стека протоколов TCP/IP используются адреса и порты. Адрес представляет собой 32-битную структуру для протокола IPv4, 128-битную для IPv6. Номер порта — целое число в диапазоне от 0 до 65535 (для протокола TCP) [2][4].

Эта пара определяет сокет («гнездо», соответствующее адресу и порту).

В процессе обмена, как правило, используется два сокета — сокет отправителя и сокет получателя. Например, при обращении к серверу на HTTP-порт сокет будет выглядеть так: 194.106.118.30:80, а ответ будет поступать на mmm.nnn.ppp.qqq:xxxxx [2][4].

Каждый процесс может создать «слушающий» сокет (серверный сокет) и *привязать* его к какому-нибудь порту операционной системы (в UNIX непривилегированные процессы не могут использовать порты меньше 1024) [2].

Слушающий процесс обычно находится в цикле ожидания, то есть просыпается при появлении нового соединения. При этом сохраняется

возможность проверить наличие соединений на данный момент, установить тайм-аут для операции и т. д. [2].

Каждый сокет имеет свой адрес. ОС семейства UNIX могут поддерживать много типов адресов, но обязательными являются INET-адрес и UNIX-адрес. Если привязать сокет к UNIX-адресу, то будет создан специальный файл (файл сокета) по заданному пути, через который смогут сообщаться любые локальные процессы путём чтения/записи из него. Сокеты типа INET доступны из сети и требуют выделения номера порта [2].

Обычно клиент явно «подсоединяется» к слушателю, после чего любое чтение или запись через его файловый дескриптор будут передавать данные между ним и сервером [2].

1.2.2 Сокеты в python

Модуль socket обеспечивает доступ к интерфейсу сокета BSD. Он доступен на всех современных системах Unix, Windows, macOS и, возможно, на дополнительных платформах [2][4].

Интерфейс Python представляет собой простую транслитерацию интерфейса системного вызова и библиотеки Unix для сокетов в объектно-ориентированный стиль Python: функция «socket()» возвращает объект сокета, методы которого реализуют различные системные вызовы сокетов, функция «bind(“IPv4address”, port)» привязывает сокет сервера к порту указанного адреса протокола IPv4. Типы параметров несколько более высокого уровня, чем в интерфейсе C: как и в случае операций «read()» и «write()» в файлах Python, выделение буфера при операциях приема происходит автоматически, а длина буфера неявно указывается при операциях отправки [2][4].

Основной функционал данного модуля:

1. метод socket(<протокол IP адреса>, <TCP or UDP socket>) (Рисунок 5).
2. метод setsockopt() – устанавливает в сокет передаваемые опции (Рисунок 5).

3. метод bind(<IP сервера в выбранном протоколе>, номер порта). Как правило первые 1024 порта заняты системными функциями (Рисунок 5).

4. метод setblocking(0/1) – метод который блокирует (1) или отключает блокировку (0) при отсутствии подключений (Рисунок 5).

5. метод listen(n) – метод ожидания до n подключений (Рисунок 5).

6. метод accept() – метод ожидающий подключений, при подключении возвращает кортеж из двух значений – новый сокет, и адрес подключенного клиента (Рисунок 6).

7. метод recv(n) – метод получения до n байт информации с данного сокета (Рисунок 7).

8. методы encode() и decode() – кодируют и декодируют соответственно информацию из строкового типа в байты и наоборот (Рисунок 8, Рисунок 9).

9. метод send(data) – метод отправки закодированных данных data на данный сокет(Рисунок 9).

10. метод close() – метод закрытия соединений с данным сокетом (Рисунок 10).

```
main_socket=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
main_socket.setsockopt(socket.IPPROTO_TCP, socket.TCP_NODELAY, 1)
main_socket.bind(('localhost',10000))
main_socket.setblocking(0)
main_socket.listen(30)
```

Рисунок 5 - Инициализация главного (серверного) сокета с протоколом IPv4 и опцией отключения пакетирования отправляемых данных по адресу ‘localhost’ на порт 10000 без блокирования с готовностью обрабатывать до 30 подключений

```
new_socket, addr = main_socket.accept()
```

Рисунок 6 - Пример реализации метода accept()

```
#Чтение данных из сокетов клиентов в объеме 1024 байт
data = player.conn.recv(1024)
```

Рисунок 7 - Пример реализации метода recv()

```
#Перевод полученной инфы из байт в необходимый тип данных
data = data.decode()
```

Рисунок 8 - Пример реализации метода decode()

```
player.conn.send((str(START_PLAYER_R)+' '+player.color).encode())
```

Рисунок 9 - Пример реализации методов send() и encode()

```
main_socket.close()
```

Рисунок 10 - Пример реализации метода close()

2 Описание практической части работы

2.1 Импортирование всех необходимых библиотек

Импортируем библиотеки pygame и socket для работы с графикой и соединения и общения клиента и сервера.

Листинг 1 – Импортирование всех необходимых библиотек

```
    import socket
    import time
    import pygame
    import random
```

2.2 Создание и настройка сокета сервера

Данный отрывок кода создает серверный сокет с адресом localhost согласно протоколу IPv4, на порту :10000, с отключенным пакетированием отправляемых с серверного сокета данных, без блокировки и готовый слушать (принимать соединения) до 30 клиентов одновременно.

Листинг 2 – Создание и настройка сервера

```
    main_socket=socket.socket(socket.AF_INET,socket.SOCK_STREAM
)
    main_socket.setsockopt(socket.IPPROTO_TCP,
socket.TCP_NODELAY, 1)
    main_socket.bind(('localhost',10000))
    main_socket.setblocking(0)
    main_socket.listen(30)
```

2.3 Задание начальных параметров игровой комнаты

Задаем произвольную ширину и высоту кода как константы. Пускай будут равны 5000 пикселей. Так как такое большое разрешение невозможно отрисовать на экране меньшего размера, зададим так же размер окна для визуализации общения клиента и сервера равным 500x500.

Листинг 3 – Задание начальных параметров игровой комнаты

```
     #Параметры комнаты
    WIDTH_ROOM, HEIGHT_ROOM = 5000,5000
    WIDTH_SERVER_WINDOW, HEIGHT_SERVER_WINDOW = 500,500
```

2.4 Задание параметров игрока и микробов

Задаем начальный размер игрока – 50, а микробов – 30. Антиплотность микробов инициализируем как 80000, чем она больше, тем меньше микробов на карте. Количество микробов на карте задаем, как площадь игровой комнаты, деленную на антиплотность микробов.

Листинг 4 – Задание параметров игрока и микробов

```
    #Начальный размер игроков
    START_PLAYER_R = 50
    #Параметры микробов
    MICROBES_SIZE = 30
    #Чем больше плотность тем меньше микробов
    MICROBS_ANTI_DENSITY = 80000
    MICROBS_QUANTITY = WIDTH_ROOM * HEIGHT_ROOM //
MICROBS_ANTI_DENSITY
```

2.5 Задание максимальной частоты кадров

Устанавливаем максимальный FPS (frames per second)

Листинг 5 – Задание начальных параметров игровой комнаты

```
    #Частота игры макс.
    FPS = 144
```

2.6 Задание палитры цветов

Создаем словарь цветов с ключами 1 – 4, значения которых – представление цветов в палитре RGB

Листинг 6 – Задание палитры цветов

```
    #Словарь всех цветов игроков доступных в игре
    colors = {'0':(255,255,0), '1':(255,0,0), '2':(0,255,255),
'3':(0,0,255), '4':(255,0,255)}
```

2.7 Функция find(s)

Функция преобразования получаемых от клиента данных из формата <”Данные”,”Данные”> в формат массива с данными ([data,data])

Листинг 7 – Функция find(s)

```
    def find(s):
        otkr = None
        for i in range(len(s)):
            if s[i] == '<':
                otkr = i
            if s[i] == '>' and otkr != None:
                zakr = i
                res = s[otkr+1:zakr]
                res = list(map(int,res.split(',')))
                return res
        return ""
```

2.8 Создание класса объектов «микроб»

Класс обладает только конструктором, который задает расположение, размер и цвет микробов согласно принимаемым аргументам.

Листинг 8 – Класс объектов «микроб»

```
    #Создание класса объектов корма
    class Microbe():
        def __init__(self,x,y,r,color):
            self.x = x
            self.y = y
            self.r = r
            self.c = color
```

2.9 Создание класса объектов «игрок»

У данного класса есть параметризованный конструктор, а также методы: установки свойств игрока, смены скорости по осям x и y, обновления координат в пространстве игрового поля, размера игрока и масштаба игрового поля.

Конструктор класса инициализирует сокет данного игрока, его аддрес, координаты по оси x, y, размер игрока, его цвет, количество идущих подряд отловленных ошибок, стандартное имя = UserName, начальный масштаб = 1, поле зрения игрока, его готовность, скорость по осям x, y, абсолютное значение скорости.

Метод установки свойств объекта использует полученные от клиента данные, чтобы установить имя игрока, которое было им выбрано в начале игры, вводом в консоль, а также поле зрения, определенное на клиенте, таким же образом.

Метод смены скорости устанавливает скорость по осям x и y в нуль при условии, что в качестве параметра был принят нулевой вектор, иначе меняет скорость по осям на соответствующую направлению вектора движения.

Последний метод изменяет координаты игрока в пространстве игрового поля согласно составляющим скорости по осям x и y. Также обновляет абсолютную скорость игрока в зависимости от размера (чем больше, тем медленнее). Помимо этого, он обеспечивает уменьшение радиуса игрока со временем до предела 100. И в конце концов, изменяет поле зрения в соответствии с масштабом.

Листинг 9 – Класс объектов «игрок», 1 часть

```
    #Создание класса объектов игрока
    class Player():
        def __init__(self, conn, addr, x, y, r, color):
            self.conn = conn
            self.addr = addr
            self.x = x
            self.y = y
            self.r = r
            self.color = color
            self.errors = 0
            self.name = "UserName"

            self.L = 1
            self.width_window = 1000
            self.height_window = 800
            self.w_vision = 1000
            self.h_vision = 800
            self.ready = False

            self.speed_x = 0
            self.speed_y = 0
            self.abs_speed = 30/(self.r**0.5)

        def set_options(self, data):
            data = data[1:-1].split(' ')
            self.name = data[0]
            self.width_window = int(data[1])
            self.height_window = int(data[2])
            self.w_vision = int(data[1])
            self.h_vision = int(data[2])

        def change_speed(self, v):
            if (v[0] == 0) and (v[1] == 0):
                self.speed_x = 0;
                self.speed_y = 0;
            else:
                lenv = (v[0]**2 + v[1]**2)**0.5
                v = (v[0]/lenv,v[1]/lenv)
                v = (v[0]*self.abs_speed,v[1]*self.abs_speed)
                self.speed_x = v[0]
                self.speed_y = v[1]
```

Листинг 10 – Класс объектов «игрок», 2 часть

```
        def update(self):
            #По-горизонтали
            if self.x <= 0:
                if self.speed_x >= 0:
                    self.x += self.speed_x
            elif self.x >= WIDTH_ROOM:
                if self.speed_x <= 0:
                    self.x += self.speed_x
            else:
                self.x += self.speed_x

            #По-вертикали
            if self.y <= 0:
                if self.speed_y >= 0:
                    self.y += self.speed_y
            elif self.y >= HEIGHT_ROOM:
                if self.speed_y <= 0:
                    self.y += self.speed_y
            else:
                self.y += self.speed_y

            self.abs_speed = 30/(self.r**0.5)

            #Постепенное уменьшение радиуса игрока со временем
до предела(100)
            if self.r >= 100:
                self.r -= self.r/18000

            if self.r >= self.w_vision/4 or self.r >=
self.h_vision/4:
                if self.w_vision <= WIDTH_ROOM or self.h_vision
<= HEIGHT_ROOM:
                    self.L*=2
                    self.w_vision = self.width_window*self.L
                    self.h_vision = self.height_window*self.L

            if self.r < self.w_vision/8 and self.r <
self.h_vision/8:
                if self.L > 1:
                    self.L = self.L//2
                    self.w_vision = self.width_window*self.L
                    self.h_vision = self.height_window*self.L
```

2.10 Создание окна серверного монитора

Данный участок кода инициализирует игровое приложение, создает объект screen, класса display с заданными ранее шириной и высотой

серверного окна, инициализирует фиксатор кадров в секунду. Также устанавливает значение предиката основного игрового цикла в значение «истина».

Листинг 11 – Создание окна сервера для визуализации игры

```
    #Создание окна сервера
    pygame.init()
    screen = pygame.display.set_mode((WIDTH_SERVER_WINDOW,
HEIGHT_SERVER_WINDOW))
    clock = pygame.time.Clock()
    running = True
```

2.11 Создание массива с сокетами игроков, массива имен и очков игроков и массива стартового набора микробов

Листинг 12 – Создание массива с сокетами игроков, массива имен и очков игроков и массива стартового набора микробов

```
    #Массив с сокетами игроков
    players = []
    #Массив с именами и размерами игроков
    top = ""
    #Создание стартового набора микробов
    microbes = [Microbe(random.randint(0,WIDTH_ROOM),
                        random.randint(0,HEIGHT_ROOM),
                        MICROBES_SIZE,
                        str(random.randint(0,4)))
                for i in range (MICROBS_QUANTITY)]
```

2.12 Запуск основного цикла игры и фиксация кадров в секунду

Листинг 13 – Запуск основного цикла игры и фиксация кадров в секунду

```
    #Цикл игры
    while running:
        clock.tick(FPS)
```

2.13 Прием подключений от игроков если они есть

В этой части кода при помощи обработчика исключений ожидаются подключения от клиентов. При успешном подключении будут получены сокет и адрес нового клиента и будет создан объект класса «игрок» с соответствующими параметрами в случайной части игрового поля и со случайным цветом кружка. При появлении ошибки при подключении она будет игнорироваться.

Листинг 14 – Прием подключений от игроков если они есть

```
    #Прием подключений от игроков, если они есть
        try:
            new_socket, addr = main_socket.accept()
            print("Игрок ", addr, " подключился")
            new_socket.setblocking(0)
            new_player =
Player(new_socket,addr,random.randint(0,WIDTH_ROOM),
            random.randint(0,HEIGHT_ROOM),START_PLAYER_R,str(ra
ndom.randint(0,4)))
            players.append(new_player)
        except:
            pass
```

2.14 Считывание команд клиентов

В массиве игроков перебираем их и принимаем данные из сокета каждого из них в объеме до 1024 байт. Если пришло сообщение о готовности, присваиваем полю ready этого игрока значение «истина», иначе если пришло сообщение для установки первичных свойств, устанавливаем начальные свойства игроку, передаваемые от клиента, отправляем клиенту данные для отрисовки, иначе распаковываем данные от клиента при помощи функции find(data) и вызываем метод change_speed(data) данного игрока. При возникновении ошибки, отлавливаем и игнорируем. Обновляем инфу об игроке вызовом его метода update()

Листинг 15 – Считывание команд клиентов

```
    #Считываем команды игроков
        for player in players:
            try:
                #Чтение данных из сокетов клиентов в объеме
1024 байт
                data = player.conn.recv(1024)
                #Перевод полученной инфы из байт в необходимый
тип данных
                data = data.decode()
                if data[0] == '!':
                    player.ready = True
                else:
                    if data[0] == '.' and data[-1] == '.':
                        player.set_options(data)

                        player.conn.send((str(START_PLAYER_R)+'
'+player.color).encode())
                    else:
                        data = find(data)
                        #Обрабатываем полученные от игрока
данные
                        player.change_speed(data)
            except:
                pass
            player.update()
```

2.15 Определение, что видит каждый игрок

Создаем массив, в котором находятся подмассивы с видимыми каждому игроку кружками (и игроками и кормом). Для начала рассмотрим какой корм игрок видит. Для этого проходимся по массиву микробов и если модуль расстояния от игрока до корма входит в видимый диапазон, то в массив видимых шаров добавляем инфу о данном шарике корма. Так же проверяем ситуацию если игрок есть корм (расстояние меньше чем радиус игрока). Если корм таки был съеден, то он регенерируется в новой случайной точке комнаты, а игроку добавляются очки размера.

Листинг 16 – Определение, что видит каждый игрок, 1 часть (обнаружение корма)

```
    #Определим, что видит каждый игрок
        #Каждый подмассив содержит объекты видимых игроков для
данного
        visible_balls = [[] for i in range(len(players))]

        for i in range(len(players)):
            #Каких микробов видит i
            for k in range(len(microbes)):
                dist_x = microbes[k].x - players[i].x
                dist_y = microbes[k].y - players[i].y

                #микроб в поле видимости i
                if ((abs(dist_x) <= (players[i].w_vision)//2 +
microbes[k].r)
                    and (abs(dist_y) <=
(players[i].h_vision)//2 + microbes[k].r)):

                    #Подготовим данные к добавлению в список
видимых микробов
                    x_ = str(round(dist_x/players[i].L))
                    y_ = str(round(dist_y/players[i].L))
                    r_ = str(round(microbes[k].r/players[i].L))
                    c_ = str(microbes[k].c)

                    visible_balls[i].append(x_+' '+y_+' '+r_+'
'+c_)

                    #i ест микроба
                    if ((dist_x**2 + dist_y**2)**0.5 <=
players[i].r):
                        #Меняем положение микроба и его цвет
                        microbes[k].x =
random.randint(0,WIDTH_ROOM)
                        microbes[k].y =
random.randint(0,HEIGHT_ROOM)
                        microbes[k].c =
str(random.randint(0,4))
                        players[i].r = (players[i].r**2 +
microbes[k].r**2)**0.5
```

Аналогично массиву с кормом, проходимся по массиву игроков, и если расстояние от одного до другого входит в видимый диапазон, то добавляем необходимую информацию об игроке в массив видимых шаров. Если расстояние меньше радиуса игрока, то обнуляем радиус (удаляем игрока с поля) увеличивая очки поглотившего его. Сразу же рассматриваем ситуацию,

когда данный игрок видим для других и добавим данные i-того игрока j-тому, аналогично так же рассмотрим ситуацию, когда j-тый игрок ест i-того.

Листинг 17 – Определение, что видит каждый игрок, 2 часть (i видит j)

```
    for j in range(i+1, len(players)):
                dist_x = players[j].x - players[i].x
                dist_y = players[j].y - players[i].y

                #j в поле видимости i
                if (abs(dist_x) <= (players[i].w_vision)//2 +
players[j].r
                    and abs(dist_y) <= (players[i].h_vision)//2
+ players[j].r):

                    #Может ли i съесть j
                    if ((dist_x**2 + dist_y**2)**0.5 <=
players[i].r) and players[i].r > 1.1 * players[j].r:
                        #Удаляем игрока с поля добавляя радиус
поглотившего его игрока
                        players[i].r = (players[i].r**2 +
players[j].r**2)**0.5
                        players[j].r, players[j].speed_x,
players[j].speed_y = 0, 0, 0

                    #Подготовим данные к добавлению в список
видимых шаров
                    x_ = str(round(dist_x/players[i].L))
                    y_ = str(round(dist_y/players[i].L))
                    r_ = str(round(players[j].r/players[i].L))
                    c_ = str(players[j].color)
                    n_ = players[j].name

                    if players[j].r >= 30*players[i].L:
                        visible_balls[i].append(x_+' '+y_+'
'+r_+' '+c_+' '+n_)
                    else:
                        visible_balls[i].append(x_+' '+y_+'
'+r_+' '+c_)
```

Листинг 18 – Определение, что видит каждый игрок, 2 часть (j видит i)

```
    #i в поле видимости j
                if (abs(dist_x) <= (players[j].w_vision)//2 +
players[i].r
                    and abs(dist_y) <= (players[j].h_vision)//2
+ players[i].r):

                    #Может ли j съесть i
                    if ((dist_x**2 + dist_y**2)**0.5 <=
players[j].r) and players[j].r > 1.1 * players[i].r:
                        #Удаляем игрока с поля добавляя радиус
поглотившего его игрока
                        players[j].r = (players[j].r**2 +
players[i].r**2)**0.5
                        players[i].r, players[i].speed_x,
players[i].speed_y = 0, 0, 0

                    #Подготовим данные к добавлению в список
видимых шаров
                    x_ = str(round(-dist_x/players[j].L))
                    y_ = str(round(-dist_y/players[j].L))
                    r_ = str(round(players[i].r/players[j].L))
                    c_ = str(players[i].color)
                    n_ = players[i].name

                    if players[i].r >= 30*players[j].L:
                        visible_balls[j].append(x_+' '+y_+'
'+r_+' '+c_+' '+n_)
                    else:
                        visible_balls[j].append(x_+' '+y_+'
'+r_+' '+c_)
```

2.16 Формирование ответа каждому игроку

Создаем массив ответов сервера каждому игроку. Стандартно пустая строка. Проходимся по игрокам фиксируем текущие параметры каждого игрока. Строку top составляем таким образом, чтобы в ней были все игроки и их размеры через пробел. Конкатенируем массивы строковых значений в одну строку при помощи метода join(), оборачивая полученный результат метода в угловые скобки “<”, “>”. Таким образом ответ каждому игроку будет иметь вид строки <’данные об этом игроке’,’видимые для данного игрока кружки’,’кол-во игроков в комнате’,’список игроков с их очками через пробел’>

Листинг 18 – Формирование ответа каждому игроку

```
    #Формируем ответ каждому игроку
        responses = ['' for i in range(len(players))]
        for i in range(len(players)):
            r_ = str(round(players[i].r/players[i].L))
            x_ = str(round(players[i].x/players[i].L))
            y_ = str(round(players[i].y/players[i].L))
            L_ = str(players[i].L)
            top = ""
            for j in range(len(players)):
                top += (players[j].name + ' ' +
str(round(players[j].r))) + ' '*(j!=len(players)-1)
            responses[i] = '<' + (','.join([r_+' '+x_+' '+y_+'
'+L_]
            + visible_balls[i] + [str(len(players))] + [top]))
+ '>'
```

2.17 Отправка нового состояния игрового поля клиентам

Перебираем массив игроков. Если игроки находятся в состоянии готовности (уже находятся в игре), отправляем закодированный подготовленный выше соответствующий индексу пользователя ответ, обнуляем количество вызванных подряд ошибок у данного игрока. При обработке ошибки увеличиваем кол-во ошибок, появляющихся подряд у данного игрока, на единицу.

Листинг 19 – Отправка обновленного состояния игрового поля

```
    #Отправляем обновленное состояние игрового поля 100 раз в
секунду
        for i in range(len(players)):
            if players[i].ready:
                try:
                    players[i].conn.send(responses[i].encode())
                    players[i].errors = 0
                except:
                    players[i].errors += 1
```

2.18 Чистка игрового поля от проигравших игроков и игроков со слишком большим временем ожидания ответа к серверу

Перебираем игроков в массиве игроков. Если данный игрок был съеден (радиус == 0) или его время ожидания истекло (кол-во ошибок >= 500), то разрываем с его клиентом соединение и удаляем его из списка игроков.

Листинг 20 – Чистка игрового поля

```
    #Чистим список от отвалившихся игроков
        for player in players:
            if player.errors >= 500 or player.r == 0:
                player.conn.close()
                players.remove(player)
```

2.19 Обработка события закрытия серверного монитора

Обрабатываем все события на сервере. Если был нажат крестик завершаем работу основного игрового цикла.

Листинг 21 – Ожидание закрытия окна серверного монитора

```
    #Ждем нажатия крестика
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
               running = False
               break
```

2.20 Отрисовка состояния комнаты в серверном мониторе

Заполняем дисплей оттенком серого «grey25». Проходимся по элементам массива игроков. Фиксируем информацию о них и отрисовываем каждого игрока на серверном мониторе. Обновляем дисплей.

Листинг 22 – Отрисовка состояния комнаты в серверном мониторе

```
      #Нарисуем состояние комнаты на сервере
        screen.fill('grey25')
        for player in players:
            x = round(player.x*WIDTH_SERVER_WINDOW/WIDTH_ROOM)
            y =
round(player.y*HEIGHT_SERVER_WINDOW/HEIGHT_ROOM)
            r = round(player.r*WIDTH_SERVER_WINDOW/WIDTH_ROOM)
            c = colors[player.color]
            pygame.draw.circle(screen,c,(x,y),r)
        pygame.display.update()
```

2.21 Закрытие приложения и всех подключений

Листинг 23 – Закрытие приложения и всех подключений

```
    pygame.quit()
    main_socket.close()
```

ЗАКЛЮЧЕНИЕ

В ходе работы были изучены алгоритмы и методы создания серверной части онлайн-игры. Практическая часть работы была выполнена успешно. Серверная часть игры работает корректно и в совокупности с клиентской частью представляет полноценное функционирующее онлайн-приложение.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Pygame documentation, Reference. Режим доступа: https://www.pygame.org/docs/ (Дата обращения 11.05.2022)

2. Python documentation, socket – Low-level networking interface. Режим доступа: https://docs.python.org/3/library/socket.html# (Дата обращения 11.05.2022)

3. Златопольский Д. М., Основы программирования на языке python – М.: ДМК Пресс, 2017. – 284 с.

4. Мэтиз Эрик, Изучаем программирование игр, визуализация данных, веб-приложения. 3-е изд. –СПб.: Питер, 2020. – 512 с.: ил. – (Серия «Библиотека программиста»)