Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные системы и сети (КСИС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

**ПРОГРАММНОЕ ИГРОВОЕ СРЕДСТВО**

**«Palack»**

БГУИР КП 1-40 01 01 028 ПЗ

Студент Прокопчук Д.М.

Руководитель Красковский П.Н.

Минск 2023

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc135050817)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc135050818)

[1.1 Обзор аналогов 6](#_Toc135050819)

[1.2 Постановка задачи 8](#_Toc135050820)

[2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 9](#_Toc135050821)

[2.1 Структура программы 9](#_Toc135050822)

[2.2 Проектирование интерфейса программного средства 9](#_Toc135050823)

[2.3 Проектирование функционала программного средства 12](#_Toc135050824)

[3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА 20](#_Toc135050825)

[3.1 Взаимодействие с сервером 20](#_Toc135050826)

[4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 27](#_Toc135050827)

[5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 30](#_Toc135050828)

[5.1 Интерфейс программного средства 30](#_Toc135050829)

[5.2 Управление программным средством 33](#_Toc135050830)

[ЗАКЛЮЧЕНИЕ 34](#_Toc135050831)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 35](#_Toc135050832)

[ПРИЛОЖЕНИЕ А 35](#_Toc135050833)

# ВВЕДЕНИЕ

Видеоигра (от [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) video game) – [игра](https://ru.wikipedia.org/wiki/%D0%98%D0%B3%D1%80%D0%B0) с использованием изображений, сгенерированных электронной аппаратурой. Другими словами, видеоигра является [электронной игрой](https://en.wikipedia.org/wiki/electronic_game), которая базируется на взаимодействии человека и устройства посредством визуального [интерфейса](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81), например [телевизора](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BB%D0%B5%D0%B2%D0%B8%D0%B7%D0%BE%D1%80), [монитора](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%BD%D0%B8%D1%82%D0%BE%D1%80_(%D1%83%D1%81%D1%82%D1%80%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE)) компьютера или телефона. Исторически видеоигры появились раньше, чем был создан персональный компьютер. Это обусловлено тем, что в 1960-х годах компьютерные технологии не были массовыми и не рассматривались как серьёзный инструмент в гражданской сфере. Создаваемые видеоигры на электронной базе (электронные игры) сыграли важную роль в мирной социализации компьютерной техники и её интеграции в популярную культуру. В 1970-х видеоигры стали одним из самых популярных развлечений, когда устанавливались игровые автоматы в барах, кафе и других заведениях. Компьютерные игры оказали столь существенное влияние на общество, что в [информационных технологиях](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5_%D1%82%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D0%B8) отмечена устойчивая тенденция к [геймификации](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D0%B9%D0%BC%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F) для неигрового [прикладного программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5).

Сейчас существует большое количество различных жанров видеоигр: симуляторы, стратегии, RPG, файтинги, стелс, текстовые, рогалики, визуальные новеллы и многие другие. Популярностью пользуются и различного рода казуальные иры. Казуальная игра — [компьютерная игра](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0), предназначенная для широкого круга пользователей. Казуальные игры отличаются простыми правилами и не требуют от пользователя затрат времени на обучение или каких-либо особых навыков; они дёшевы в разработке и при дистрибуции. Многие подобные игры обладают также яркой и привлекательной графикой и минимумом текста. Казуальным играм противопоставляют «хардкорные» игры со сложными правилами, рассчитанные на сравнительно узкую аудиторию опытных игроков. Казуальные игры как отдельное направление, которое противопоставляется хардкорным играм, начало популяризироваться с 2000 года. Несмотря на это, казуальные игры являются и самыми первыми играми, которые появились на компьютерах и [игровых приставках](https://ru.wikipedia.org/wiki/%D0%98%D0%B3%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BF%D1%80%D0%B8%D1%81%D1%82%D0%B0%D0%B2%D0%BA%D0%B0). В 2000 году компания [PopCap](https://ru.wikipedia.org/wiki/PopCap_Games) запустила игру под названием Bejeweled. Как признаются сами представители компании, «мы даже не представляли, что эта скромная игра, основанная на перестановке бриллиантиков, зародит целый жанр — казуальные игры».

Целью данной курсовой работы является разработка игрового средства «Palack».

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Казуальные игры предназначены для широкой аудитории пользователей. Они как правило не требовательны к персональным компьютерам пользователей, а также не требуют особых навыков и знаний для игры. Простейшими казуальными играми можно назвать те, которые включаются в стандартную поставку дистрибутивов операционных систем. Казуальные игры по своему уровню сложности годятся практически для любой категории пользователей компьютера. Зачастую время на прохождение таких игр невелико, а потому они хорошо подходят для тех, кто не может уделять игре много времени. Дистрибутивы таких игр обычно имеют небольшой размер и распространяются так, чтобы их было легко скачать и тут же начать игровой процесс. В развитии индустрии казуальных игр не последнюю роль сыграла общедоступность Интернета — зародившиеся технологии электронной коммерции (платёжные сервисы) и передача дистрибутивов через Интернет позволили сделать механизм покупки и получения игр быстрым и удобным.

## Обзор аналогов

1.1.1 “Gartic Phone”

“Gartic Phone” — игра, выпущенная в 2018 году. Играть можно в любом браузере, данная игра не является требовательной к машинам пользователей. Игра представляет собой игру, известную многим еще с детства, когда группа детей по очереди шепчет друг другу на ухо некоторое сообщение, а в конце последний озвучивает результат. Дополнением в “Gartic Phone” является наличие рисунков. Таким образом сперва пользователь выбирает некоторую фразу, описывающую какое-либо событие. Далее он получает такую же фразу, но от другого пользователя и пользователь должен нарисовать ту ситуацию, что описана текстом. Далее следующий уже получает рисунок и должен описать происходящее и так далее. Игровой процесс сам по себе не замысловатый, но зато способен дать положительные эмоции сразу группе людей. На рисунке 1.1 предоставлено меню аутентификации программы.

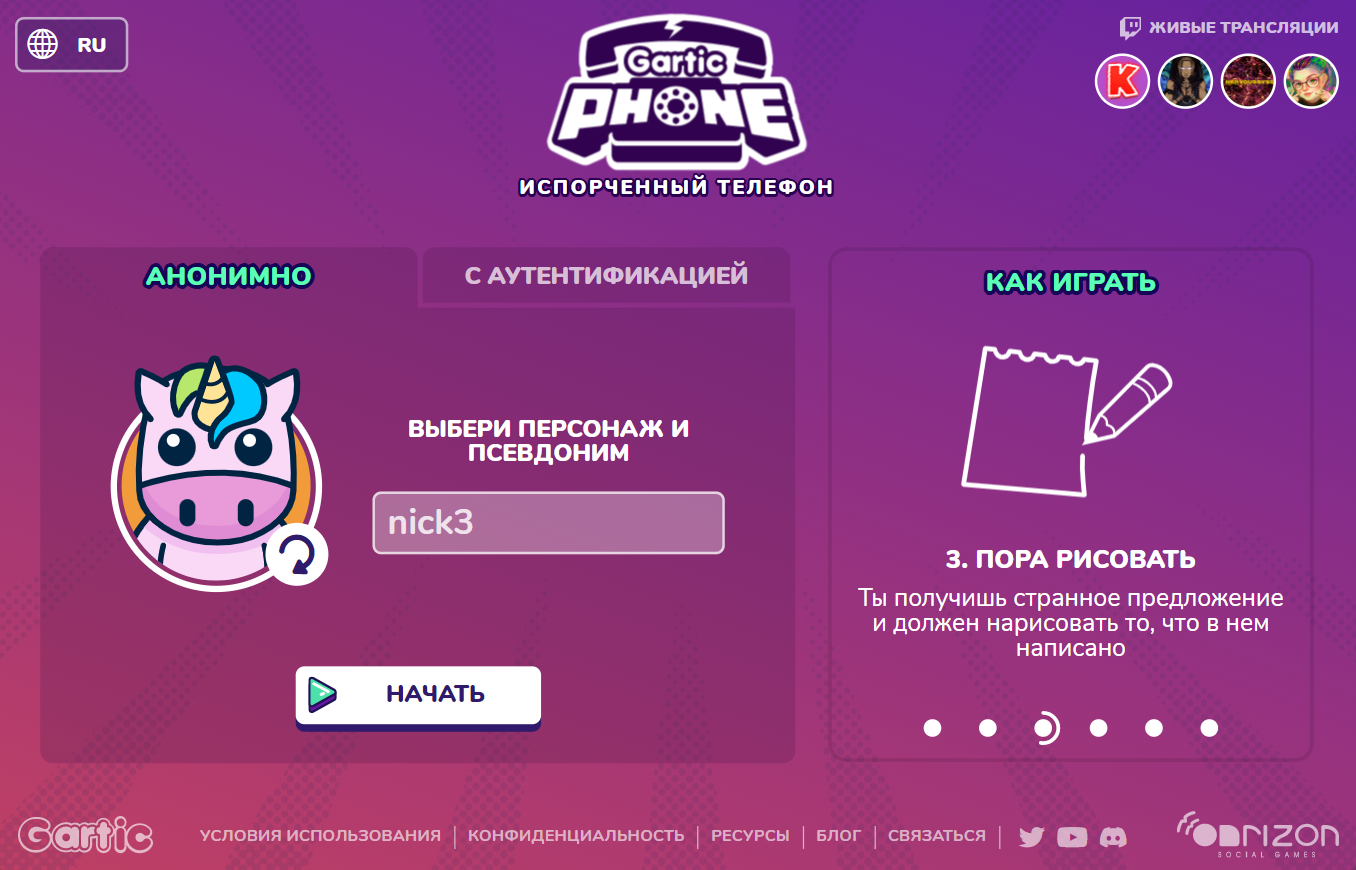


Рисунок 1.1 – Интерфейс игры “Gartic Phone”

Таким образом пользователь регистрирует себя на сервере, а далее уже может начать игру. Сразу после регистрации создается игровая комната, к которой могут подключиться остальные пользователи по ссылке, введя ее в браузер. Когда необходимое количество игроков подключиться, игрок, создавший комнату, сможет начать игру в выбранном формате. На рисунке 1.2 будет показана игровая комната игры “Gartic Phone”.



Рисунок 1.2 – Игровая комната игры “Gartic Phone”

## Постановка задачи

В данном игровом средстве планируется реализовать следующие функции:

* создание игровой комнаты;
* подключение игрока к игровой комнате;
* проверка псевдонима на уникальность;
* осуществление игрогого процесса;
* составление структуры передачи сообщений между игроками;
* вывод результатов на экран;

Для разработки программного средства будет использоваться язык программирования Python и среда разработки PyCharm, а также библиотеки tkinter и tkinter.ttk, socket, pickle, threading.

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

## 2.1 Структура программы

При разработке приложения будет использовано 6 модулей:

* CreateWindow – окно регистрации персонажа;
* HubWindow – окно, в котором находится таблица доступных комнат и кнопки “Создать”, “Присоединиться” и “Обновить”;
* RoomWindow – окно, в котором игроки будут находиться перед стартом игры, запустить игру может только хост;
* GameWindow – окно, в котором происходит сама игра;
* Player – реализация класса, описывающего персонажа;
* GameRoom – реализация класса, описывающео игровую комнату;
* HubServer – общий сервер для всех игроков, весь сетевой код будет так или иначе взаимодействовать с сервером.

## 2.2 Проектирование интерфейса программного средства

2.2.1 Окно регистрации персонажа

Окно регистрации должно содержать некое меню с выбором аватара персонажа, а также поле ввода для никнейма и кнопку “Начать игру”. Выбор аватара может быть путем выбора одного из преустановленных или выбора файла с компьютера пользователя. Макет окна регистрации приложения представлен на рисунке 2.1.

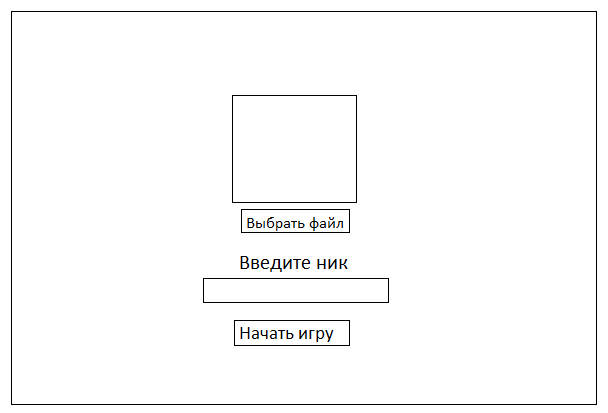


Рисунок 2.1 – Окно регистрации приложения

2.2.2 Окно главного меню

Окно главного меню должно представлять из окно, содержащее аватар и никнейм персонажа, а также таблицу, в который указаны существующие комнаты и 3 кнопки: “Создать комнату”, “Присоединиться” и “Обновить”. По нажатию на данные кнопки игрок будет создавать новую комнату, присоединяться к выбранной в списке и получать актульный список напрямую с сервера соответственно. Увидеть можно на рисунке 2.2.

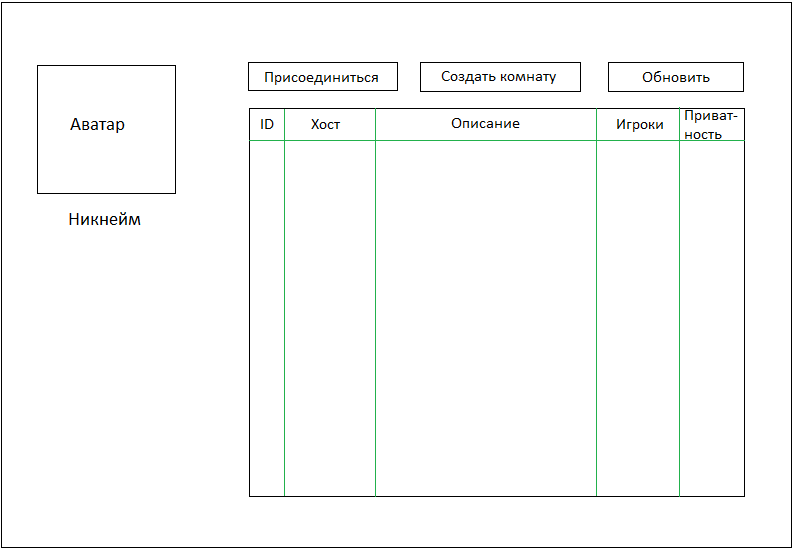


Рисунок 2.2 – Окно главного меню

2.2.3 Окно создания комнаты

Окно создания комнаты должно содержать кнопку подтверждения, а также несколько полей для ввода значений:

* описание комнаты;
* режим игры в комнате;
* максимальная вместимость комнаты;
* приватность комнаты(открытая и закрытая);
* пароль, если комната закрытая.

Макет окна выбора аватара представлен на рисунке 2.3.

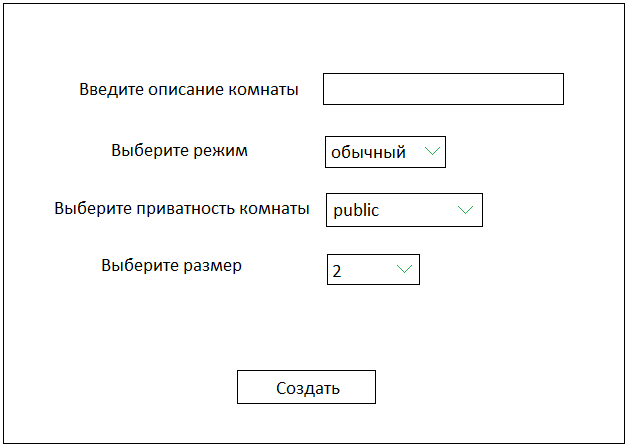


Рисунок 2.3 – Окно создания комнаты

Отдельно будет показано графа выбора пароля, так как она появляется только в случае, если выбрана приватность private. Рядом с полем ввода пароля будет находиться инструмент интерфейса, позволяющий заменить вводимые символы на “\*” для большей безопасности. Графу выбора пароля можно увидеть на рисунке 2.4.



Рисунок 2.4 – Графа ввода пароля

2.2.4 Окно комнаты(окно ожидания)

Окно комнаты будет представлять собой окно с таблицей текущих ироков в комнате, игрок, являющийся хостом, будет иметь доступ к кнопке “Начать игру”, которая автоматически запустит игру для всех пользователей. В таблице будет информация о аватаре, никнейме и статусе(хост или нет) игрока. Макет окна комнаты представлен на рисунке 2.5.



Рисунок 2.5 – Окно комнаты

## 2.3 Проектирование функционала программного средства

При создании игры очень важно сразу определить задачу и цели, а также хорошо составить рабочие алгоритмы. Искать ошибки в коде придётся в любом случае, но хорошо написанный алгоритм упрощает это в разы. В игре должны быть использованы следующие методы:

* создание структуры передачи сообщений между пользователями;
* проверка уникальности никнейма;
* вывод результатов игры;
* механизм передачи больших сообщений через сеть;
* определение текущей стадии иры на основании имеющихся данных.

2.3.1 Структура передачи сообщений между пользователями

Структура передачи сообщений между пользователями необходима, чтобы правильно распределять полученные от пользователей сообщения. Данная задача включает в себя создание словаря, который в последствии будет использоваться как хранилище данных. При создании будет использоваться основная процедура makePlayerStructure. Блок-схема алгоритма данной функции приведена на рисунке 2.6.

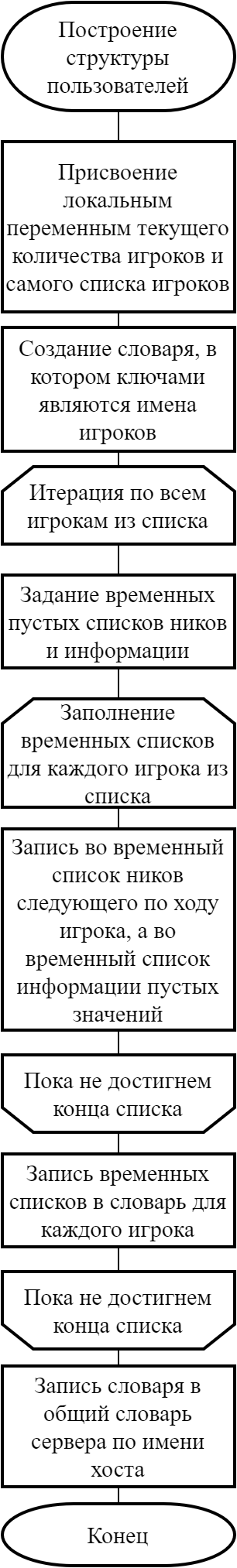


Рисунок 2.6 – Блок-схема процедуры makePlayerStructure.

2.3.2 Проверка уникальности никнейма

Проверка уникальности никнейма является обязательным атрибутом данного программного средства. Суть в том, что очень многие алгоритмы построены на уникальности никнейма каждого отдельного пользователя, поэтому данный алгоритм является обязательным к реализации. Стоит также рассмотреть процедуры добавления и удаления игрока из списка. Блок-схемы алгоритмов функций findNickInBase, addNewPlayer и deleteAPlayer приведены на рисунке 2.7.

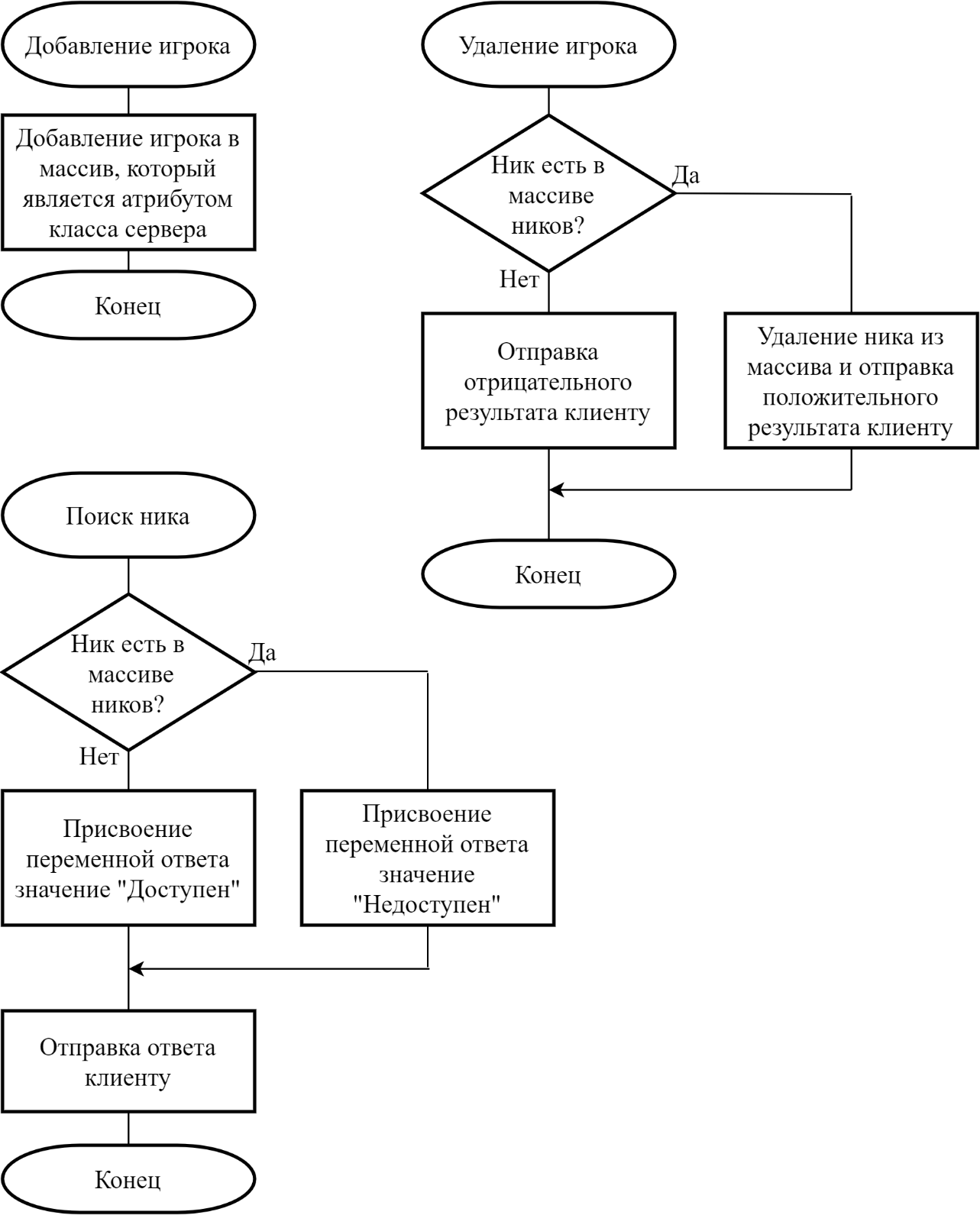


Рисунок 2.7 – Блок-схема процедур findNickInBase, addNewPlayer,deleteAPlayer

2.3.3 Вывод результатов игры

Вывод результатов игры необходим для сохранения сути игры, так как игра мультиплеерная, то возможность просмотреть результаты ответов всех пользователей игры несет в себе огромный интерес для игроков. Данная процедура получает от сервера набор игровой информации и выовдит его на экран. Блок-схема алгоритма процедуры generateResult приведена на рисунке 2.8.



Рисунок 2.8 – Блок-схема процедуры generateResult

2.3.4 Механизм передачи больших сообщений через сеть

Передача больших сообщений является неотъемлимым атрибутом игры, так как зачастую необходимо передать изображение(аватар персонажа) или целую структуру данных больших размеров через сеть на сервер или от сервера к клиенту. Этот механизм будет отличаться от передачи обычного текста. Блок-схема алгоритма процедуры recvBigData приведена на рисунке 2.9.

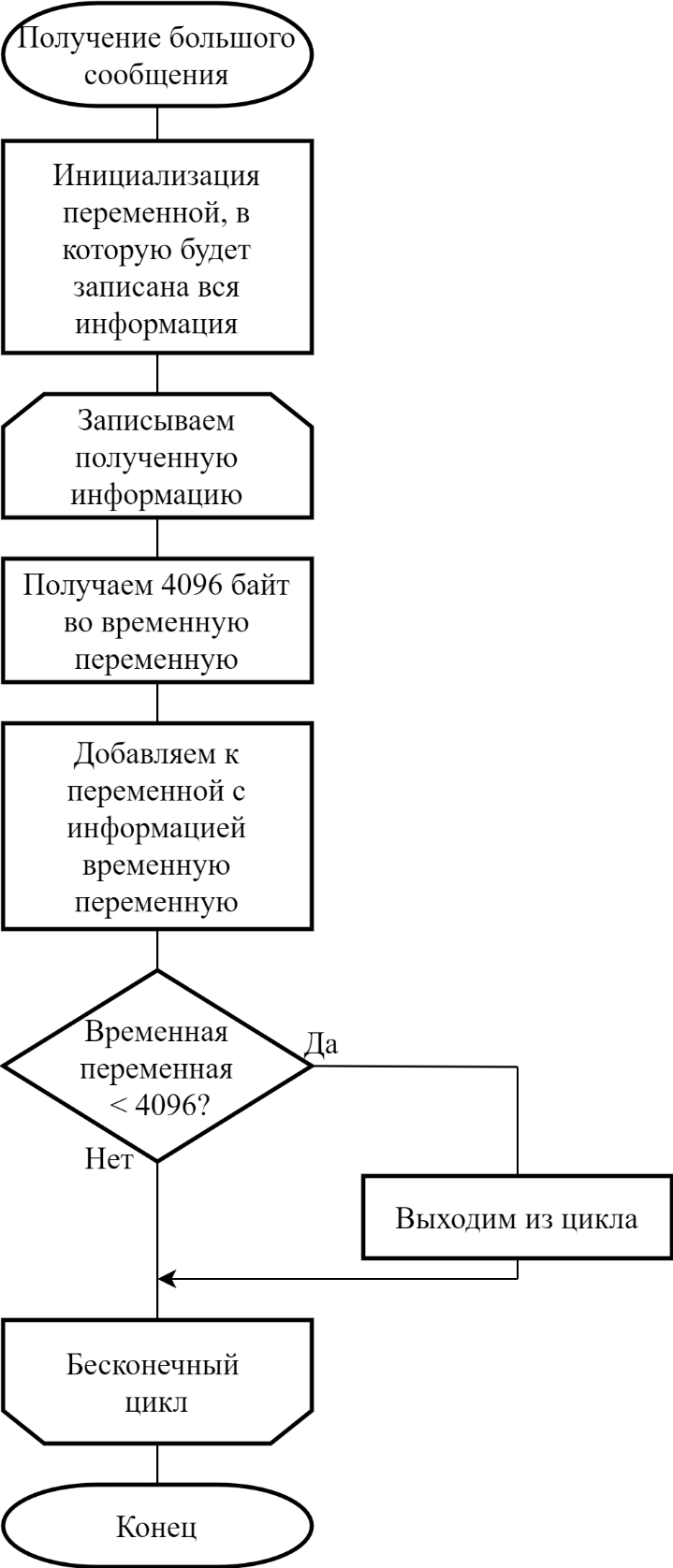


Рисунок 2.9 – Блок-схема процедуры recvBigData

2.3.5 Определение текущей стадии игры на основании имеющихся данных

Определение текущей стадии игры необходимо для автоматической записи данных, так как при вызове функции записи мы передаем только сами данные, а уже куда их написать решается благодаря алгоритму определения текущей стадии игры. Блок-схема алгоритма процедуры calculateGameStage приведена на рисунке 2.9.



Рисунок 2.9 – Блок-схема процедуры calculateGameStage

Стоит также упомянуть про механизм получения запросов от клиента. Данный механизм является наиважнейшей составляющей игры, так как засчет данного механизма осуществляется все сетевое взаимодействие. Суть механизма в том, что полученное сообщение содержит в себе команду и данные для этой команды. По команде серве отслеживает, какой метод необходимо вызвать, и передает в этот метод данные, а также объект соединения.

# РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

## 3.1 Взаимодействие с сервером

Основой данного программного средства является механизм распределения запросов от клиента.

Вызов необходимой функции осуществляется каждый раз, как сервер получает сообщение через сеть от клиента, далее сервер разделяет сообщение и данные к нему. Если сообщение(команда) определено на сервере, то управление передается соответствующей процедуре, также в нее передаются исходные данные. Код процедуры manageMessages будет предоставлен ниже.

def manage\_messages(self, command, message, conn):

match command:

case "CHECKING\_NICK":

self.find\_nick\_in\_base(message, conn)

return False

case "HELLO":

self.adding\_a\_new\_player(message)

return True

case "DISCONNECT":

self.delete\_a\_player(message, conn)

return False

case "REFRESH":

self.send\_current\_list(conn)

return True

case "CREATE\_ROOM":

self.create\_a\_room(conn)

return True

case "DELETE\_ROOM":

self.delete\_a\_room(message)

return True

case "REFRESH\_ROOM":

self.send\_players\_room(message, conn)

return True

case "JOIN\_ROOM":

self.join\_player\_to\_room(conn)

return True

case "CHECK\_STATUS":

self.send\_room\_status(message, conn)

return True

case "START\_ROOM":

self.start\_a\_room(message, conn)

return True

case "ASK\_START":

self.synchronize\_all\_room(message, conn)

return True

case "STOP\_GAME":

self.close\_a\_game(message, conn)

return True

case "PROMPT":

self.receive\_prompt(message, conn)

return True

case "CANVAS":

self.receive\_canvas(message, conn)

return True

case "ASK\_NEXT":

self.send\_next\_game\_data(message, conn)

return True

case "CHECK\_PIN":

self.check\_room\_pin(message, conn)

return True

case "SEND\_GAME\_RESULT":

self.send\_local\_storage(message, conn)

return True

Данная процедура относится к классу HubServer, который по сути является игровым сервером и хранит у себя все данные об игре. У данного сервера каждого подключение клиента разбивается на отдельный поток, а в каждом потоке как раз и идет постоянное получение нового сообщения командой recv. После этого полученные данные разделяются на команду и данные и передаются функции manageMessages. Если результат функции True или истина, то соедениние с клиентом не разрывается после обработки команды, если же результат False, то после обработки команды и данных соединение будет разорвано, это необходимо для разовой проверки пароля или выхода игрока из игры. Еще будет продемонстрировано несколько процедур, которые вызываются после получения определенной команды, а именно findNickInBase, createARoom, startARoom.

def find\_nick\_in\_base(self, nick, conn):

if nick in self.player\_nicks:

response = "UNAVAILABLE"

else:

response = "AVAILABLE"

conn.send(pickle.dumps(response))

def create\_a\_room(self, conn):

conn.send(pickle.dumps("ACCEPTED"))

size = int.from\_bytes(conn.recv(2048), byteorder="big")

data = b''

while len(data) < size:

data += conn.recv(4096)

room = pickle.loads(data)

if self.\_check\_uniqueness(room):

room\_port = random.randint(self.room\_port\_range[0], self.room\_port\_range[1])

while room\_port in self.used\_ports:

room\_port = random.randint(self.room\_port\_range[0], self.room\_port\_range[1])

self.used\_ports.append(room\_port)

room\_address = (self.IP, room\_port)

room.set\_address(room\_address)

self.active\_rooms.append(room)

conn.send(pickle.dumps(f"CREATED\r\n{room\_address}"))

else:

conn.send(pickle.dumps("EXISTS"))

def start\_a\_room(self, message, conn):

for room in self.active\_rooms:

if message == room.get\_host\_name():

room.start\_room()

room.set\_start\_time(time.time())

self.\_make\_player\_structure(room)

conn.send(pickle.dumps(f"SUCCESS\r\n{room.get\_current()}"))

Процедура findNickInBase отвечает за проверку уникальности никнейма, путём проверки вхождения никнейма в соответствующий массив. После данной процедуры соединение с сервером разрывается, так как необходима только разовая проверка. Процедура createARoom сперва получает от клиента объект класса GameRoom, который содержит всю информацию о созданной комнате: объект класса Player хоста, размеры, приватность и др. Далее процедура createARoom проверяет, уникальна ли комната, чтобы не возникало комнат с одним и тем же хостом, затем результат создания отправляется клиенту через сеть. Процедура startARoom сперва находит комнату по имени хоста в массиве активных комнат, далее стартует комнату и выставляет время старта, для последующей синхронизации, а также создается структура для хранения игровых данных.

Рассмотрим процедуру, отвечающую за создание структуры игровых данных комнаты. Данная процедура называется makePlayerStructure. Вызов происходит в практически в коцне тела процедуры StartARoom.

def \_make\_player\_structure(self, room):

room\_size = room.get\_current()

player\_list = room.get\_player\_list()

info\_list = {player.get\_nickname(): [] for player in player\_list}

for j in range(room\_size):

nicks = []

info = []

for i in range(room\_size):

nicks.append(player\_list[(i + j) % room\_size].get\_nickname())

info.append(None)

info\_list[player\_list[j].get\_nickname()].append(nicks)

info\_list[player\_list[j].get\_nickname()].append(info)

self.local\_storage[room.get\_host\_name()] = info\_list

## Создание комнаты и присоединение к ней

При нажатии на кнопку “Создать комнату” в главном меню программного средства вызывается процедура createARoom. Данная процедура перенаправляет пользователя в новое окно создания, в котором необходимо заполнить несколько полей ввода и нажать кнопку “Создать”. Процесс создания комнаты важная составляющая геймплея, так как игрок может задавать количество игроков, приватность комнаты, также может быть хостом самостоятельно. Код процедуры createARoom будет показан ниже.

def create\_a\_room(self, ev):

if not self.in\_new\_window:

if len(self.room\_list) > 5:

messagebox.showwarning("Can't create", "There is maximum of rooms already(6/6)")

return

self.create\_room\_window = CreateWindow()

self.in\_new\_window = True

self.create\_room\_window.start()

self.wait\_window(self.create\_room\_window)

self.in\_new\_window = False

created\_room = self.create\_room\_window.get\_created\_room()

if not created\_room:

return

self.launching\_room(created\_room)

В показанном отрезке кода происходит создание и получение комнаты от окна создания, после этого необходимо “запустить” комнату, то есть установить все необходимые значения, а также отправить экземпляр комнаты на сервер. Все это происходит в процедуре launchingRoom, в которую передается “новорожденная” комната. Код данной процедуры будет ниже.

def launching\_room(self, created\_room):

created\_room.set\_host\_player(self.player)

created\_room.add\_player(self.player)

self.window\_conn.send(pickle.dumps(f"CREATE\_ROOM\r\n"))

self.window\_conn.recv(1024)

pickled = pickle.dumps(created\_room)

self.window\_conn.sendall(len(pickled).to\_bytes(4, byteorder='big'))

self.window\_conn.sendall(pickled)

server\_response = pickle.loads(self.window\_conn.recv(1024))

message, addr = server\_response.split("\r\n")

self.refresh\_table\_items("important")

if message == "CREATED":

self.withdraw()

room = RoomWindow(created\_room, self.player, self.window\_conn)

room.start()

self.wait\_window(room)

self.deiconify()

self.check\_is\_empty()

self.refresh\_table\_items("important")

Сперва устанавливается хостовый игрок для комнаты, он также добавляется в список игроков комнаты, после этого происходит взаимодействие с серверов, если сервер одобрил создание комнаты, то создается новое окно, в которое передается комната, экземпляр игрока и экземпляр сетевого соединения(socket). Можно заметить, что для передачи объекта используется модуль pickle и его методы loads и dumps, которые нужны для загрузки сериализованных данных и сериализации соответственно.

## Вывод результата

Результат игры выводится на экран сразу после того, как прошла последняя игровая стадия. Процесс вывода результата довольно многоступенчатый, сперва необходимо запросить у сервера игровые данные, далее вывести на экран объект ttk.Canvas, в котором будет происходить отображение, а далее вывести на экран первую “порцию” данных. Следующие данные будут отображаться по нажатию на одну из кнопок, которые изменяют индекс для отображения на +-1. Сейчас будет приведен метод callEnd, с которого и начинается вывод результата.

def call\_end(self):

self.prompt\_lbl['text'] = 'Результаты'

self.paint.place\_forget\_paint()

self.prompt\_input.place\_forget()

self.ready\_btn.place\_forget()

game\_data = self.receive\_game\_results()

self.result\_game\_data = game\_data

if not self.is\_solo\_player(game\_data):

self.print\_data\_on\_screen(game\_data)

В этой процедуре сперва с экрана убираются все элементы игры, а также выводится надпись “Результаты”. Далее от сервера получаются данные, путем вызова функции receiveGameResults. После идет проверка на то, что игрок не единственный в комнате, так как для одного игрока нет смысла выводить результаты. После подтверждения, что игроков больше 1 идет вывод на экран данных, путем вызова метода printDataOnScreen. Ниже будут указаны методы receiveGameResults и printDataOnScreen.

def receive\_game\_results(self):

message = f"SEND\_GAME\_RESULT\r\n{self.room.get\_host\_name()}"

self.room\_conn.send(pickle.dumps(message))

data = b''

while True:

part = self.room\_conn.recv(4096)

data += part

if len(part) < 4096:

break

data = pickle.loads(data)

return data

def print\_data\_on\_screen(self, game\_data):

self.\_generate\_avatars\_dict()

self.left\_swipe\_btn.place(relx=0.9, rely=0.957)

self.right\_swipe\_btn.place(relx=0.935, rely=0.957)

keys = list(game\_data.keys())

scroll\_x, scroll\_y = len(game\_data[keys[0]][1])\*240, len(game\_data[keys[0]][1])\*240

self.\_result\_canvas\_config((0, 0, scroll\_x, scroll\_y))

self.generate\_result(game\_data, self.curr\_page)

В методе receiveGameResults осуществляется запрос на сервер, а после происходит получение больших данных от сервера. Метод printDataOnScreen реализует вывод на экран кнопок перехода к следующей странице информации, вывод объекта ttk.Canvas для отображения результатов, а также его конфигурацию. Вызов метода generateResult необходим для вывода всей информации на подготовленные элементы интерфейса. Код данного метода будет ниже.

def generate\_result(self, game\_data, index):

key = list(game\_data.keys())[index]

mini\_storage = game\_data[key]

self.\_clear\_result\_canvas()

sides = [tk.RIGHT, tk.LEFT]

offset = BASE\_OFFSET\_Y

i = 0

length = len(mini\_storage[0])

data\_index = index + 1

data\_key = list(game\_data.keys())[data\_index % length]

data\_storage = game\_data[data\_key]

while i < length:

nick = mini\_storage[0][i]

self.\_place\_mini\_avatar(sides[i % 2], offset, nick)

data = data\_storage[1][i]

if i % 2 == 0:

self.text\_item = self.result\_canvas.create\_text(0, 0, text=data, font=(FONT\_NAME, 12))

bounds = self.result\_canvas.bbox(self.text\_item)

text\_width = bounds[2] - bounds[0]

x = (LEFT\_AVATAR\_X - text\_width - 20)

self.result\_canvas.create\_text(x, offset + MINI\_AVATAR\_SIZE[0]/4 - 15,

text=data, font=(FONT\_NAME, 12))

self.result\_canvas.delete(self.text\_item)

offset += AVATAR\_SIZE[0] + 25

else:

self.temp\_photos.append(ImageTk.PhotoImage(data.resize(CANVAS\_IMG\_SIZE), master=self.result\_canvas))

self.result\_canvas.create\_image(RIGHT\_AVATAR\_X + MINI\_AVATAR\_SIZE[0], offset, anchor="nw",

image=self.temp\_photos[-1])

offset += AVATAR\_SIZE[0] + 200

i += 1

data\_index += 2

data\_key = list(game\_data.keys())[data\_index % length]

data\_storage = game\_data[data\_key]

Данный метод сперва очищает ttk.Canvas, чтобы после пройтись по структуры игровых данных по индексу, переданному как параметр, и вывести никнеймы и аватары вместе с информацией от этих никнеймов. Вывод картинки и текста различается по своей сути, по этому данные участки кода разделены оператором if, в любом случае переменная смещения offset будет увеличена на некоторое значение, чтобы вывод выглядел красиво. Также в конце каждой итерации цикла структура, откуда берется информация будет изменена, а если быть точнее, то сдвинута на 2 значения по общей игровой информации.

# ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

В ходе тестирования приложения были выявлены некоторые ошибки и недочеты в работе программного средства.

Первый недочёт. На начальных стадиях разработки имело место ошибка при передаче больших объектов через сеть, например, экзмепляров класса Player или GameRoom, которые содержали в себе изображение, являющееся аватаром персонажа. При передаче данных через сеть в данном программном средстве использовалась библиотека pickle, позволяющая сериализовать и десериализовать данные. Таким образом условно сериализовалось 90 кБ, а на стороне сервера принимались только 4096 байт. В таком случае метод pickle loads не мог корректно расшифровать данные и возникала ошибка. Данная проблема была решена введением алгоритма передачи больших объектов через socket.

На ранних этапах была еще одна проблема, суть заключается в том, что если из игровой комнаты выходит хостовый игрок, то возникают некоторые критические проблемы, так как при выходе хост отправляет серверу сообщение об удалении комнаты, и остальные игроки остаются в по сути несуществующей комнате. Данная проблема была решена так, что примерно каждые 0.2-0.3 секунды каждый клиент отправляет на сервер запрос, в котором просит выслать статус комнаты, и если комната уже удалена, то каждый клиент получит в ответ “DELETED”, и тогда самостоятельно закроет окно комнаты и вернет пользователя в главное меню. Подобная проблема имела место и при старте комнаты. Так как игру запускает хост, то остальным игрокам необходимо постоянно спрашивать у сервера состояние комнаты, чтобы игра началась у всех одновременно, а также не происходило локальных сбоев у клиентов. Проблема начала игры одновременно была решена своего рода синхронизацией, завязанной на текущем времени. Сервер записывал время старта комнаты, а далее выставлял необходимую задержку для каждого клиента, чтобы игра началась вовремя. Код процедуры, отвечающей за синхронизацию будет ниже.

def synchronize\_all\_room(self, message, conn):

for room in self.active\_rooms:

if message == room.get\_host\_name():

start\_time = room.get\_start\_time()

time.sleep(start\_time + 0.75 - time.time())

conn.send(pickle.dumps("START"))

Существовала еще одна проблема. Она была связана с игровым таймером, который используется уже в самом игровом процессе, так как пользователю дается ограниченное время на написание текста или рисование картинки. Проблема в том, что сразу после таймера необходимо вызвать метод, который соберет все введенные данные и отправит их на сервер. Так как таймер является объектом класса Timer, то каким-то образом необходимо было внедрить в него объект игрового окна. Проблема была решена с помощью передачи в констуктор Timer ссылки на объект текущего окна, а уже в самом объекте Timer после истечения времени происходит вызов функции окна, то есть callback. Так как стадий может быть 2 и более, то после выполнения всех операций по отправлению данных серверу и получению новых происходит повторный запуск таймера, а после снова вызывается необходимый метод из игрового окна. Код будет оставлен ниже.

class Timer:

def \_\_init\_\_(self, master, time\_ms, mode):

self.master = master

self.time = 0

self.time\_ms = time\_ms

self.mode = mode

if mode == "prompt":

master.prompt\_input.state(['!disabled'])

else:

master.paint.canvas.configure(state="normal")

self.is\_timer\_end = False

self.progressbar = ttk.Progressbar(self.master, orient="horizontal", length=200, mode="determinate")

self.progressbar.pack(pady=10)

def start\_timer(self):

self.progressbar["maximum"] = self.time\_ms \* 10

self.time += 1

self.progressbar["value"] = self.time \* 2

if self.time < self.time\_ms \* 2 \* 2.5:

self.master.after(200, self.start\_timer)

elif self.time == self.time\_ms \* 2 \* 2.5:

self.is\_timer\_end = True

self.callback(self.master)

def callback(self, master):

result = None

self.progressbar.pack\_forget()

if self.mode == "prompt":

master.prompt\_input.state(['disabled'])

result = master.prompt\_input.get()

elif self.mode == "canvas":

master.paint.canvas.configure(state="disabled")

result = master.paint.capture\_img()

master.after\_timer(result, self.mode)

А вот это уже метод в классе GameWindow, который вызывается из Timer.

def after\_timer(self, message, mode):

self.ready\_btn['text'] = "Готово"

self.prompt\_input.state(['!disabled'])

if mode == "prompt":

self.prompt\_call(message)

elif mode == "canvas":

self.paint\_call(message)

Самое главное в тестировании, это исправить все проблемы и ошибки в программе до того, как она выйдет в свет. Можно считать, что в данном случае задача тестирования была выполнена успешно.

# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## 5.1 Интерфейс программного средства

5.1.1 Начальное окно

Начальное окно представляет собой небольшое окно, состоящее из картинки, кнопки переключения на следущюю, кнопки выбора картинки из файлов, поля ввода никнейма и кнопки подтвердить. Никнейм может состоять только из английких символов и длинной не более 15 символов. Внешний вид начального окна приложения представлен на рисунке 5.1.

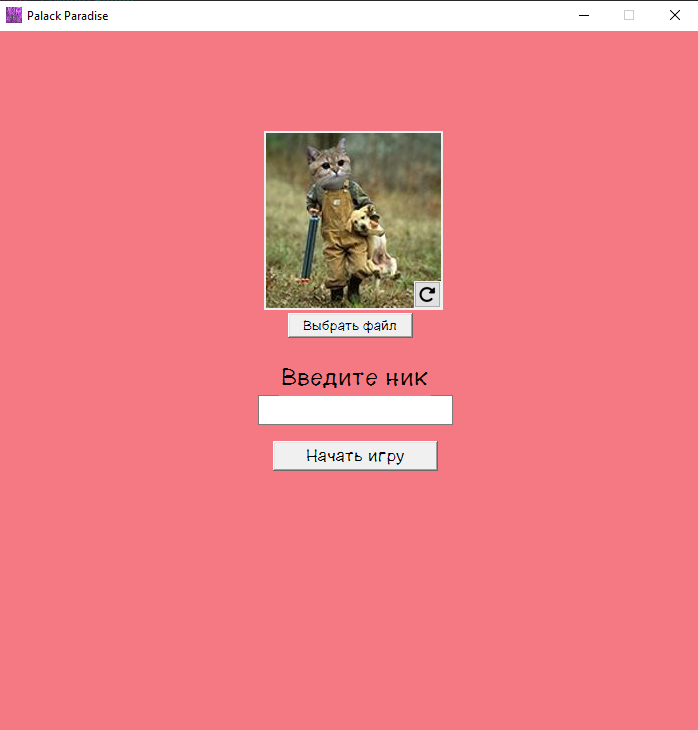


Рисунок 5.1 – Начальное окно приложения

5.1.2 Главное окно

Главное окно представляет собой своеобразную общую комнату, в которой расположены аватар игрока и его никнейм, таблица активных комнат, кнопка “Присоединиться”, позволяющая присоединитья к выбранной комнате, кнопка “Создать комнату”, открывающая новое окно создания комнаты, кнопка “Обновить”, которая посылает серверу запрос на обновление, а после выводит полученные результаты в таблицу. Внешний вид продемонстрирован на картинке 5.2.

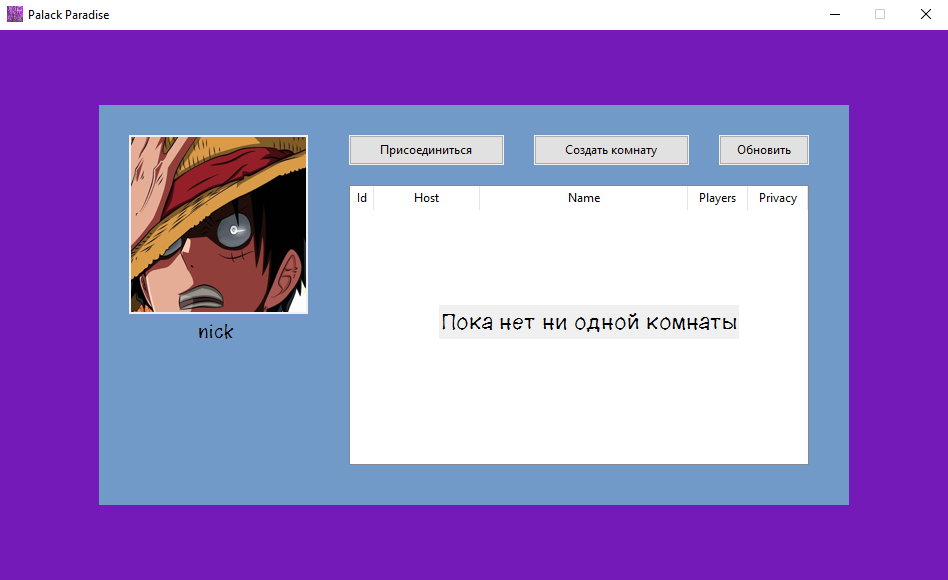


Рисунок 5.2 – Главное окно приложения

5.1.3 Создание комнаты

Кнопка “Создать комнату” из главного меню отправляет пользователя в новое окно создания, которое содержит поле ввода для описания комнаты, выпадающей список для определения размеров, выпадающий список для определения режима приватности, выпадающий список для определения игрового режима комнаты, поле ввода для пароля и индикатор скрытия пароля, если режим приватности “private”. Внешний вид окна создания представлен на рисунке 5.3.

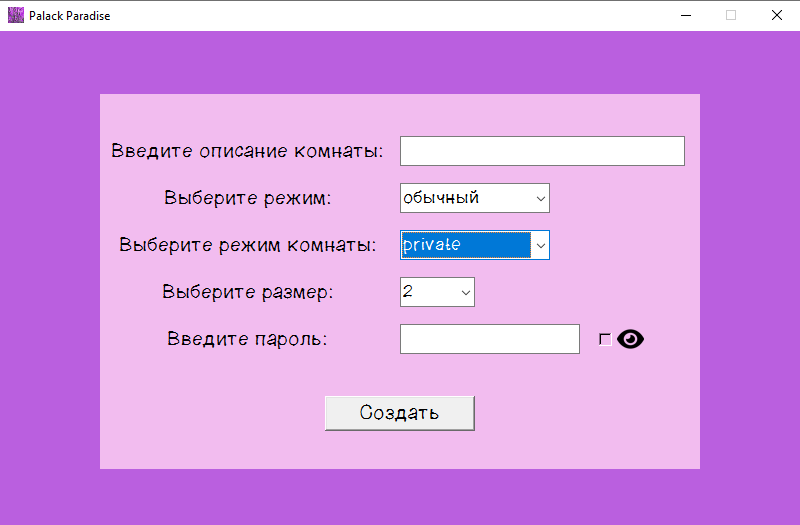


Рисунок 5.3 – Окно создания комнаты

5.1.4 Окно комнаты

Игрок попадает в новое окно после нажатия на кнопку “Присоединиться” в главном окне. Создается новое окно, в котором отображается текущий состав комнаты, а именно игроки и их аватары, а также индикатор хоста комнаты. Также хост на своем окне обязательно имеет кнопку “Начать игру”, которая начнет игровую сессию. Внешний вид окна комнаты о разработчике представлен на рисунке 5.4.

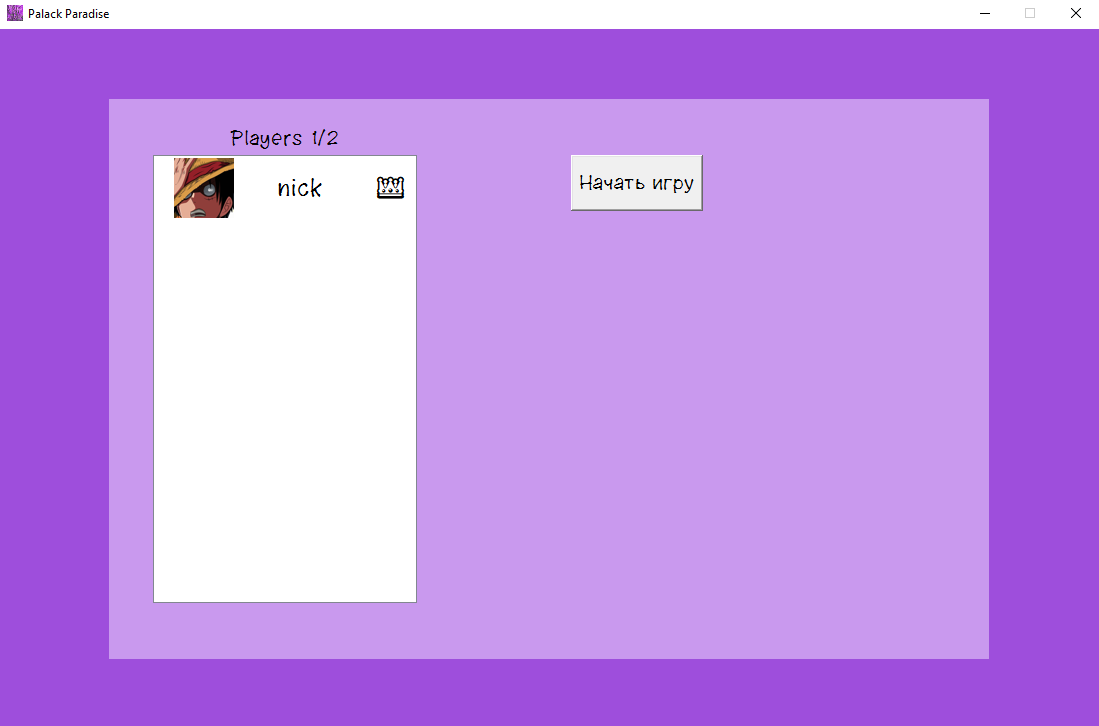


Рисунок 5.4 – Окно комнаты

**5.1.5** Окно игры

Окно игры по внешнему виду не сильно отличается от окна комнаты, однако содержит несколько новых элементов интерфейса. В окне игры появляется поле для ввода текста, также кнока “Готов”, которая заблокирует доступ к полю вводу по нажатию, также на одной из игровых стадий появляется своеобразный аналог Paint, который используется для рисования изображений, кнопка “Готов” работает и на Paint. А после завершения игры появится область в центре экрана, куда будут выведены результаты игры. Внешний вид окна игры можно посмотреть на рисунке 5.5.

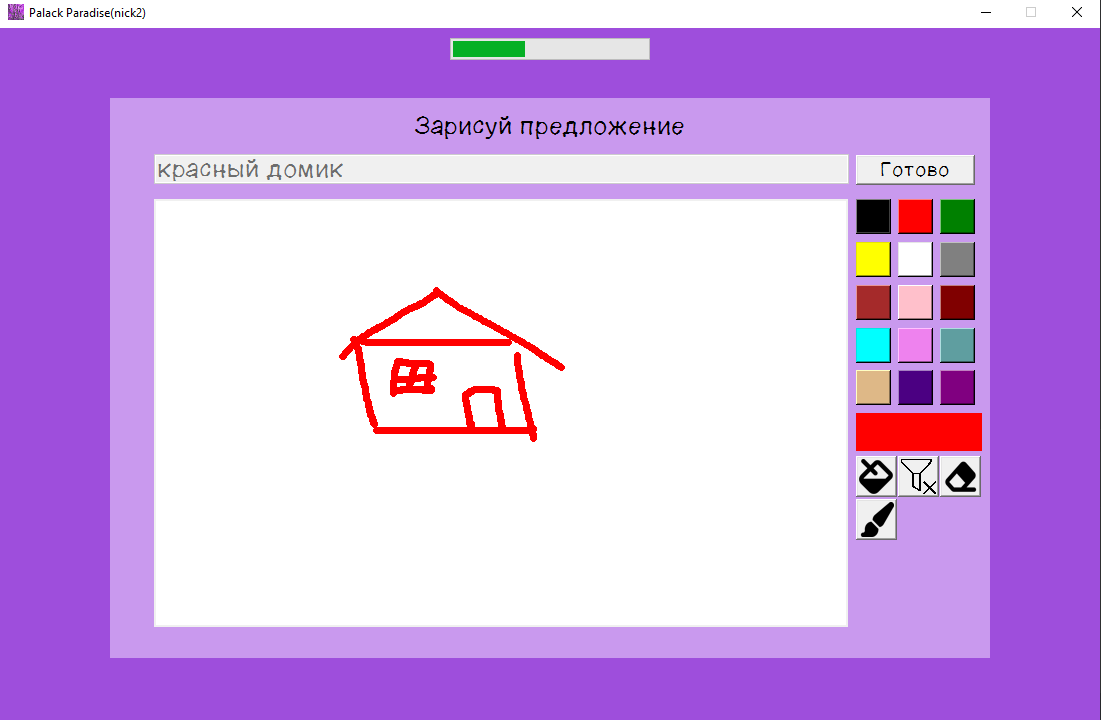


Рисунок 5.5 — Внешний вид игрового окна

## 5.2 Управление программным средством

Попасть в комнату можно 2 путями: нажать на кнопку “Создать комнату”, после чего в меню создания нажать “Создать”, нажать на кнопку “Присоединиться”. Но сперва необходимо попасть в главное меню, для этого необходимо ввести никнейм длинной более 4 символов и менее 15, а после нажать на кнопку “Начать игру”, если же всплывает окно с текстом “Никней уже занят”, то необходимо ввести новый никнейм. Практически во всех окнах клавиша “Enter” отвечает за самые главные кнопки.

После попадания в комнату необходимо просто ждать начала, если вы не хост, или нажать на кнопку “Начать игру”, если вы хост. После старта игры в верхней части экрана запустится зеленая полоска таймера, а также появится поле для ввода любого приложения. Пользователю необходимо ввести какой-либо текст и подождать конца таймера, если же оставить графу пустой, то сообщение автоматически получит значение “no prompt input”. Далее начнется следующая стадия, в которой появится аналог Paint. Необходимо нарисовать то, что описано в предложении с помощью средств рисования, а именно кнопок выбора цвета, кнопки изменения размеров кисти, кнопки очистки области для рисования, кнопки включения ластика, которая меняет текущий цвет на белый, кнопки заливки, которая зальет весь холст текущим цветом. По истечению таймера холст автоматически будет отправлен на сервер, а пользователь получит новое изображение и задачей будет описать изображение. Таким образом процесс будет протекать, пока количество раундов не истечет. После этого начнется подведение результатов.

Сперва появится новый холст, но которым будут выведены результаты одной ветки. Условно первый игрок написал определенный текст, второй его получил и описал, а третий получил картинку от второго и описал её. Все это вместе с данными будет выведено на экран, а справа снизу появятся 2 кнопки: стрелка влево и стрелка вправо. По нажатию на одну из них будет выведена следующая ветка результатов, таким образом возможно просмотреть все результаты.

# ЗАКЛЮЧЕНИЕ

В заключении можно сказать, что была проделана хорошая работа по созданию данного программного средства. Взаимодействие с сервером является довольно непростой, но интересной задачей, выполнение которой несомненно будет хорошим опытом. Также изучение языка python и некоторых библиотек в целом является хорошим опытом для начинающего программиста. В рамках данного курсового проекта было разработано программное средство “Palack”, которое позволит пользователям операционной системы Windows играть в данную игру. При разработке данного средства были успешно выполнены следующие поставленные задачи:

* реализации серверного взаимодействия;
* генерация структуры для игровых данных;
* грамотное взаимодействие между окнами;
* корректная передача объектов через socket;
* вывод данных на экран в приятном глазу виду;
* разделение сервера на потоки;
* перезапуск игры.

Для успешного выполнения поставленных задач потребовалось изучить

основные принципы работы с сетью и способами передачи данных через сеть, библиотеками пользовательского интерфейса, созданием потоков в приложении, динамическим созданием компонентов. Были изучены библиотеки pickle, tkinter, socket и др.

Существует множество способов улучшить данное программное средство: добавить больше игровых режимов, добавить звуковое сопровождение, улучшить оптимизацию и уменьшить количество передаваемой информации, добавить базу данных вместо простых массивов на серверной стороне.

Игровое средство “Palack” позволяет компаниям людям весело и с интересом провести время за игрой.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Глухова, Л. А. Основы алгоритмизации и программирования: Лаб. практикум для студ. спец. I-40 01 01 «Программное обеспечение информационных технологий» дневной формы обуч. В 4 ч. / Л. А. Глухова, Е. Е. Фадеева, Е. П. Фадеева. – Минск: БГУИР, 2007. – Ч. 3. – 51 с.

[2] Артур М. SFML Game Development. – Birmingham, 2013. – 296c.

[3] Майкл Д. Изучаем C++ через программирование игр. – СПб.: Питер. 2015. – 450с.

[4] Орлов, С. А. Технологии разработки программного обеспечения: учеб. Пособие. – СПб, 2003. – 321 с.

[5] Уилсон, С. Принципы проектирования и разработки программного обеспечения, yчебн. курс. – СПб, 2003. – 570 с.

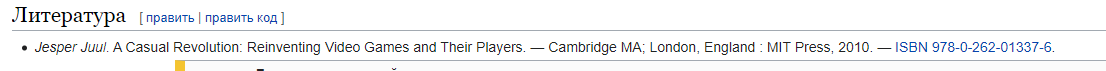
[6] https://www.sfml-dev.org/documentation/2.5.1/ – Документация к SFML.

[7] Петцольд 4. Programming Windows®, 1998. - СПб.: БХВ-Петербург, 2007. – 1448 c.

[8] Алгоритмы. Теория и практические применение / Род Стивенс. - Москва, 2016. – 544 с.

[9] Мартин Р. Чистый код: создание, анализ и рефакторинг. Библиотека программиста. - СПб.: Питер. 2018. – 464 с.

[10] Пол А. Объектно-ориентированное программирование на С++. – Москва, 2017. – 476с.



# ПРИЛОЖЕНИЕ А

**Исходный код программы**

from start\_window import StartWindow

from hub\_window import HubWindow

import socket

import pickle

class Client:

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

self.server\_ip = '192.168.1.104'

self.server\_port = 5050

self.server\_addr = (self.server\_ip, self.server\_port)

self.start\_window = StartWindow(self.server\_addr)

self.hub\_window = None

self.curr\_player = None

self.socket = None

self.encoding\_format = 'utf-8'

def start\_program(self):

self.start\_window.start()

self.start\_window.mainloop()

self.curr\_player = self.start\_window.get\_created\_player()

if self.curr\_player:

self.handle\_hub()

def handle\_hub(self):

self.connection\_to\_server()

self.hub\_window = HubWindow(self.curr\_player, self.socket)

self.hub\_window.start()

self.hub\_window.mainloop()

def send\_welcome\_message(self, conn):

message = "HELLO\r\n" + self.curr\_player.get\_nickname()

conn.send(pickle.dumps(message))

def connection\_to\_server(self):

self.socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

self.socket.connect((self.server\_ip, self.server\_port))

self.send\_welcome\_message(self.socket)

myClient = Client()

myClient.start\_program()

import tkinter as tk

import tkinter.ttk as ttk

from settings import \*

from support import center, is\_pin\_valid, is\_description\_valid

from PIL import ImageTk, Image

from game\_room import GameRoom

from tkinter import messagebox

class CreateWindow(tk.Toplevel):

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

super().\_\_init\_\_()

# window

self.title("Palack Paradise")

self.geometry(f'{CREATE\_WIDTH}x{CREATE\_HEIGHT}')

self.resizable(width=False, height=False)

self.iconbitmap('../graphics/icons/main\_ico.ico')

center(self)

self.configure(background=CREATE\_BG\_COLOR, padx=PADDING, pady=PADDING)

self.half\_height = CREATE\_HEIGHT / 2

self.half\_width = CREATE\_WIDTH / 2

self.protocol("WM\_DELETE\_WINDOW", self.on\_closing)

self.created\_room = None

def return\_pressed(ev):

self.create()

self.bind("<Return>", return\_pressed)

"""

Frame

"""

s = ttk.Style(self)

s.configure('MyCreate.TFrame', background=FRAME\_COLOR)

# s.configure('Test.TFrame', background="black")

self.bg\_frame = ttk.Frame(self, borderwidth=BORDER, style="MyCreate.TFrame")

self.lbl\_frame = ttk.Frame(self.bg\_frame, borderwidth=BORDER, style="MyCreate.TFrame")

self.entry\_frame = ttk.Frame(self.bg\_frame, borderwidth=BORDER, style="MyCreate.TFrame")

"""

Text Entries

"""

self.check\_description = (self.register(is\_description\_valid), "%P")

self.room\_description\_input = ttk.Entry(self.entry\_frame, font=(FONT\_NAME, NAME\_INPUT\_FONT\_SIZE),

validate="key",

validatecommand=self.check\_description)

self.check = (self.register(is\_pin\_valid), "%P")

self.pin\_input = ttk.Entry(self.entry\_frame, font=(FONT\_NAME, NAME\_INPUT\_FONT\_SIZE), validate="key",

validatecommand=self.check)

"""

Combo boxes

"""

self.room\_size\_input = ttk.Combobox(self.entry\_frame, values=("2", "3", "4", "5"),

font=(FONT\_NAME, SIZE\_INPUT\_FONT\_SIZE), state="readonly")

self.room\_size\_input.current(0)

self.room\_private\_cfg = ttk.Combobox(self.entry\_frame, values=("private", "public"),

font=(FONT\_NAME, SIZE\_INPUT\_FONT\_SIZE), state="readonly")

self.room\_private\_cfg.current(1)

self.room\_private\_cfg.bind("<<ComboboxSelected>>", self.change\_is\_private)

self.mode\_chooser = ttk.Combobox(self.entry\_frame, values=["обычный"], font=(FONT\_NAME, SIZE\_INPUT\_FONT\_SIZE),

state="readonly")

self.mode\_chooser.current(0)

"""

Labels

"""

self.lbl\_font = (FONT\_NAME, LBL\_FONT\_SIZE)

self.room\_name\_lbl = ttk.Label(self.lbl\_frame, text="Введите описание комнаты: ", font=self.lbl\_font,

background=FRAME\_COLOR)

self.room\_size\_lbl = ttk.Label(self.lbl\_frame, text="Выберите размер: ", font=self.lbl\_font,

background=FRAME\_COLOR)

self.room\_private\_lbl = ttk.Label(self.lbl\_frame, text="Выберите режим комнаты: ", font=self.lbl\_font,

background=FRAME\_COLOR)

self.pin\_lbl = ttk.Label(self.lbl\_frame, text="Введите пароль: ", font=self.lbl\_font, background=FRAME\_COLOR)

self.mode\_chooser\_lbl = ttk.Label(self.lbl\_frame, text="Выберите режим: ",

font=self.lbl\_font, background=FRAME\_COLOR)

"""

Other

"""

eye\_width = int(EYE\_ICO\_SIZE \* 5 / 4)

self.opened\_eye = ImageTk.PhotoImage(Image.open('../graphics/utility/opened\_eye.png').

resize((eye\_width, EYE\_ICO\_SIZE)), master=self)

self.closed\_eye = ImageTk.PhotoImage(Image.open('../graphics/utility/closed\_eye.png').

resize((eye\_width, EYE\_ICO\_SIZE)), master=self)

val = tk.IntVar(value=1)

self.visible\_password\_checkbox = tk.Checkbutton(self.entry\_frame, image=self.closed\_eye, variable=val,

bg=FRAME\_COLOR, command=self.check\_visibility,

selectcolor=FRAME\_COLOR, activebackground=FRAME\_COLOR)

self.confirm\_button = tk.Button(self.bg\_frame, text="Создать", font=self.lbl\_font, command=self.create)

self.is\_private = False

self.is\_password\_visible = False

self.change\_is\_private(None)

self.check\_visibility()

def get\_created\_room(self):

return self.created\_room

def is\_fields\_fulfill(self):

is\_fulfill = True

is\_fulfill = is\_fulfill and (len(self.room\_description\_input.get()) > 0) and \

(self.is\_private and (len(self.pin\_input.get()) > 0)

or (not self.is\_private and True))

return is\_fulfill

def create(self):

if self.is\_fields\_fulfill():

self.created\_room = GameRoom(None, None, self.room\_description\_input.get(), self.room\_size\_input.get(),

self.is\_private, pin=self.pin\_input.get() if self.is\_private else None)

self.destroy()

else:

messagebox.showwarning("Ошибка ввода", "Заполните все поля")

def check\_visibility(self):

self.is\_password\_visible = not self.is\_password\_visible

if self.is\_password\_visible:

self.pin\_input.config(show="")

self.visible\_password\_checkbox.configure(image=self.opened\_eye)

else:

self.pin\_input.config(show="\*")

self.visible\_password\_checkbox.configure(image=self.closed\_eye)

def change\_is\_private(self, ev):

value = self.room\_private\_cfg.get()

relheight = ALL\_FIELDS\_HEIGHT / FRAME\_HEIGHT + 0.05

self.is\_private = True if value == "private" else False

if self.is\_private:

self.pin\_input.place(relx=0, rely=0.15 + relheight \* 4, height=ALL\_FIELDS\_HEIGHT, relwidth=0.6, anchor="w")

self.pin\_lbl.place(relx=.5, rely=0.15 + relheight \* 4, height=ALL\_FIELDS\_HEIGHT, anchor="center")

self.visible\_password\_checkbox.place(relx=0.65, rely=0.11 + relheight \* 4)

else:

self.pin\_input.place\_forget()

self.pin\_input.delete(0, 'end')

self.pin\_lbl.place\_forget()

self.visible\_password\_checkbox.place\_forget()

def start(self):

relheight = ALL\_FIELDS\_HEIGHT / FRAME\_HEIGHT + 0.05

self.bg\_frame.place(relx=.5, rely=.5, relwidth=0.75, relheight=0.75, anchor="center")

# labels

self.lbl\_frame.place(relx=0, rely=.5, relwidth=0.5, relheight=1, anchor="w")

self.room\_name\_lbl.place(relx=.5, rely=0.15, height=ALL\_FIELDS\_HEIGHT, anchor="center")

self.room\_size\_lbl.place(relx=.5, rely=0.15 + relheight \* 3, height=ALL\_FIELDS\_HEIGHT, anchor="center")

self.room\_private\_lbl.place(relx=.5, rely=0.15 + relheight \* 2, height=ALL\_FIELDS\_HEIGHT, anchor="center")

self.mode\_chooser\_lbl.place(relx=.5, rely=0.15 + relheight, height=ALL\_FIELDS\_HEIGHT, anchor="center")

# entries

self.entry\_frame.place(relx=1, rely=.5, relwidth=0.5, relheight=1, anchor="e")

self.room\_description\_input.place(relx=0, rely=0.15, height=ALL\_FIELDS\_HEIGHT, relwidth=0.95, anchor="w")

self.room\_size\_input.place(relx=0, rely=0.15 + relheight \* 3, relwidth=0.25, height=ALL\_FIELDS\_HEIGHT,

anchor="w")

self.room\_private\_cfg.place(relx=0, rely=0.15 + relheight \* 2, height=ALL\_FIELDS\_HEIGHT, relwidth=0.5,

anchor="w")

self.mode\_chooser.place(relx=0, rely=0.15 + relheight, height=ALL\_FIELDS\_HEIGHT, relwidth=0.5, anchor="w")

# confirm btn

self.confirm\_button.place(relx=.5, rely=0.85, height=35, anchor="center", relwidth=0.25)

def on\_closing(self):

self.destroy()

class GameRoom:

def \_\_init\_\_(self, host, address, description, capacity, is\_private=False, mode="default", pin=None):

self.host\_player = host

self.capacity = capacity

self.address = address

self.room\_description = description

self.room\_mode = mode

self.room\_pin = pin

self.is\_room\_private = is\_private

self.player\_list = []

self.is\_started = False

self.start\_time = None

def start\_room(self):

self.is\_started = True

def set\_start\_time(self, start\_time):

self.start\_time = start\_time

def get\_start\_time(self):

return self.start\_time

def stop\_room(self):

self.is\_started = False

def is\_room\_started(self):

return self.is\_started

def get\_player\_list(self):

return self.player\_list

def set\_player\_list(self, player\_list):

self.player\_list = player\_list

def get\_address(self):

return self.address

def set\_address(self, addr):

self.address = addr

def set\_host\_player(self, player):

self.host\_player = player

def get\_host\_player(self):

return self.host\_player

def get\_description(self):

return self.room\_description

def get\_size(self):

return self.capacity

def get\_current(self):

return len(self.player\_list)

def get\_host\_name(self):

return self.host\_player.get\_nickname()

def add\_player(self, player):

if player not in self.player\_list:

self.player\_list.append(player)

def is\_private(self):

return self.is\_room\_private

def check\_pin(self, pin):

return pin == self.room\_pin

def \_\_repr\_\_(self):

return f"[ROOM] {self.room\_description}; [HOST] {self.host\_player.get\_nickname()}; size: {self.capacity};" \

f"privacy: " + ("private;" if self.is\_room\_private else "public;") + "\n"

import pickle

from PIL import ImageTk, Image

from support import is\_prompt\_valid

import tkinter as tk

from tkinter import ttk, messagebox

from support import center

from settings import \*

from Timer import Timer

from Paint import Paint

class GameWindow(tk.Toplevel):

def \_\_init\_\_(self, room, player, conn):

super().\_\_init\_\_()

# window

self.player = player

self.title(f"Palack Paradise({self.player.get\_nickname()})")

self.geometry(f'{ROOM\_WIDTH}x{ROOM\_HEIGHT}')

center(self)

self.iconbitmap('../graphics/icons/main\_ico.ico')

self.resizable(width=False, height=False)

self.configure(background=ROOM\_BG, padx=PADDING, pady=PADDING)

self.half\_height = HEIGHT\_HUB / 2

self.half\_width = WIDTH\_HUB / 2

self.room = room

self.room\_conn = conn

self.protocol("WM\_DELETE\_WINDOW", self.on\_closing)

"""GUI"""

s = ttk.Style(self)

s.configure('Game.TFrame', background=ROOM\_FRAME)

self.frame = ttk.Frame(self, borderwidth=BORDER, padding=(PADDING, PADDING), style="Game.TFrame")

self.check = (self.register(is\_prompt\_valid), "%P")

self.prompt\_input = ttk.Entry(self.frame, font=(FONT\_NAME, 18), validate="key", validatecommand=self.check)

self.prompt\_lbl = ttk.Label(self.frame, text="Напиши предложение", background=ROOM\_FRAME, font=(FONT\_NAME, 18))

self.ready\_btn = tk.Button(self.frame, text="Готово", font=(FONT\_NAME, 14), command=self.on\_ready\_click)

self.is\_ready = False

self.timer = None

self.paint = Paint(self.frame)

self.paint.place\_forget\_paint()

self.room\_stages = 0

self.current\_stage = 0

self.prompt\_time = START\_PROMPT\_TIME

self.canvas\_time = START\_CANVAS\_TIME

self.result\_canvas = tk.Canvas(self.frame, bg='#f2bcef', width=200, height=300)

self.canvas\_scroll = ttk.Scrollbar(self.frame, orient=tk.VERTICAL)

self.players = {}

self.temp\_photos = []

self.left\_swipe\_img = ImageTk.PhotoImage(Image.open('../graphics/utility/left\_swipe.png').

resize((NEXT\_ICO\_SIZE, NEXT\_ICO\_SIZE)), master=self)

self.right\_swipe\_img = ImageTk.PhotoImage(Image.open('../graphics/utility/right\_swipe.png').

resize((NEXT\_ICO\_SIZE, NEXT\_ICO\_SIZE)), master=self)

self.left\_swipe\_btn = tk.Button(self.frame, image=self.left\_swipe\_img, command=self.left\_swipe)

self.right\_swipe\_btn = tk.Button(self.frame, image=self.right\_swipe\_img, command=self.right\_swipe)

self.curr\_page = 0

self.result\_game\_data = None

self.mode = None

"""

Pre-start

"""

self.check\_room\_status()

def left\_swipe(self):

if self.curr\_page > 0:

self.curr\_page -= 1

self.generate\_result(self.result\_game\_data, self.curr\_page)

def right\_swipe(self):

if self.curr\_page < self.room.get\_current() - 1:

self.curr\_page += 1

self.generate\_result(self.result\_game\_data, self.curr\_page)

def \_generate\_avatars\_dict(self):

for player in self.room.get\_player\_list():

self.players[player.get\_nickname()] = ImageTk.PhotoImage(player.get\_avatar\_image()[1].

resize(AVATAR\_SIZE),

master=self.frame)

def on\_ready\_click(self):

if self.mode == "prompt":

if self.is\_ready:

self.prompt\_input.state(['!disabled'])

self.ready\_btn['text'] = "Готово"

self.is\_ready = False

else:

self.is\_ready = True

self.prompt\_input.state(['disabled'])

self.ready\_btn['text'] = "Изменить"

else:

if self.is\_ready:

self.paint.unsleep\_canvas()

self.ready\_btn['text'] = "Готово"

self.is\_ready = False

else:

self.is\_ready = True

self.paint.sleep\_canvas()

self.prompt\_input.state(['disabled'])

self.ready\_btn['text'] = "Изменить"

def on\_closing(self):

if self.player.get\_nickname() == self.room.get\_host\_name():

if messagebox.askokcancel("Quit", "Do you want to quit?"):

self.room\_conn.send(pickle.dumps(f"STOP\_GAME\r\n{self.player.get\_nickname()}"))

server\_response = pickle.loads(self.room\_conn.recv(1024))

if server\_response == "CLOSED":

self.destroy()

def check\_room\_status(self):

if self.player.get\_nickname() != self.room.get\_host\_name():

message = f"CHECK\_STATUS\r\n{self.room.get\_host\_name()}"

self.room\_conn.send(pickle.dumps(message))

response = pickle.loads(self.room\_conn.recv(1024))

if response == "CLOSED":

self.destroy()

self.after(200, self.check\_room\_status)

def after\_timer(self, message, mode):

self.ready\_btn['text'] = "Готово"

self.prompt\_input.state(['!disabled'])

if mode == "prompt":

self.prompt\_call(message)

elif mode == "canvas":

self.paint\_call(message)

def ask\_next\_part(self, mode):

message = f"ASK\_NEXT\r\n{self.room.get\_host\_name()},{self.player.get\_nickname()},{self.current\_stage}"

self.room\_conn.send(pickle.dumps(message))

data = b''

while True:

part = self.room\_conn.recv(4096)

data += part

if len(part) < 4096:

break

data = pickle.loads(data)

if mode == "prompt":

self.prompt\_input.state(['!disabled'])

self.prompt\_input.delete("0", tk.END)

self.prompt\_input.insert("0", data)

self.prompt\_input.state(['disabled'])

elif mode == "canvas":

self.prompt\_input.delete("0", tk.END)

self.paint.set\_canvas\_image(data)

def receive\_game\_results(self):

message = f"SEND\_GAME\_RESULT\r\n{self.room.get\_host\_name()}"

self.room\_conn.send(pickle.dumps(message))

data = b''

while True:

part = self.room\_conn.recv(4096)

data += part

if len(part) < 4096:

break

data = pickle.loads(data)

return data

def is\_solo\_player(self, data):

for key in data.keys():

return len(data[key][0]) < 2

def \_result\_canvas\_config(self, scrollregion):

self.canvas\_scroll.pack(side=tk.RIGHT, fill=tk.Y)

self.canvas\_scroll.config(command=self.result\_canvas.yview)

self.result\_canvas.config(width=300, height=300)

self.result\_canvas.config(scrollregion=scrollregion)

self.result\_canvas.config(yscrollcommand=self.canvas\_scroll.set)

self.result\_canvas.place(relx=0.05, rely=0.15, relheight=0.8, relwidth=0.9)

def \_clear\_result\_canvas(self):

self.result\_canvas.delete('all')

self.result\_canvas['bg'] = '#f2bcef'

def \_place\_mini\_avatar(self, side, offset\_y, nick):

nick = nick

if side == tk.RIGHT:

self.result\_canvas.create\_image(LEFT\_AVATAR\_X, offset\_y, image=self.players[nick])

self.result\_canvas.create\_text(LEFT\_AVATAR\_X, offset\_y + MINI\_AVATAR\_SIZE[0] - 5, anchor="center",

text=nick, font=BASE\_FONT)

elif side == tk.LEFT:

self.result\_canvas.create\_image(RIGHT\_AVATAR\_X, offset\_y, image=self.players[nick])

self.result\_canvas.create\_text(RIGHT\_AVATAR\_X, offset\_y + MINI\_AVATAR\_SIZE[0] - 5, anchor="center",

text=nick, font=BASE\_FONT)

def \_find\_nick\_by\_stage(self, stage, game\_data, nick):

for key in game\_data.keys():

if game\_data[key][0][stage] == nick:

return key

def generate\_result(self, game\_data, index):

key = list(game\_data.keys())[index]

mini\_storage = game\_data[key]

self.\_clear\_result\_canvas()

sides = [tk.RIGHT, tk.LEFT]

offset = BASE\_OFFSET\_Y

i = 0

length = len(mini\_storage[0])

data\_index = index + 1

data\_key = list(game\_data.keys())[data\_index % length]

data\_storage = game\_data[data\_key]

while i < length:

nick = mini\_storage[0][i]

self.\_place\_mini\_avatar(sides[i % 2], offset, nick)

data = data\_storage[1][i]

if i % 2 == 0:

self.text\_item = self.result\_canvas.create\_text(0, 0, text=data, font=(FONT\_NAME, 12))

bounds = self.result\_canvas.bbox(self.text\_item)

text\_width = bounds[2] - bounds[0]

x = (LEFT\_AVATAR\_X - text\_width - 20)

self.result\_canvas.create\_text(x, offset + MINI\_AVATAR\_SIZE[0]/4 - 15,

text=data, font=(FONT\_NAME, 12))

self.result\_canvas.delete(self.text\_item)

offset += AVATAR\_SIZE[0] + 25

else:

self.temp\_photos.append(ImageTk.PhotoImage(data.resize(CANVAS\_IMG\_SIZE), master=self.result\_canvas))

self.result\_canvas.create\_image(RIGHT\_AVATAR\_X + MINI\_AVATAR\_SIZE[0], offset, anchor="nw",

image=self.temp\_photos[-1])

offset += AVATAR\_SIZE[0] + 200

i += 1

data\_index += 2

data\_key = list(game\_data.keys())[data\_index % length]

data\_storage = game\_data[data\_key]

def print\_data\_on\_screen(self, game\_data):

self.\_generate\_avatars\_dict()

self.left\_swipe\_btn.place(relx=0.9, rely=0.957)

self.right\_swipe\_btn.place(relx=0.935, rely=0.957)

keys = list(game\_data.keys())

scroll\_x, scroll\_y = len(game\_data[keys[0]][1])\*240, len(game\_data[keys[0]][1])\*240

self.\_result\_canvas\_config((0, 0, scroll\_x, scroll\_y))

print(game\_data)

self.generate\_result(game\_data, self.curr\_page)

def call\_end(self):

self.prompt\_lbl['text'] = 'Результаты'

self.paint.place\_forget\_paint()

self.prompt\_input.place\_forget()

self.ready\_btn.place\_forget()

game\_data = self.receive\_game\_results()

self.result\_game\_data = game\_data

if not self.is\_solo\_player(game\_data):

self.print\_data\_on\_screen(game\_data)

def \_find\_next\_in\_list(self, player\_list):

for player in player\_list:

if player.get\_nickname() == self.player.get\_nickname():

return player\_list.index(player) + self.current\_stage

def prompt\_call(self, message):

player\_list = self.room.get\_player\_list()

name = player\_list[self.\_find\_next\_in\_list(player\_list) % len(player\_list)].get\_nickname()

message = message if len(message) > 0 else "no prompt input"

self.room\_conn.send(pickle.dumps(f"PROMPT\r\n{self.room.get\_host\_name()},{name}"))

response = pickle.loads(self.room\_conn.recv(1024))

if response == "FOUND":

self.room\_conn.send(pickle.dumps(f"{message}"))

pickle.loads(self.room\_conn.recv(1024))

self.prompt\_lbl['text'] = "Зарисуй предложение"

if self.current\_stage >= self.room\_stages:

self.after(200)

self.call\_end()

else:

self.after(300)

self.is\_ready = False

self.mode = "canvas"

self.ask\_next\_part("prompt")

self.paint.place\_paint()

self.paint.unsleep\_canvas()

self.timer = Timer(self, self.canvas\_time-3\*self.current\_stage, "canvas")

self.timer.start\_timer()

self.current\_stage += 1

def paint\_call(self, message):

player\_list = self.room.get\_player\_list()

name = player\_list[self.\_find\_next\_in\_list(player\_list) % len(player\_list)].get\_nickname()

self.room\_conn.send(pickle.dumps(f"CANVAS\r\n{self.room.get\_host\_name()},{name}"))

response = pickle.loads(self.room\_conn.recv(1024))

if response == "FOUND":

message\_pickled = pickle.dumps(message)

self.room\_conn.send(message\_pickled)

pickle.loads(self.room\_conn.recv(1024))

self.prompt\_lbl['text'] = "Опиши картинку"

if self.current\_stage >= self.room\_stages:

self.after(200)

self.call\_end()

else:

self.after(300)

self.is\_ready = False

self.mode = "prompt"

self.ask\_next\_part("canvas")

self.paint.sleep\_canvas()

self.timer = Timer(self, self.prompt\_time-2\*self.current\_stage, "prompt")

self.timer.start\_timer()

self.current\_stage += 1

def start\_game(self):

self.timer = Timer(self, self.prompt\_time, "prompt")

# self.timer = Timer(self, 4, "prompt")

self.timer.start\_timer()

self.mode = "prompt"

self.current\_stage += 1

def placing(self):

self.frame.place(relx=.5, rely=.5, relwidth=0.8, relheight=0.8, anchor="center")

self.prompt\_lbl.place(relx=.5, rely=0.05, anchor="center")

self.prompt\_input.place(relx=0.05, rely=0.10, relwidth=0.79, height=30)

self.ready\_btn.place(relx=0.915, rely=0.128, height=30, relwidth=0.135, anchor="center")

def wait\_for\_start(self):

message = f"ASK\_START\r\n{self.room.get\_host\_name()}"

self.room\_conn.send(pickle.dumps(message))

response = pickle.loads(self.room\_conn.recv(1024))

if response == "START":

self.start\_game()

def start(self):

self.placing()

self.room\_stages = self.room.get\_current()

self.current\_stage = 0

self.wait\_for\_start()

import random

import time

import socket

from support import get\_ip

import threading

import pickle

class HubServer:

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

# net data

PORT = 5050

self.IP = get\_ip()

self.header = 64

self.encode\_format = "utf-8"

self.ADDR = (self.IP, PORT)

self.used\_ports = []

self.room\_port\_range = [50000, 60000]

# server binding

self.server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

self.server.bind(self.ADDR)

self.active\_rooms = []

self.local\_storage = {}

self.player\_nicks = []

def \_make\_player\_structure(self, room):

room\_size = room.get\_current()

player\_list = room.get\_player\_list()

info\_list = {player.get\_nickname(): [] for player in player\_list}

for j in range(room\_size):

nicks = []

info = []

for i in range(room\_size):

nicks.append(player\_list[(i + j) % room\_size].get\_nickname())

info.append(None)

info\_list[player\_list[j].get\_nickname()].append(nicks)

info\_list[player\_list[j].get\_nickname()].append(info)

self.local\_storage[room.get\_host\_name()] = info\_list

def \_calculate\_game\_stage(self, host):

room\_storage = self.local\_storage[host]

keys = list(room\_storage.keys())

stages = []

for key in keys:

mini\_storage = room\_storage[key][1]

stage = 0

for i in range(len(mini\_storage)):

if mini\_storage[i] is not None:

stage += 1

else:

stages.append(stage)

return min(stages) if len(stages) > 0 else 0

def \_insert\_into\_storage(self, room, player\_name, prompt):

stage = self.\_calculate\_game\_stage(room.get\_host\_name())

print(stage, " ", player\_name)

room\_storage = self.local\_storage[room.get\_host\_name()]

try:

room\_storage[player\_name][1][stage] = prompt

except IndexError:

pass

finally:

return stage

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

print(f"[NEW CONNECTION] {addr} connected.")

connected = True

while connected:

msg = pickle.loads(conn.recv(1024))

if "\r\n" in msg:

command, message = msg.split("\r\n")

if command != "CHECK\_STATUS":

print(f"[{addr}] {command} {message}")

connected = self.manage\_messages(command, message, conn)

else:

conn.send(pickle.dumps("WRONG\_COMMAND"))

conn.close()

def manage\_messages(self, command, message, conn):

match command:

case "CHECKING\_NICK":

self.find\_nick\_in\_base(message, conn)

return False

case "HELLO":

self.adding\_a\_new\_player(message)

return True

case "DISCONNECT":

self.delete\_a\_player(message, conn)

return False

case "REFRESH":

self.send\_current\_list(conn)

return True

case "CREATE\_ROOM":

self.create\_a\_room(conn)

return True

case "DELETE\_ROOM":

self.delete\_a\_room(message)

return True

case "REFRESH\_ROOM":

self.send\_players\_room(message, conn)

return True

case "JOIN\_ROOM":

self.join\_player\_to\_room(conn)

return True

case "CHECK\_STATUS":

self.send\_room\_status(message, conn)

return True

case "START\_ROOM":

self.start\_a\_room(message, conn)

return True

case "ASK\_START":

self.synchronize\_all\_room(message, conn)

return True

case "STOP\_GAME":

self.close\_a\_game(message, conn)

return True

case "PROMPT":

self.receive\_prompt(message, conn)

return True

case "CANVAS":

self.receive\_canvas(message, conn)

return True

case "ASK\_NEXT":

self.send\_next\_game\_data(message, conn)

return True

case "CHECK\_PIN":

self.check\_room\_pin(message, conn)

return True

case "SEND\_GAME\_RESULT":

self.send\_local\_storage(message, conn)

return True

def send\_local\_storage(self, message, conn):

for room in self.active\_rooms:

if room.get\_host\_name() == message:

conn.send(pickle.dumps(self.local\_storage[message]))

def check\_room\_pin(self, message, conn):

pin = pickle.loads(conn.recv(1024))

for room in self.active\_rooms:

if message == room.get\_host\_name():

if room.check\_pin(pin):

conn.send(pickle.dumps("VALID"))

else:

conn.send(pickle.dumps("INVALID"))

def \_find\_data\_in\_storage(self, host, player, stage):

room\_storage = self.local\_storage[host]

keys = list(room\_storage.keys())

player = keys[(keys.index(player) + stage) % len(keys)]

return room\_storage[player][1][stage]

def send\_next\_game\_data(self, message, conn):

message = message.split(",")

data = self.\_find\_data\_in\_storage(message[0], message[1], int(message[2])-1)

conn.send(pickle.dumps(data))

def receive\_canvas(self, message, conn):

message, name = message.split(",")

for room in self.active\_rooms:

if message == room.get\_host\_name():

conn.send(pickle.dumps("FOUND"))

data = b''

while True:

part = conn.recv(4096)

data += part

if len(part) < 4096:

break

canvas = pickle.loads(data)

self.\_insert\_into\_storage(room, name, canvas)

conn.send(pickle.dumps("ADDED"))

def receive\_prompt(self, message, conn):

message, name = message.split(",")

for room in self.active\_rooms:

if message == room.get\_host\_name():

conn.send(pickle.dumps("FOUND"))

prompt = pickle.loads(conn.recv(1024))

self.\_insert\_into\_storage(room, name, prompt)

conn.send(pickle.dumps("ADDED"))

def close\_a\_game(self, message, conn):

for room in self.active\_rooms:

if message == room.get\_host\_name():

room.stop\_room()

self.local\_storage[room.get\_host\_name()] = None

conn.send(pickle.dumps("CLOSED"))

def synchronize\_all\_room(self, message, conn):

for room in self.active\_rooms:

if message == room.get\_host\_name():

start\_time = room.get\_start\_time()

time.sleep(start\_time + 0.75 - time.time())

conn.send(pickle.dumps("START"))

def start\_a\_room(self, message, conn):

for room in self.active\_rooms:

if message == room.get\_host\_name():

room.start\_room()

room.set\_start\_time(time.time())

self.\_make\_player\_structure(room)

conn.send(pickle.dumps(f"SUCCESS\r\n{room.get\_current()}"))

def send\_room\_status(self, message, conn):

isFound = False

for room in self.active\_rooms:

if message == room.get\_host\_name():

isFound = True

result = "STARTED" if room.is\_room\_started() else "CLOSED"

conn.send(pickle.dumps(result))

if not isFound:

conn.send(pickle.dumps("DELETED"))

def join\_player\_to\_room(self, conn):

host = pickle.loads(conn.recv(1024))

data = b''

while True:

part = conn.recv(4096)

data += part

if len(part) < 4096:

break

player = pickle.loads(data)

for room in self.active\_rooms:

if host == room.get\_host\_name():

room.add\_player(player)

conn.send(pickle.dumps("SUCCESS"))

return

conn.send(pickle.dumps("ERROR"))

def delete\_a\_room(self, message):

if len(self.active\_rooms) == 0:

print("ERROR IT CANNOT BE")

removed = False

for room in self.active\_rooms:

if room.host\_player.get\_nickname() == message:

self.active\_rooms.remove(room)

removed = True

if not removed:

print("ERROR IT CANNOT BE")

def send\_players\_room(self, message, conn):

for room in self.active\_rooms:

if message == room.get\_host\_player().get\_nickname():

conn.send(pickle.dumps(room.get\_player\_list()))

return

conn.send(pickle.dumps("NOT\_FOUND"))

def \_check\_uniqueness(self, message):

return not (message in self.active\_rooms)

def create\_a\_room(self, conn):

conn.send(pickle.dumps("ACCEPTED"))

size = int.from\_bytes(conn.recv(2048), byteorder="big")

data = b''

while len(data) < size:

data += conn.recv(4096)

room = pickle.loads(data)

if self.\_check\_uniqueness(room):

room\_port = random.randint(self.room\_port\_range[0], self.room\_port\_range[1])

while room\_port in self.used\_ports:

room\_port = random.randint(self.room\_port\_range[0], self.room\_port\_range[1])

self.used\_ports.append(room\_port)

room\_address = (self.IP, room\_port)

room.set\_address(room\_address)

self.active\_rooms.append(room)

conn.send(pickle.dumps(f"CREATED\r\n{room\_address}"))

else:

conn.send(pickle.dumps("EXISTS"))

def send\_current\_list(self, conn):

conn.send(pickle.dumps(self.active\_rooms))

def adding\_a\_new\_player(self, message):

self.player\_nicks.append(message)

def delete\_a\_player(self, message, conn):

if message in self.player\_nicks:

self.player\_nicks.remove(message)

conn.send(pickle.dumps("DELETED"))

else:

conn.send(pickle.dumps("NOT\_FOUND"))

def find\_nick\_in\_base(self, nick, conn):

if nick in self.player\_nicks:

response = "UNAVAILABLE"

else:

response = "AVAILABLE"

conn.send(pickle.dumps(response))

def start(self):

self.server.listen()

print(f'curr\_ip={self.IP}')

print("[LISTENING]")

while True:

conn, addr = self.server.accept()

thread = threading.Thread(target=self.handle\_client, args=(conn, addr))

thread.start()

print(f"[ACTIVE CONNECTIONS] {threading.active\_count() - 1}")

hubServer = HubServer()

hubServer.start()

import tkinter as tk

import tkinter.ttk as ttk

from PIL import ImageTk

from settings import \*

from tkinter import messagebox

from support import center

from pygame import mixer

from create\_window import CreateWindow

from room\_window import RoomWindow

import pickle

from pin\_input import PinInput

class HubWindow(tk.Tk):

def \_\_init\_\_(self, player, window\_conn):

super().\_\_init\_\_()

# window

self.title("Palack Paradise")

self.geometry(f'{WIDTH\_HUB}x{HEIGHT\_HUB}')

center(self)

self.iconbitmap('../graphics/icons/main\_ico.ico')

self.resizable(width=False, height=False)

self.configure(background=BG\_COLOR, padx=PADDING, pady=PADDING)

self.half\_height = HEIGHT\_HUB / 2

self.half\_width = WIDTH\_HUB / 2

self.player = player

self.window\_conn = window\_conn

self.protocol("WM\_DELETE\_WINDOW", self.on\_closing)

"""

Main Frame

"""

s = ttk.Style()

s.configure('My.TFrame', background=BG\_FRAME\_COLOR)

self.frame = ttk.Frame(self, borderwidth=BORDER, style="My.TFrame", padding=(0, 0), border=0,

width=HUB\_FRAME\_WIDTH, height=HUB\_FRAME\_HEIGHT)

self.table = ttk.Treeview(master=self.frame, selectmode="browse")

self.table\_init()

"""

Player mini-profile

"""

self.player\_avatar\_img = ImageTk.PhotoImage(self.player.get\_avatar\_image()[1], master=self.frame)

self.player\_avatar = ttk.Label(self.frame, image=self.player\_avatar\_img, borderwidth=BORDER,

padding=(PADDING, PADDING))

self.player\_nick\_lbl = ttk.Label(self.frame, text=self.player.get\_nickname(), background=BG\_FRAME\_COLOR,

font=(FONT\_NAME, NICK\_FONT\_SIZE), justify="center")

"""

Control buttons

"""

self.refresh\_list\_btn = ttk.Button(self.frame, text="Обновить")

self.join\_room\_btn = ttk.Button(self.frame, text="Присоединиться")

self.create\_room\_btn = ttk.Button(self.frame, text="Создать комнату")

"""

Table variables

"""

self.is\_empty = False

self.room\_list = []

self.empty\_room\_notification = ttk.Label(self.table, text="Пока нет ни одной комнаты",

font=(FONT\_NAME, NOTIFICATION\_FONT\_SIZE))

"""

Refresh button personal

"""

self.can\_refresh = True

mixer.init()

self.forbid\_btn\_sound = mixer.Sound("../sounds/btn\_forbid.mp3")

self.forbid\_btn\_sound.set\_volume(0.2)

"""

Some stuff

"""

self.create\_room\_window = None

self.in\_new\_window = False

self.check\_is\_empty()

self.binding()

self.refresh\_table\_items(tk.Event())

def table\_init(self):

self.table['columns'] = ('id', 'host', 'name', 'players', 'privacy')

self.table.column("#0", width=0, stretch=tk.NO)

self.table.column("id", anchor=tk.CENTER, width=int(TABLE\_WIDTH / 10 / 2))

self.table.column("host", anchor=tk.CENTER, width=int(2.3 \* TABLE\_WIDTH / 10))

self.table.column("name", anchor=tk.CENTER, width=int(4.5 \* TABLE\_WIDTH / 10))

self.table.column("players", anchor=tk.CENTER, width=int(1.3 \* TABLE\_WIDTH / 10))

self.table.column("privacy", anchor=tk.CENTER, width=int(1.3 \* TABLE\_WIDTH / 10))

self.table.heading("#0", text="", anchor=tk.CENTER)

self.table.heading("id", text="Id", anchor=tk.CENTER)

self.table.heading("host", text="Host", anchor=tk.CENTER)

self.table.heading("name", text="Name", anchor=tk.CENTER)

self.table.heading("players", text="Players", anchor=tk.CENTER)

self.table.heading("privacy", text="Privacy", anchor=tk.CENTER)

def inverse\_refresh(self):

self.can\_refresh = not self.can\_refresh

def clean\_table(self):

for i in self.table.get\_children():

self.table.delete(i)

def ask\_server\_list(self):

self.window\_conn.send(pickle.dumps(f"REFRESH\r\n{self.player.get\_nickname()}"))

data = b''

while True:

part = self.window\_conn.recv(4096)

data += part

if len(part) < 4096:

break

room\_list = pickle.loads(data)

self.room\_list = room\_list

def add\_items(self):

i = 1

for room in self.room\_list:

self.table.insert(parent='', index='end', text='',

values=(i, room.get\_host\_name(), room.get\_description(),

f"{room.get\_current()}/{room.get\_size()}", "🔑" if room.is\_private() else ""))

i += 1

def refresh\_table\_items(self, ev):

if not self.in\_new\_window:

if self.can\_refresh or ev == "important":

self.clean\_table()

self.ask\_server\_list()

self.add\_items()

self.check\_is\_empty()

self.can\_refresh = False

self.after(4000, self.inverse\_refresh)

else:

self.forbid\_btn\_sound.play()

def send\_server\_verification(self, room):

if room.is\_private():

pinWindow = PinInput()

pinWindow.start()

self.wait\_window(pinWindow)

input\_pin = pinWindow.get\_input\_pin()

if input\_pin:

message = f"CHECK\_PIN\r\n{room.get\_host\_name()}"

self.window\_conn.send(pickle.dumps(message))

self.window\_conn.send(pickle.dumps(input\_pin))

response = pickle.loads(self.window\_conn.recv(1024))

if response == "INVALID":

messagebox.showerror("Ошибка пароля", "Введен неверный пароль")

return

else:

return

message = "JOIN\_ROOM\r\n"

self.window\_conn.send(pickle.dumps(message))

self.window\_conn.send(pickle.dumps(room.get\_host\_name()))

self.window\_conn.send(pickle.dumps(self.player))

response = self.window\_conn.recv(1024)

if response == "ERROR":

return False

return True

def join\_to\_room(self, ev):

if not self.in\_new\_window:

if self.table.selection():

curItem = self.table.focus()

index = int(self.table.item(curItem)['values'][0]) - 1

room\_to\_join = self.room\_list[index]

if room\_to\_join.is\_room\_started():

messagebox.showwarning("Ошибка доступа", "Игра в комнате уже начата")

return

if room\_to\_join.get\_current() >= int(room\_to\_join.get\_size()):

messagebox.showwarning("Ошибка доступа", "В комнату больше не влезет")

return

if not self.send\_server\_verification(room\_to\_join):

return

self.withdraw()

room = RoomWindow(room\_to\_join, self.player, self.window\_conn)

room.start()

self.wait\_window(room)

self.deiconify()

self.refresh\_table\_items("important")

def launching\_room(self, created\_room):

created\_room.set\_host\_player(self.player)

created\_room.add\_player(self.player)

self.window\_conn.send(pickle.dumps(f"CREATE\_ROOM\r\n"))

self.window\_conn.recv(1024)

pickled = pickle.dumps(created\_room)

self.window\_conn.sendall(len(pickled).to\_bytes(4, byteorder='big'))

self.window\_conn.sendall(pickled)

server\_response = pickle.loads(self.window\_conn.recv(1024))

message, addr = server\_response.split("\r\n")

self.refresh\_table\_items("important")

if message == "CREATED":

self.withdraw()

room = RoomWindow(created\_room, self.player, self.window\_conn)

room.start()

self.wait\_window(room)

self.deiconify()

self.check\_is\_empty()

self.refresh\_table\_items("important")

def create\_a\_room(self, ev):

if not self.in\_new\_window:

if len(self.room\_list) > 5:

messagebox.showwarning("Can't create", "There is maximum of rooms already(6/6)")

return

self.create\_room\_window = CreateWindow()

self.in\_new\_window = True

self.create\_room\_window.start()

self.wait\_window(self.create\_room\_window)

self.in\_new\_window = False

created\_room = self.create\_room\_window.get\_created\_room()

if not created\_room:

return

self.launching\_room(created\_room)

def binding(self):

self.refresh\_list\_btn.bind("<Button-1>", self.refresh\_table\_items)

self.join\_room\_btn.bind("<Button-1>", self.join\_to\_room)

self.table.bind('<Button-1>', self.handle\_click)

self.create\_room\_btn.bind("<Button-1>", self.create\_a\_room)

# placing tkinter widgets

def placing\_components(self):

self.frame.place(relx=.5, rely=.5, anchor="center")

self.table.place(x=TABLE\_LEFT, y=TABLE\_TOP, width=TABLE\_WIDTH, height=TABLE\_HEIGHT)

self.player\_avatar.place(relx=0.04, rely=0.075)

self.player\_nick\_lbl.configure(anchor="center")

self.player\_nick\_lbl.place(relx=0.04, rely=0.53, width=PREFERRED\_SIZE[0])

self.join\_room\_btn.place(x=TABLE\_LEFT, rely=0.075, width=BUTTON\_WIDTH, height=BUTTON\_HEIGHT)

self.create\_room\_btn.place(x=TABLE\_LEFT + BUTTON\_WIDTH + OFFSET, rely=0.075,

width=BUTTON\_WIDTH, height=BUTTON\_HEIGHT)

self.refresh\_list\_btn.place(x=TABLE\_LEFT + 2 \* (BUTTON\_WIDTH + OFFSET), rely=0.075, width=REFRESH\_WIDTH,

height=BUTTON\_HEIGHT)

def check\_is\_empty(self):

if len(self.room\_list) == 0:

self.is\_empty = True

self.empty\_room\_notification.place(x=TABLE\_WIDTH / 2 - 140, y=TABLE\_HEIGHT / 2 - 20)

else:

self.is\_empty = False

self.empty\_room\_notification.place\_forget()

# binding

def on\_closing(self):

if messagebox.askokcancel("Quit", "Do you want to quit?"):

self.window\_conn.send(pickle.dumps(f"DISCONNECT\r\n{self.player.get\_nickname()}"))

server\_response = pickle.loads(self.window\_conn.recv(1024))

if server\_response == "DELETED":

self.window\_conn.close()

self.destroy()

def handle\_click(self, event):

if self.table.identify\_region(event.x, event.y) == "separator":

return "break"

# starting of window

def start(self):

self.placing\_components()

import tkinter as tk

from tkinter import ttk

from PIL import Image, ImageDraw, ImageTk

from settings import \*

class Paint(ttk.Frame):

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

# some consts

super().\_\_init\_\_(master=frame)

self.x = 0

self.y = 0

self.brush\_size = 3

# init

self.color = 'black'

self.local\_master = frame

# components

s = ttk.Style(self)

s.configure('Paint.TFrame', background=ROOM\_FRAME)

self['style'] = 'Paint.TFrame'

self.place(relx=0.05, rely=0.18, relwidth=0.95, relheight=0.85)

self.canvas = tk.Canvas(self, bg='white')

self.canvas.place(relheight=0.9, relwidth=0.83, relx=0, rely=0)

self.canvas.bind('<B1-Motion>', self.draw)

self.menu = tk.Menu(self, tearoff=0)

self.brush\_1px = ImageTk.PhotoImage(Image.open('../graphics/utility/brush\_1px.png').

resize((BRUSH\_ICO\_SIZE\*3, BRUSH\_ICO\_SIZE)), master=self)

self.brush\_4px = ImageTk.PhotoImage(Image.open('../graphics/utility/brush\_4px.png').

resize((BRUSH\_ICO\_SIZE \* 3, BRUSH\_ICO\_SIZE)), master=self)

self.brush\_7px = ImageTk.PhotoImage(Image.open('../graphics/utility/brush\_7px.png').

resize((BRUSH\_ICO\_SIZE \* 3, BRUSH\_ICO\_SIZE)), master=self)

self.brush\_13px = ImageTk.PhotoImage(Image.open('../graphics/utility/brush\_13px.png').

resize((BRUSH\_ICO\_SIZE \* 3, BRUSH\_ICO\_SIZE)), master=self)

self.brush\_17px = ImageTk.PhotoImage(Image.open('../graphics/utility/brush\_17px.png').

resize((BRUSH\_ICO\_SIZE \* 3, BRUSH\_ICO\_SIZE)), master=self)

self.menu.add\_command(image=self.brush\_1px, command=lambda size=1: self.brush\_change(size))

self.menu.add\_command(image=self.brush\_4px, command=lambda size=3: self.brush\_change(size))

self.menu.add\_command(image=self.brush\_7px, command=lambda size=6: self.brush\_change(size))

self.menu.add\_command(image=self.brush\_13px, command=lambda size=10: self.brush\_change(size))

self.menu.add\_command(image=self.brush\_17px, command=lambda size=15: self.brush\_change(size))

self.canvas\_img = Image.new('RGB', GAME\_CANVAS\_SIZE, 'white')

self.image\_draw = ImageDraw.Draw(self.canvas\_img)

self.colors = ['black', 'red', 'green', 'yellow', 'white', 'gray', 'brown', 'pink', 'maroon', 'aqua',

'violet', 'CadetBlue', 'burlywood', 'indigo', 'purple']

self.labels = []

base\_x = 0.84

base\_y = 0

offset\_x = 0.05

offset\_y = 0.09

last\_value = (0, 0)

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

label = tk.Button(self, bg=self.colors[i], text='', command=lambda index=i: self.on\_label\_click(index))

label.place(relx=base\_x + (i % 3) \* offset\_x, rely=base\_y + int(i / 3) \* offset\_y, width=35, height=35)

last\_value = (base\_x + (i % 3) \* offset\_x, base\_y + int(i / 3) \* offset\_y)

self.labels.append(label)

self.color\_indicator\_lbl = tk.Label(self, bg=self.color, width=10)

self.color\_indicator\_lbl.place(relx=base\_x, rely=last\_value[1] + offset\_y, relwidth=0.15, relheight=0.08)

base\_y = last\_value[1] + offset\_y\*2

self.pour\_img = ImageTk.PhotoImage(Image.open('../graphics/utility/fill.png').

resize((FILL\_ICO\_SIZE, FILL\_ICO\_SIZE)), master=self)

self.pour\_btn = tk.Button(self, image=self.pour\_img, command=self.pour)

self.pour\_btn.place(relx=base\_x, rely=base\_y)

self.clear\_img = ImageTk.PhotoImage(Image.open('../graphics/utility/clear.png').

resize((FILL\_ICO\_SIZE, FILL\_ICO\_SIZE)), master=self)

self.clear\_canvas\_btn = tk.Button(self, image=self.clear\_img, command=self.clear\_canvas)

self.clear\_canvas\_btn.place(relx=base\_x + offset\_x, rely=base\_y)

self.eraser\_img = ImageTk.PhotoImage(Image.open('../graphics/utility/eraser.png').

resize((FILL\_ICO\_SIZE, FILL\_ICO\_SIZE)), master=self)

self.eraser\_btn = tk.Button(self, image=self.eraser\_img, command=self.erase)

self.eraser\_btn.place(relx=base\_x + 2\*offset\_x, rely=base\_y)

self.brush\_img = ImageTk.PhotoImage(Image.open('../graphics/utility/brush\_size.png').

resize((FILL\_ICO\_SIZE, FILL\_ICO\_SIZE)), master=self)

self.brush\_btn = tk.Button(self, image=self.brush\_img)

self.brush\_btn.bind("<Button-1>", self.popup)

self.brush\_btn.place(relx=base\_x, rely=base\_y + offset\_y)

self.canvas\_bg = None

self.is\_sleep = False

def brush\_change(self, size):

self.brush\_size = size

def on\_label\_click(self, index):

self.color = self.labels[index]['bg']

self.color\_indicator\_lbl['bg'] = self.color

def erase(self):

self.color = 'white'

def draw(self, event):

if not self.is\_sleep:

x1, y1 = (event.x - self.brush\_size), (event.y - self.brush\_size)

x2, y2 = (event.x + self.brush\_size), (event.y + self.brush\_size)

self.canvas.create\_oval(x1, y1, x2, y2, fill=self.color, width=0)

self.image\_draw.ellipse((x1, y1, x2, y2), fill=self.color, width=0)

def select\_brush\_size(self, value):

self.brush\_size = int(value)

def unsleep\_canvas(self):

self.is\_sleep = False

def sleep\_canvas(self):

self.is\_sleep = True

def pour(self):

if not self.is\_sleep:

self.canvas.delete('all')

self.canvas['bg'] = self.color

self.image\_draw.rectangle(CANVAS\_RECT, width=0, fill=self.color)

def clear\_canvas(self):

if not self.is\_sleep:

self.canvas.delete('all')

self.canvas['bg'] = 'white'

self.image\_draw.rectangle(CANVAS\_RECT, width=0, fill='white')

def save\_img(self):

pass

def popup(self, event):

self.menu.post(event.widget.winfo\_rootx(), event.widget.winfo\_rooty())

def place\_paint(self):

self.clear\_canvas()

self.place(relx=0.05, rely=0.18, relwidth=0.95, relheight=0.85)

def place\_forget\_paint(self):

self.place\_forget()

def set\_canvas\_image(self, image):

self.canvas\_bg = ImageTk.PhotoImage(image)

self.clear\_canvas()

self.canvas.create\_image(0, 0, anchor="nw", image=self.canvas\_bg)

def capture\_img(self):

return self.canvas\_img

from tkinter import ttk

import tkinter as tk

from support import is\_pin\_valid, center

from settings import \*

class PinInput(tk.Toplevel):

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

super().\_\_init\_\_()

self.title("Palack Paradise")

self.geometry(f'{PIN\_WIDTH}x{PIN\_HEIGHT}')

self.resizable(width=False, height=False)

self.iconbitmap('../graphics/icons/main\_ico.ico')

self.attributes("-toolwindow", True)

center(self)

self.check = (self.register(is\_pin\_valid), "%P")

self.pin\_input = ttk.Entry(self, validate="key", validatecommand=self.check, font=BASE\_FONT)

self.confirm\_btn = tk.Button(self, text="Подтвердить", command=self.submit\_password, font=BASE\_FONT)

self.password = None

def submit\_password(self):

if len(self.pin\_input.get()) > 0:

self.password = self.pin\_input.get()

self.destroy()

def start(self):

self.pin\_input.pack(pady=5)

self.confirm\_btn.pack(pady=10)

def get\_input\_pin(self):

return self.password

class Player:

def \_\_init\_\_(self, nickname, avatar\_image):

self.nickname = nickname

self.avatar\_image = avatar\_image

def get\_nickname(self):

return self.nickname

def get\_avatar\_image(self):

return self.avatar\_image

def \_\_repr\_\_(self):

return f"[PLAYER]{self.nickname}[PHOTO]{self.avatar\_image[0]}, {self.avatar\_image[1]}\n"

import pickle

import tkinter as tk

from tkinter import ttk

from support import center

from settings import \*

from PIL import ImageTk

from game\_window import GameWindow

class RoomWindow(tk.Toplevel):

def \_\_init\_\_(self, room, player, conn):

super().\_\_init\_\_()

# window

self.title("Palack Paradise")

self.geometry(f'{ROOM\_WIDTH}x{ROOM\_HEIGHT}')

center(self)

self.iconbitmap('../graphics/icons/main\_ico.ico')

self.resizable(width=False, height=False)

self.configure(background=ROOM\_BG, padx=PADDING, pady=PADDING)

self.protocol("WM\_DELETE\_WINDOW", self.on\_closing)

self.half\_height = HEIGHT\_HUB / 2

self.half\_width = WIDTH\_HUB / 2

self.room = room

self.player = player

self.room\_conn = conn

self.is\_started = False

def return\_pressed(ev):

self.start\_room\_click()

self.bind("<Return>", return\_pressed)

"""

GUI

"""

s = ttk.Style(self)

s.configure('Room.TFrame', background=ROOM\_FRAME)

s.configure('My.Treeview', rowheight=65, font=(FONT\_NAME, 18), )

s.map('My.Treeview', background=[('selected', '#B682A5')])

self.frame = ttk.Frame(self, borderwidth=BORDER, padding=(PADDING, PADDING), style="Room.TFrame")

self.players\_view = ttk.Treeview(self.frame, selectmode="browse", show='tree', style="My.Treeview")

self.current\_players\_lbl = ttk.Label(self.frame, font=(FONT\_NAME, 15), background=ROOM\_FRAME,

text=f"Players {self.room.get\_current()}/{self.room.get\_size()}", )

self.start\_game\_btn = tk.Button(self.frame, text="Начать игру", font=(FONT\_NAME, 14),

command=self.start\_room\_click)

"""

Pre-start

"""

self.players\_avatars = {}

self.table\_init()

self.refresh\_player\_list()

self.check\_room\_status()

def check\_room\_status(self):

if not self.is\_started and self.player.get\_nickname() != self.room.get\_host\_name():

message = f"CHECK\_STATUS\r\n{self.room.get\_host\_player().get\_nickname()}"

self.room\_conn.send(pickle.dumps(message))

response = pickle.loads(self.room\_conn.recv(1024))

if response == "STARTED":

self.is\_started = True

self.start\_room()

elif response == "DELETED":

self.destroy()

self.after(300, self.check\_room\_status)

def clean\_table(self):

for i in self.players\_view.get\_children():

self.players\_view.delete(i)

def notify\_server\_about\_start(self):

message = f"START\_ROOM\r\n{self.room.get\_host\_player().get\_nickname()}"

self.room\_conn.send(pickle.dumps(message))

response = pickle.loads(self.room\_conn.recv(1024))

if response == "SUCCESS":

print("Room started")

def start\_room(self):

self.withdraw()

game = GameWindow(self.room, self.player, self.room\_conn)

game.start()

self.wait\_window(game)

self.is\_started = False

self.deiconify()

def start\_room\_click(self):

if self.player.get\_nickname() == self.room.get\_host\_player().get\_nickname():

self.notify\_server\_about\_start()

self.is\_started = True

self.start\_room()

def table\_init(self):

self.players\_view['columns'] = ('nickname', 'host')

table\_width = ROOM\_WIDTH \* 0.8 \* 0.35

self.players\_view.column("#0", anchor=tk.CENTER, width=int(2 \* table\_width / 10))

self.players\_view.column("nickname", anchor=tk.CENTER, width=int(3.7 \* table\_width / 10))

self.players\_view.column("host", anchor=tk.CENTER, width=int(1 \* table\_width / 10))

def add\_items(self, player\_list):

for i, player in enumerate(player\_list):

self.players\_avatars[player.get\_nickname()] = ImageTk.PhotoImage(player.get\_avatar\_image()[1].

resize(MINI\_AVATAR\_SIZE),

master=self.frame)

self.players\_view.insert('', "end", text='', image=self.players\_avatars[player.get\_nickname()],

values=(player\_list[i].get\_nickname(),

("👑" if self.room.get\_host\_player().get\_nickname() ==

player\_list[i].get\_nickname() else "")))

self.current\_players\_lbl["text"] = f"Players {self.room.get\_current()}/{self.room.get\_size()}"

def refresh(self):

self.clean\_table()

self.add\_items(self.room.get\_player\_list())

def refresh\_player\_list(self):

if not self.is\_started:

self.room\_conn.send(pickle.dumps(f"REFRESH\_ROOM\r\n{self.room.get\_host\_player().get\_nickname()}"))

data = b''

while True:

part = self.room\_conn.recv(4096)

data += part

if len(part) < 4096:

break

players = pickle.loads(data)

if players != "NOT\_FOUND":

self.room.set\_player\_list(players)

self.refresh()

self.after(2000, self.refresh\_player\_list)

def get\_room(self):

return self.room

def placing(self):

self.frame.place(relx=.5, rely=.5, relwidth=0.8, relheight=0.8, anchor="center")

self.players\_view.place(relx=0.05, rely=0.1, relwidth=0.30, relheight=0.80)

self.current\_players\_lbl.place(relx=0.2, rely=0.07, anchor="center")

if self.player.get\_nickname() == self.room.get\_host\_name():

self.start\_game\_btn.place(relx=0.6, rely=0.15, relheight=0.1, relwidth=0.15, anchor="center")

def start(self):

self.placing()

def on\_closing(self):

if self.player.get\_nickname() == self.room.get\_host\_name():

self.room\_conn.send(pickle.dumps(f"DELETE\_ROOM\r\n{self.player.get\_nickname()}"))

self.destroy()

"""

MAIN CONFIG

"""

BORDER = 0

PADDING = 0

AVATAR\_MAX\_SIZE = 4194304

"""

Start room config

"""

START\_WIDTH = 700

START\_HEIGHT = 700

PREFERRED\_SIZE = (175, 175)

FPS = 120

START\_WINDOW\_WIDTH = 250

START\_WINDOW\_HEIGHT = 350

BORDER\_SIZE = 4

UI\_BACKGROUND\_COLOR = '#c3dbd7'

UI\_BORDER\_COLOR = '#415a77'

NEXT\_ICO\_SIZE = 17

"""

Hub WINDOW ui config

"""

WIDTH\_HUB = 950

HEIGHT\_HUB = 550

TABLE\_WIDTH = 460

TABLE\_HEIGHT = 280

TABLE\_LEFT = 250

TABLE\_TOP = 80

BG\_COLOR = '#731ab9'

BG\_FRAME\_COLOR = '#729ac9'

HUB\_FRAME\_WIDTH = 750

HUB\_FRAME\_HEIGHT = 400

FONT\_NAME = "Skellyman"

NICK\_FONT\_SIZE = 14

NOTIFICATION\_FONT\_SIZE = 16

BUTTON\_WIDTH = 155

OFFSET = 30

BUTTON\_HEIGHT = 30

BUTTON\_PANEL\_TOP = 30

TOP\_BUTTON\_FONT\_SIZE = 16

REFRESH\_WIDTH = 90

MINI\_PROFILE\_LEFT = 30

AVATAR\_TOP = 30

NICK\_TOP = 210

"""Create window config"""

CREATE\_WIDTH = 800

CREATE\_HEIGHT = 500

CREATE\_BG\_COLOR = "#ba5fdf"

NAME\_INPUT\_FONT\_SIZE = 13

SIZE\_INPUT\_FONT\_SIZE = 13

LBL\_FONT\_SIZE = 14

ALL\_FIELDS\_HEIGHT = 30

FRAME\_COLOR = "#f2bcef"

FRAME\_WIDTH = 650

FRAME\_HEIGHT = 400

EYE\_ICO\_SIZE = 22

"""Room Window"""

ROOM\_WIDTH = 1100

FRAME\_W = 880

FRAME\_H = 560

ROOM\_HEIGHT = 700

ROOM\_BG = "#9E4EDC"

ROOM\_FRAME = "#C999EE"

MINI\_AVATAR\_SIZE = (60, 60)

"""Game"""

GAME\_CANVAS\_SIZE = (800, 480)

CANVAS\_WIDTH = 800

CANVAS\_HEIGHT = 480

CANVAS\_RECT = (0, 0, 800, 480)

FILL\_ICO\_SIZE = 35

BRUSH\_ICO\_SIZE = 35

START\_PROMPT\_TIME = 25

START\_CANVAS\_TIME = 40

"""Pin input"""

PIN\_WIDTH = 300

PIN\_HEIGHT = 150

BASE\_FONT = (FONT\_NAME, 14)

"""Result canvas"""

BASE\_OFFSET\_Y = 50

CANVAS\_W = 792

CANVAS\_H = 448

AVATAR\_SIZE = (MINI\_AVATAR\_SIZE[0] + 15, MINI\_AVATAR\_SIZE[0] + 15)

LEFT\_AVATAR\_X = CANVAS\_W - MINI\_AVATAR\_SIZE[0] - 15

RIGHT\_AVATAR\_X = 50

CANVAS\_IMG\_SIZE = (420, 245)

import os.path

import re

from PIL import Image, ImageTk

import tkinter as tk

import tkinter.ttk as ttk

from tkinter import filedialog

from tkinter import messagebox

from settings import \*

from support import center, get\_images\_from\_dir, is\_valid

from random import shuffle

from player import Player

import socket

import pickle

class StartWindow(tk.Tk):

def \_\_init\_\_(self, addr, player=None):

super().\_\_init\_\_()

# window

self.title("Palack Paradise")

self.geometry(f'{START\_WIDTH}x{START\_HEIGHT}')

center(self)

self.iconbitmap('../graphics/icons/main\_ico.ico')

self.configure(background="#F47983", padx=0, pady=0)

self.resizable(width=False, height=False)

self.half\_height = START\_HEIGHT / 2

self.half\_width = START\_WIDTH / 2

self.player = player

self.check\_server\_addr = addr

self.protocol("WM\_DELETE\_WINDOW", self.on\_closing)

def return\_pressed(ev):

self.start\_game()

self.bind("<Return>", return\_pressed)

"""

Pre-load

"""

self.avatars = list(get\_images\_from\_dir('../graphics/avatars').items())

self.current\_avatar\_index = 0

self.next\_img = ImageTk.PhotoImage(Image.open('../graphics/utility/next.png').

resize((NEXT\_ICO\_SIZE, NEXT\_ICO\_SIZE)), master=self)

shuffle(self.avatars)

"""

Avatar stuff

"""

# avatar picture

self.current\_PIL\_avatar = self.avatars[0]

self.current\_avatar\_image = ImageTk.PhotoImage(self.current\_PIL\_avatar[1], master=self)

self.current\_avatar = ttk.Label(self, image=self.current\_avatar\_image,

borderwidth=BORDER, padding=(PADDING, PADDING))

self.next\_avatar = ttk.Button(self, image=self.next\_img, command=self.change\_avatar)

"""

Other

"""

# buttons

# self.file\_button = tk.Button(self, text="Выбрать файл", command=self.choose\_image, font=(FONT\_NAME, 10),

# state="disabled")

self.file\_button = tk.Button(self, text="Выбрать файл", command=self.choose\_image, font=(FONT\_NAME, 10))

# start game btn

self.create\_person\_btn = tk.Button(self, font=(FONT\_NAME, 12), text="Начать игру", command=self.start\_game)

"""

Nickname stuff

"""

# nickname entry

self.check = (self.register(is\_valid), "%P")

self.nickname\_field = ttk.Entry(self, font=(FONT\_NAME, 15), validate="key", validatecommand=self.check)

self.nickname\_lbl = ttk.Label(self, text="Введите ник", font=("Skellyman", 17), background="#F47983")

"""

Binding

"""

# image choosing

def choose\_image(self):

image\_file = filedialog.Open(filetypes=[('Image formats', '.jpg .png')]).show()

if image\_file:

if os.path.getsize(image\_file) < AVATAR\_MAX\_SIZE:

image = Image.open(image\_file).resize(PREFERRED\_SIZE)

real\_name = os.path.basename(image\_file)

dest\_dir = "../graphics/avatars/"

dest\_path = dest\_dir + real\_name

if os.path.exists(dest\_path):

ext = os.path.splitext(real\_name)[1]

name = os.path.splitext(real\_name)[0]

i = 1

while os.path.exists(dest\_dir + real\_name):

real\_name = f"{name} ({i}){ext}"

i += 1

dest\_path = dest\_dir + real\_name

self.current\_PIL\_avatar = (dest\_path, image)

self.current\_PIL\_avatar[1].save(dest\_path)

self.current\_avatar\_image = ImageTk.PhotoImage(self.current\_PIL\_avatar[1], master=self)

self.current\_avatar.configure(image=self.current\_avatar\_image)

else:

messagebox.showwarning("Некорректный файл", "Выбранный файл имеет слишком\n большой размер(> 4MB)")

# taking next photo

def change\_avatar(self):

if self.current\_PIL\_avatar not in self.avatars:

os.remove(self.current\_PIL\_avatar[0])

i = self.current\_avatar\_index + 1 if self.current\_avatar\_index + 1 < len(self.avatars) else 0

self.current\_avatar\_index = i

self.current\_PIL\_avatar = self.avatars[i]

self.current\_avatar\_image = ImageTk.PhotoImage(self.current\_PIL\_avatar[1], master=self)

self.current\_avatar.configure(image=self.current\_avatar\_image)

"""

Placing

"""

# placing

def placing\_components(self):

# avatar stuff

self.current\_avatar.place(x=self.half\_width - 84, y=self.half\_height - 250)

self.file\_button.place(x=self.half\_width - 60, y=self.half\_height - 68, width=125, height=25)

self.next\_avatar.place(x=self.half\_width + 66, y=self.half\_height - 100)

# nickname stuff

self.nickname\_lbl.place(x=self.half\_width - 69, y=self.half\_height - 22)

self.nickname\_field.place(x=self.half\_width - 90, y=self.half\_height + 14, width=195, height=30)

# create person btn

self.create\_person\_btn.place(x=self.half\_width - 75, y=self.half\_height + 60, width=165, height=30)

"""

Other stuff

"""

def check\_field\_fill(self):

if len(self.nickname\_field.get()) > 3 and not re.match("[ \-\_()\*]+$", self.nickname\_field.get()):

if self.current\_avatar:

return True

return False

def get\_created\_player(self):

return self.player

def check\_nickname\_availability(self, nickname):

check\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

try:

check\_socket.connect(self.check\_server\_addr)

check\_socket.send(pickle.dumps(f"CHECKING\_NICK\r\n{nickname}"))

response = pickle.loads(check\_socket.recv(1024))

except ConnectionRefusedError:

messagebox.showerror("Error", "Server not started")

return "error"

except TimeoutError:

messagebox.showerror("Error", "Wrong IP")

finally:

check\_socket.close()

if response:

if response == "AVAILABLE":

return "available"

else:

return "unavailable"

"""

Start

"""

# after clicking start btn

def start\_game(self):

if self.check\_field\_fill():

result = self.check\_nickname\_availability(self.nickname\_field.get())

if result == "available":

# self.player = Player(self.nickname\_field.get(), self.current\_PIL\_avatar[0])

self.player = Player(self.nickname\_field.get(), self.current\_PIL\_avatar)

self.destroy()

elif result == "unavailable":

messagebox.showwarning("Nick warning", "Nickname is unavailable!")

else:

messagebox.showerror("Input error", "Nick should be 4-15 symbols")

# start of this window

def start(self):

self.placing\_components()

def on\_closing(self):

if self.current\_PIL\_avatar not in self.avatars:

os.remove(self.current\_PIL\_avatar[0])

self.destroy()

from os import walk

from PIL import Image

from settings import \*

import re

import socket

def center(win):

win.update\_idletasks()

width = win.winfo\_width()

frm\_width = win.winfo\_rootx() - win.winfo\_x()

win\_width = width + 2 \* frm\_width

height = win.winfo\_height()

titlebar\_height = win.winfo\_rooty() - win.winfo\_y()

win\_height = height + titlebar\_height + frm\_width

x = win.winfo\_screenwidth() // 2 - win\_width // 2

y = win.winfo\_screenheight() // 2 - win\_height // 2

win.geometry('{}x{}+{}+{}'.format(width, height, x, y))

win.deiconify()

def get\_images\_from\_dir(dir\_name):

surface\_list = {}

for \_, \_\_, img\_files in walk(dir\_name):

for img in img\_files:

full\_path = dir\_name + '/' + img

image\_surf = Image.open(full\_path)

image\_surf = image\_surf.resize(PREFERRED\_SIZE)

image\_surf.save(full\_path)

surface\_list.update({full\_path: image\_surf})

return surface\_list

def is\_pin\_valid(new\_val):

if not new\_val:

return True

result = re.match("[0-9a-zA-Z\_\b]+$", new\_val) is not None

if not result or len(new\_val) > 8:

return False

return True

def get\_ip():

s = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

try:

s.connect(('10.255.255.255', 1))

IP = s.getsockname()[0]

except TimeoutError:

IP = '127.0.0.1'

finally:

s.close()

return IP

def is\_description\_valid(new\_val):

if not new\_val:

return True

result = re.match("[0-9a-zA-Zа-яА-Я\_\-() ]+$", new\_val) is not None

if not result or len(new\_val) > 25:

return False

return True

def is\_prompt\_valid(new\_val):

if not new\_val:

return True

result = re.match("[0-9a-zA-Zа-яА-Я()\-=<>,/?.\"{}+!@$;:&\*\_`% ]+$", new\_val) is not None

if not result or len(new\_val) > 50:

return False

return True

def is\_valid(new\_val):

if not new\_val:

return True

result = re.match("[0-9a-zA-Z\_\-\*() ]+$", new\_val) is not None

if not result or len(new\_val) > 15:

return False

return True

from tkinter import ttk

class Timer:

def \_\_init\_\_(self, master, time\_ms, mode):

self.master = master

self.time = 0

self.time\_ms = time\_ms

self.mode = mode

if mode == "prompt":

master.prompt\_input.state(['!disabled'])

else:

master.paint.canvas.configure(state="normal")

self.is\_timer\_end = False

self.progressbar = ttk.Progressbar(self.master, orient="horizontal", length=200, mode="determinate")

self.progressbar.pack(pady=10)

def start\_timer(self):

self.progressbar["maximum"] = self.time\_ms \* 10

self.time += 1

self.progressbar["value"] = self.time \* 2

if self.time < self.time\_ms \* 2 \* 2.5:

self.master.after(200, self.start\_timer)

elif self.time == self.time\_ms \* 2 \* 2.5:

self.is\_timer\_end = True

self.callback(self.master)

def callback(self, master):

result = None

self.progressbar.pack\_forget()

if self.mode == "prompt":

master.prompt\_input.state(['disabled'])

result = master.prompt\_input.get()

elif self.mode == "canvas":

master.paint.canvas.configure(state="disabled")

result = master.paint.capture\_img()

master.after\_timer(result, self.mode)