

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«КРЫМСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ им. В. И. ВЕРНАДСКОГО»
ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
Кафедра компьютерной инженерии и моделирования

РАЗРАБОТКА ПРИЛОЖЕНИЯ «МОРСКОЙ БОЙ НА 4 ИГРОКА»

Курсовая работа
по дисциплине «Программирование»
студента 1 курса группы ПИ-б-о-201
Баламутина Егора Никитовича
направления подготовки 09.03.04 «Программная инженерия»

Научный руководитель
старший преподаватель кафедры

компьютерной инженерии и моделирования

(оценка)

(подпись, дата)

Чабанов В.В.

Симферополь, 2021

РЕФЕРАТ

Разработка приложения «Морской бой на 4 игрока». – Симферополь: ФТИ КФУ им. В. И. Вернадского, 2021. – 41 с., 15 ил., 5 ист.

Объект разработки – игра «морской бой» для четырёх игроков, сервер для игры.

Цель работы – создание рабочего приложения, использующего клиент-серверную структуру. Организация связи клиентской и серверной частей через сеть с помощью Post и Get запросов.

После рассмотрения возможных вариантов реализации проекта, было принято решение использовать язык C++, ввиду его быстродействия, для серверной части и Python, ввиду его простоты и широких возможностей, для клиентской. Для реализации клиент-серверной структуры было решено использовать как API, так и webhook-и.

ПРОГРАММИРОВАНИЕ, C++, Python, WEBHOOK, СОЗДАНИЕ ИГРОВОГО ПРИЛОЖЕНИЯ

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....	4
ГЛАВА 1 ПОСТАНОВКА ЗАДАЧИ.....	5
1.1 Цель проекта.....	5
1.2 Существующие аналоги	5
1.3 Основные отличия от аналогов	5
1.4 Техническое задание.....	6
ГЛАВА 2 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ.....	10
2.1 Анализ инструментальных средств.....	10
2.2 Описание алгоритмов	10
2.3 Описание структур данных.....	15
2.4 Описание основных модулей.....	18
ГЛАВА 3 ТЕСТИРОВАНИЕ ПРОГРАММЫ.....	23
3.1 Тестирование исходного кода.....	23
3.2 Тестирование интерфейса пользователя и юзабилити.....	25
ГЛАВА 4 ПЕРСПЕКТИВЫ ДАЛЬНЕЙШЕГО РАЗВИТИЯ ПРОЕКТА.....	26
4.1 Перспективы технического развития.....	26
4.2 Перспективы монетизации.....	26
ЗАКЛЮЧЕНИЕ	27
ЛИТЕРАТУРА	28
ПРИЛОЖЕНИЕ 1 КЛАСС GAME СЕРВЕРНОЙ ЧАСТИ.....	29
ПРИЛОЖЕНИЕ 2 ТЕСТ-КЕЙСЫ ПРОЕКТА.....	35

ВВЕДЕНИЕ

Предметом данной работы является создание приложения, реализующего графический интерфейс, которое позволяет пользователям сыграть в старый добрый «морской бой» в командном режиме. Вместо того чтобы играть друг с другом один на один, игроки будут объединяться в команду из двух человек и сражаться против другой команды.

Целью проекта является создание приятного пользовательского интерфейса и реализация удобных для игроков способов создания игр и распределения по командам. Для реализации графического интерфейса клиентского приложения (язык Python) был использован модуль tkinter, а также библиотека Pillow для работы с изображениями. Для использования API на клиенте понадобился модуль requests, а для вебхуков использовался микрофреймворк flask. На сервере (язык C++) за работу с сетью отвечала библиотека cpp_httplib. Также была использована библиотека nlohmann json.

ГЛАВА 1

ПОСТАНОВКА ЗАДАЧИ

1.1 Цель проекта

Создать рабочее приложение «Морской бой 2x2» (Sea Battle 2x2). Получить навыки разработки больших программ. Закрепить уже имеющиеся и получить новые навыки создания клиент-серверной архитектуры и использования вебхуков. Получить опыт создания больших программ с графическим интерфейсом.

1.2 Существующие аналоги

«Морской бой» – довольно известная игра. Поэтому существует бесчисленное множество её компьютерных реализаций. Например:

- «Морской бой 2» – Популярное приложение в Play Market. Ссылка на приложение:
<https://play.google.com/store/apps/details?id=com.byril.seabattle2&hl=ru&gl=US>
- «Морской бой (онлайн игра для двоих)» – браузерная игра. Ссылка:
<http://ru.battleship-game.org/>
- Морской бой на ПК. Ссылка: <https://m.moreigr.com/simulyatory-games/4199-morskoy-boy.html>

1.3 Основные отличия от аналогов

Подобных приложений существует очень много. Однако при поиске аналогов не было найдено ни одной игры, предусматривающей режим игры для четырёх игроков. Это значительное отличие данного проекта от аналогов. Также механика боя в проекте будет немного отличаться от классической: количество выстрелов у игрока за ход будет зависеть от количества оставшихся у него кораблей.

1.4 Техническое задание

Перед программной реализацией проекта было составлено техническое задание. В нём были описаны требования к функционалу приложения, описан интерфейс клиентской части и указано, что должно происходить в приложении в ответ на то или иное действие пользователя. Содержание технического задания:

1. Интерфейс программы должен содержать следующие страницы:

- 1.1. Главная страница (1), на которую попадает пользователь при запуске приложения. Она должна содержать кнопку «Играть», поле для ввода никнейма игрока, а также поля, в которые пользователь должен будет ввести URL сервера и свой webhook.
- 1.2. Страница со списком доступных игр (2), также содержащая кнопку «Создать игру» и поле для ввода ID игры. На каждой игре в списке отображается её ID и никнейм создателя, а также количество игроков.
- 1.3. Страница подготовки к бою (3), содержащая поле для расстановки кораблей и рядом с ним сами корабли. Также на странице содержатся четыре ячейки – две синих и две красных. Они обозначают одну и вторую команду. Также на этой странице отображаются никнеймы подключившихся игроков и ID игры.
- 1.4. Страница боя (4). На ней отображаются игровые поля: своё и союзника, а также два поля противников. Поля подписаны соответствующими никнеймами. На против своего поля в четыре линии отображаются все корабли, в конце линий – символы выстрела (подробнее в пункте 5. Процесс боя). Также на странице должен быть индикатор, показывающий чей сейчас ход.

2. Процедура подбора игроков (страница 2).

- 2.1. Пользователь может нажать на кнопку «Создать игру». В этом случае он попадёт на страницу 3 и будет там единственным игроком. Созданная им игра появится в списке доступных игр, другие игроки смогут к ней присоединиться.

2.2. Пользователь может нажать на любую игру из списка. В этом случае он присоединится к этой игре и попадёт на страницу 3.

2.3. Любой пользователь, находящийся на странице 3, видит ID игры. Он может сообщить его другому пользователю, а тот должен будет ввести его в поле на странице 2. Когда пользователь вводит ID игры, он присоединяется к этой игре и попадает на страницу 3, либо получает сообщение, что игра не найдена.

3. Процедура выбора команды (страница 3).

3.1. На странице 3 находятся две синие и две красные ячейки, обозначающие соответственно первую и вторую команду. Игрок может нажать на одну из этих ячеек и, если эта ячейка пуста, то его имя начнёт отображаться в этой ячейке, и он станет считаться членом этой команды.

3.2. Находясь в команде, пользователь может нажать на ячейку со своим именем – тогда он выйдет из команды. Также он может нажать на любую другую свободную ячейку, тогда он автоматически удалится из своей ячейки и появится в другой.

3.3. Если пользователь нажимает на ячейку, в которой уже записано имя другого игрока, то ничего не происходит.

4. Процедура расстановки кораблей (страница 3).

4.1. На странице 3 находится поле для расстановки кораблей, рядом – сами корабли. Количество кораблей – как в классических правилах: один корабль на четыре клетки, два корабля на три клетки, три на две клетки и четыре корабля по одной клетке.

4.2. Также рядом должна быть кнопка, нажав на которую игрок сообщает другим, что он готов. Нажать на неё можно, только если игрок расставил все корабли и вступил в команду. Когда он нажимает на кнопку, она становится подсвеченной и его никнейм в списке игроков также подсвечивается. С нажатой кнопкой игрок не может поменять команду и расстановку кораблей. Чтобы что-то изменить, он должен повторно нажать

на кнопку, чтобы она перестала быть подсвеченной (имя игрока в списке также перестаёт быть подсвеченным).

4.3. Игрок может нажимать на корабли (как на ещё не расставленные, так и на уже расставленные). При этом корабль, на который он нажал, становится выбранным (подсвеченным), а корабль, который был выбранным до этого (если такой был) перестаёт быть выбранным.

4.4. Если игрок нажал на любую пустую клетку поля, и при этом был выбран какой-нибудь корабль, то если на новой позиции кораблю ничего не мешает, он пропадает со старого места и появляется на ней.

4.5. Игрок может нажать правой кнопкой мыши на корабль на поле. Тогда, если кораблю ничего не мешает, он поворачивается на 90 градусов.

4.6. Игрок может убрать корабль с поля. Для этого он должен выбрать этот корабль и нажать на место, в котором он стоял до того, как был поставлен на поле.

5. Процесс боя (страница 4).

5.1. Игроки ходят по очереди. Чтобы не было соперничества за место в определённой команде, игрок, который будет делать первый ход, выбирается случайным образом. Далее право ходить передаётся по кругу, при чём после хода одного игрока, следующий ход достаётся игроку из противоположной команды. То есть два игрока из одной команды не могут походить подряд.

5.2. За ход у игрока может быть до четырёх выстрелов – это зависит от оставшихся у него кораблей: пока у него цел хотя бы один корабль из класса, игрок получает выстрел за этот класс. Класс корабля определяется его длиной, то есть всего существует четыре класса: первый (4 корабля), второй (3 корабля), третий (2 корабля) и четвёртый (1 корабль).

5.3. Как говорилось в пункте 1.4, напротив поля в четыре линии отображаются все корабли (корабли каждого класса в своей линии). В конце каждой линии – символ выстрела. В начале хода все четыре символа активны, с каждым выстрелом один символ потухает. Когда все символы потухли, ход

игрока заканчивается. В начале его следующего хода символы снова активируются.

5.4. Когда игроку уничтожают корабль, он вычёркивается и с линии возле поля. Если на линии зачёркнуты все корабли, то вычёркивается и символ выстрела – игрок больше не сможет его использовать.

5.5. Если у игрока не осталось кораблей (следовательно – не осталось выстрелов), то он больше не сможет делать ходы. Игра заканчивается, когда у одной из команд не останется кораблей.

6. Клиент - сервер.

6.1. При первом запуске клиента, он обращается к серверу, сервер выдаёт ему уникальный ID, который клиент сохраняет в файле. При всех последующих запросах клиент передаёт в теле запроса своё ID.

6.2. При повторном запуске клиента, он достаёт ID из файла. Если файл открыть не удалось, то он обращается к серверу за новым ID.

6.3. Также при запуске клиент начинает слушать определённый порт. Он отправляет на сервер вебхук, на который сервер будет отправлять ему новые данные.

ГЛАВА 2

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

2.1 Анализ инструментальных средств

Для реализации серверного приложения был выбран язык C++ и среда разработки Visual Studio Community. Такой выбор обусловлен прежде всего тем, что C++ – один из наиболее быстродействующих языков программирования (а на сервер может возлагаться большая нагрузка, при подключении большого числа клиентов), а также уже имеющимся опытом использования библиотек для этого языка: `cpr_httplib` для работы с сетью и `nlohmann json` для обработки данных в формате json.

Для создания клиентского приложения с графическим интерфейсом был выбран язык Python, так как он довольно удобный для решения этой задачи, и позволит создать требуемое приложение за более короткий срок. Также для простого клиентского приложения нет необходимости в быстрых вычислениях. Для реализации пользовательского интерфейса был использован модуль `tkinter`, а также библиотека `Pillow` для работы с изображениями. Для использования API использовался модуль `requests`, а для приёма вебхуков – микрофреймворк `flask`. Для одновременной работы графического интерфейса и `flask`-а потребовалось запустить последний в отдельном потоке. Для реализации многопоточности был использован встроенный модуль `threading`.

2.2 Описание алгоритмов

Одна из первых задач при реализации проекта – как различать клиентов, которые отправляют запросы на сервер? Для решения этой задачи было решено, что каждый клиент при первом запуске будет отправлять запрос на сервер для получения своего уникального ID. Полученный от сервера ID клиент сохраняет в файл и во всех последующих запросах к серверу (кроме Get-запросов) отправляет его в теле запроса).

Как же генерировать уникальный ID для каждого клиента? Было решено, что ID будет строкой, представляющей собой восьмизначное шестнадцатеричное число. Первым ID будет «00000000», каждое следующее ID будет увеличиваться на 1. Таким образом, всего вариантов ID – 16^8 или 2^{32} . Функция, генерирующая ID, представлена на рисунке 2.1.

```

406 std::string id_up(std::string id) {
407     bool finish = false;
408     for (int i = 7; i >= 0; i--) {
409         if (finish) break;
410         switch (id[i])
411         {
412             case('0'):
413             case('1'):
414             case('2'):
415             case('3'):
416             case('4'):
417             case('5'):
418             case('6'):
419             case('7'):
420             case('8'):
421             case('A'):
422             case('B'):
423             case('C'):
424             case('D'):
425             case('E'):
426                 id[i] += 1;
427                 finish = true;
428                 break;
429             case('9'):
430                 id[i] = 'A';
431                 finish = true;
432                 break;
433             case('F'):
434                 id[i] = '0';
435             }
436     }
437     return id;
438 }
439 
```

Рисунок 2.1. Функция id_up.

Следующей задачей стало хранение информации об активных на данный момент играх. Для этого был реализован класс `Game`, описанный в пункте 2.3. Был создан глобальный словарь `active_games` (рисунок 2.2). Ключом в словаре была строка – ID игры (генерировалось аналогично ID клиентов), а значением – адрес объекта класса `Game`

```
std::map <std::string, Game*> active_games;
```

Рисунок 2.2. Словарь активных игр.

Для связи клиентов использовались API и вебхуки. При определённых действиях пользователя клиентское приложение отправляло запрос на сервер. На сервере в зависимости от того, на какой адрес пришёл запрос, вызывалась соответствующая функция (рисунок 2.3). Функция обрабатывала запрос, отправляла ответ и, если требовалось, производилась отправка новых данных на вебхуки.

```

389  int main() {
390      srand(time(0));
391      Server svr;
392      svr.Get("/register", new_id_reg);
393      svr.Get("/activegames", show_act_games);
394      svr.Post("/create", create_game);
395      svr.Post("/join", join_game);
396      svr.Post("/exit", exit_game);
397      svr.Post("/enterteam", enter_team);
398      svr.Post("/quitteam", quit_team);
399      svr.Post("/ready", set_ready);
400      svr.Post("/notready", deselect_ready);
401      svr.Post("/shoot", do_shoot);
402      std::cout << "Starting server on \"http://localhost:1234/\" \n";
403      svr.listen("localhost", 1234);
404  }
405
406  void new_id_reg(const Request& req, Response& res) { ... }
440
441  void show_act_games(const Request& req, Response& res) { ... }
456
457  void create_game(const Request& req, Response& res) { ... }
472
473  void join_game(const Request& req, Response& res) { ... }
513
514  void exit_game(const Request& req, Response& res) { ... }
547
548  void enter_team(const Request& req, Response& res) { ... }
593
594  void quit_team(const Request& req, Response& res) { ... }
630
631  void set_ready(const Request& req, Response& res) { ... }
711
712  void deselect_ready(const Request& req, Response& res) { ... }
752
753  void do_shoot(const Request& req, Response& res) { ... }

```

Рисунок 2.3. Функции, отвечающие за сетевое взаимодействие.

Когда клиент запускается в первый раз, он отправляет Get-запрос на “/register”. Запускается функция `new_id_reg`. В ней генерируется и отправляется в ответ новый ID.

Когда на клиенте пользователь нажимает кнопку «Play» и открывается страница со списком активных игр, то выполняется Get-запрос на “/activegames”.

В результате выполняется функция `show_act_games`, которая отправляет в ответ данные в формате `json`, содержащие информацию об активных играх.

Если Пользователь нажимает на кнопку «Создать игру», то выполняется Post-запрос на `“/create”`, вызывается функция `create_game`. В ней создаётся объект класса `Game` и добавляется в словарь `active_games`. В ответ функция отправляет ID созданной игры.

Если пользователь, находясь на странице со списком игр, нажимает на не пустой пункт списка или вводит ID игры в поле и нажимает «Присоединится», то отправляется Post-запрос (содержащий ID игры) на `“/join”` и вызывается функция `join_game`. Эта функция проверяет, есть ли игра с таким ID, и если есть, то добавляет игрока в эту игру, отправляет ему в ответ информацию о других игроках в этой игре, а также отсылает на вебхуки игрокам информацию о новом игроке.

Находясь в игре (на странице подготовки к бою), пользователь может нажать кнопку «назад» и покинуть игру. Тогда отправится запрос на `“exit”` и вызовется функция `exit_game`, которая отправит на вебхуки остальным игрокам, оповещение о том, что игрок вышел.

Также, при подготовке к бою игрок может нажать на пустую ячейку, чтобы вступить в команду, или на ячейку со своим именем (если он уже состоит в команде), чтобы выйти из команды. Это приведёт к отправке соответствующего запроса на `“/enterteam”` либо `“/quitteam”`. Вызовется функция `enter_team` либо `quit_team` соответственно, которые посредством вебхуков оповестят игроков об изменении команд.

Если пользователь нажмёт на кнопку «Готов», то отправится Post-запрос на `“/ready”`, который помимо всего прочего будет содержать информацию о расставленных игроком кораблях. На сервере выполнится проверка и, если расстановка кораблей валидная, то отправляется утвердительный ответ и оповещение на все вебхуки.

Если игрок захочет что-то изменить в расстановке кораблей или команде, то он должен будет отменить готовность. Он должен повторно нажать на кнопку

«Готов». Тогда выполнится запрос на “/notready”, вызовется функция `deselect_ready` и сервер оповестит все вебхуки об отмене готовности игрока.

Если при утверждении готовности очередного игрока сервер выяснит, что все четверо игроков готовы, то он разошлёт сообщение о начале боя на все вебхуки. В этом сообщении будет в том числе и информация о том, кому достаётся право ходить первым.

Во время боя игрок может нажать на поле противника, и если в этот момент право хода принадлежит ему, то будет отправлен запрос на сервер (на “/shoot”) с информацией о том, в какую клетку выстрелил игрок. Вызовется функция `do_shoot`, которая обработает выстрел, разошлёт результаты на вебхуки и если у игрока не осталось выстрелов, то скажет, кто делает следующий ход.

2.3 Описание структур данных

В коде клиента были реализованы классы `Ship`, `Cell`, `Field`. Класс `Ship` (рисунок 2.4) – корабль, у которого есть поля: `x`, `y` – координаты клетки, в которой находится его основание (первая клетка корабля), `size` – размер (от 1 до 4 клеток), `orient` – направление корабля от основания (по сторонам горизонта), `parts` – список частей корабля, показывающий в каких местах подбили корабль (`parts[i] == True` – часть не подбита, иначе – часть подбита). Также класс содержит поле `img`, хранящее изображение этого корабля (используется для отрисовки игрового поля).

```

class Ship:
    def __init__(self, x, y, size, orient='e', ltl=False):
        self.x = x
        self.y = y
        self.size = size
        self.orient = orient
        self.parts = []
        for i in range(size):
            self.parts.append(True)

        angle = 0
        if orient == 'n': angle = 90
        elif orient == 'w': angle = 180
        elif orient == 's': angle = 270

        if not(ltl):
            if size == 1: self.img = ImageTk.PhotoImage(ship_1_img.rotate(angle, expand=True))
            elif size == 2: self.img = ImageTk.PhotoImage(ship_2_img.rotate(angle, expand=True))
            elif size == 3: self.img = ImageTk.PhotoImage(ship_3_img.rotate(angle, expand=True))
            else: self.img = ImageTk.PhotoImage(ship_4_img.rotate(angle, expand=True))
        else:
            if size == 1: self.img = ImageTk.PhotoImage(ltl_ship_1_img.rotate(angle, expand=True))
            elif size == 2: self.img = ImageTk.PhotoImage(ltl_ship_2_img.rotate(angle, expand=True))
            elif size == 3: self.img = ImageTk.PhotoImage(ltl_ship_3_img.rotate(angle, expand=True))
            else: self.img = ImageTk.PhotoImage(ltl_ship_4_img.rotate(angle, expand=True))

```

Рисунок 2.3. Класс Ship.

Класс Cell (рисунок 2.4) – клетка поля, вспомогательный класс для класса Field. Имеет координаты (x, y), содержит информацию о том, сбита ли она (shooted), содержит ли корабль (contains_ship), а также поле ship_key, содержащее ключ корабля в словаре кораблей (см. класс Field).

```

class Cell:
    def __init__(self, x, y, contains_ship=False):
        self.x = x
        self.y = y
        self.shooted = False
        self.contains_ship = contains_ship
        self.ship_key = None

```

Рисунок 2.4. Класс Cell.

Класс Field (рисунок 2.5) – игровое поле игрока. Содержит двумерный массив клеток (Cell), а также словарь всех расставленных кораблей, в котором ключ – строка “ху”, образованная от координат основания корабля.


```

class Field:
    def __init__(self):
        self.battle_field = []
        for x in range(10):
            self.battle_field.append([])
            for y in range(10):
                self.battle_field[x].append(Cell(x, y))
        self.ships = {} # словарь содержит инфу о всех кораблях на поле ("xy": Ship)

    def check_for_add(self, ship): ...

    def add(self, ship): ...

    def check_area(self, x, y): ...

    def delete(self, ship): ...

    def print_field(self): ...

```

Рисунок 2.4. Класс Cell.

Также в классе Field есть пять методов. Метод `check_for_add` принимает объект класса Ship и возвращает истину, если этот корабль может быть добавлен на поле, в противном случае возвращает ложь. Метод `add` принимает объект класса Ship и добавляет его на поле (предварительно выполняя проверку методом `check_for_add`). Метод `check_area` – вспомогательный метод для `check_for_add`. Метод `delete` – принимает объект класса Ship и удаляет такой корабль с поля (если такой корабль там есть). Метод `print_field` был нужен только на ранних этапах разработки – он печатает поле в консоль из нулей и единиц.

Также был реализован ещё один небольшой класс – ChosenShip. Он хранит информацию о выбранном корабле во время расстановки кораблей.

На сервере тоже были реализованы классы Ship, Cell и Field, но с урезанным функционалом, так как серверу не нужно отрисовывать поле и расставлять корабли. Он принимает от клиента уже готовое поле и просто проверяет его правильность.

Но на сервере был реализован класс Game и вложенный в него класс Player (см. приложение 1).

2.4 Описание основных модулей

Проект состоит из двух модулей – клиентов, с которыми взаимодействуют пользователи, и сервера, который объединяет клиентов.

Клиент состоит из скрипта, написанного на Python 3.9, и набора изображений, используемых в графическом интерфейсе. Таким образом, чтобы запустить клиентское приложение, нужно запустить скрипт (`sea_battle.py`), который должен находиться в папке с изображениями. Также, при работе клиент создаст файл `game_data.json`. Нужно также отметить, что для запуска клиента на компьютер должен быть установлен интерпритатор Python версии 3+ и дополнительные модули: `tkinter`, `Pillow`, `flask`, `requests`.

Серверная часть – только один файл (`server_for_seabattle.exe`). Его достаточно просто запустить. Но при работе он создаст файл `id_reged.json` в той же директории.

Интерфейс клиента был сделан в простом минималистичном стиле. При запуске клиента пользователь попадает на главную страницу (рисунок 2.5), на которой находятся кнопка «Play», поле для ввода никнейма и поля для ввода адреса сервера и вебхука, который слушает клиент. Таким образом пользователи могут играть вместе, используя, например, программу `ngrok`. Нужно запустить сервер, ввести в `ngrok` команду «`http 1234`» и ссылку, которую покажет `ngrok` все пользователи должны ввести в поле «Server URL». Затем нужно запустить клиент, посмотреть в консоли, на каком порту он запущен, и ввести в `ngrok` команду: «`http + порт`». `Ngrok` покажет ссылку, которую пользователь должен будет ввести в поле «Your webhook URL».

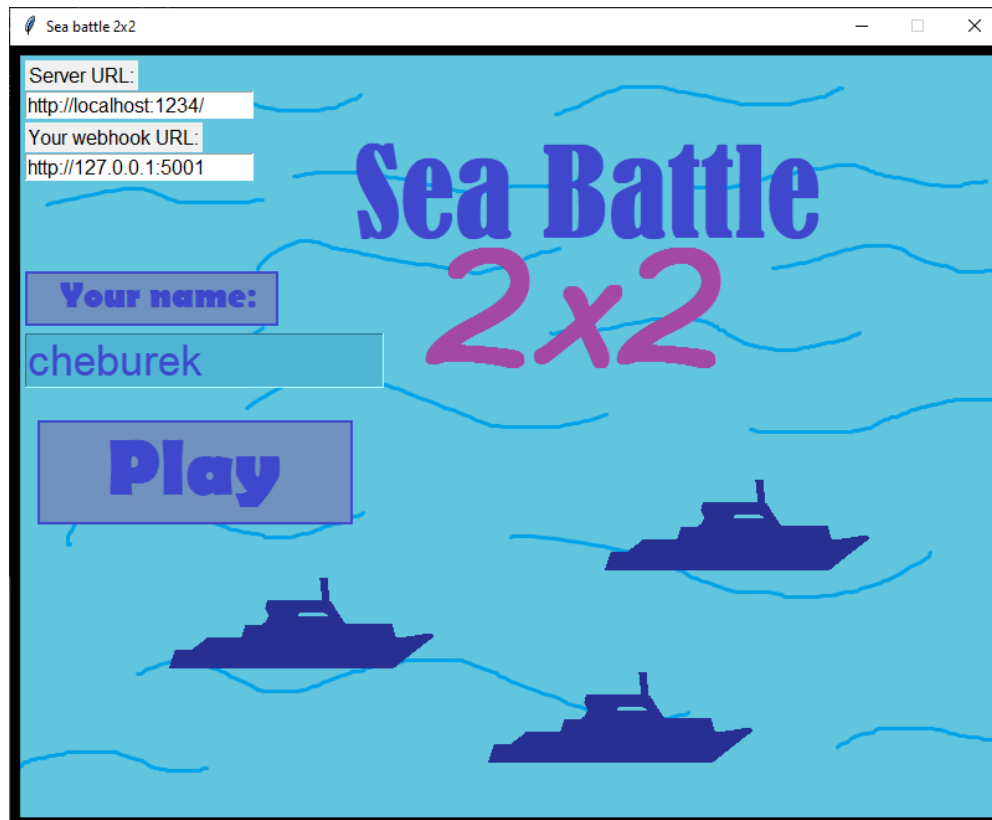


Рисунок 2.5. Главная страница клиента.

Введя данные и нажав «Play», пользователь попадает на страницу выбора игры (рисунок 2.6). На ней отображается список активных игр. Пользователь может нажать на любую игру из списка и присоединится к ней. Также он может присоединится к игре, введя её ID или создать свою игру.

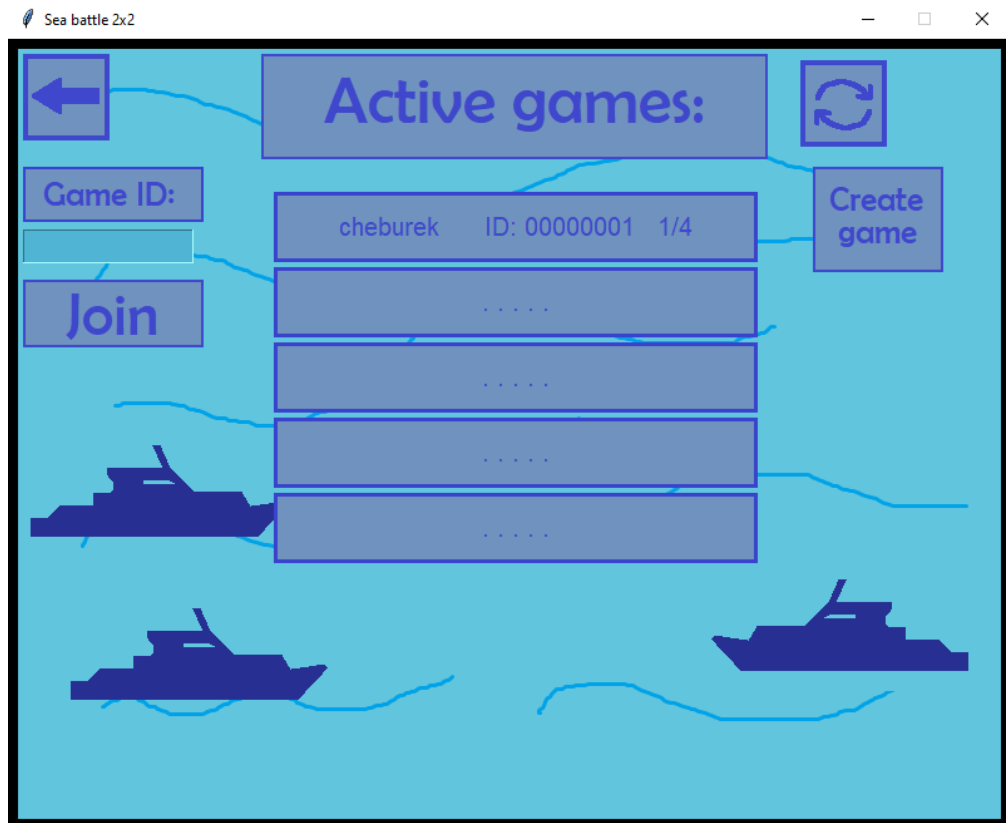


Рисунок 2.6. Страница выбора игры.

Присоединившись к игре, пользователь попадает на страницу подготовки к бою (рисунок 2.7), где он должен расставить корабли и выбрать команду. Когда он это сделает он должен будет нажать на кнопку «Ready», чтобы оповестить всех, что он готов. Когда все игроки будут готовы, они попадут на страницу боя.

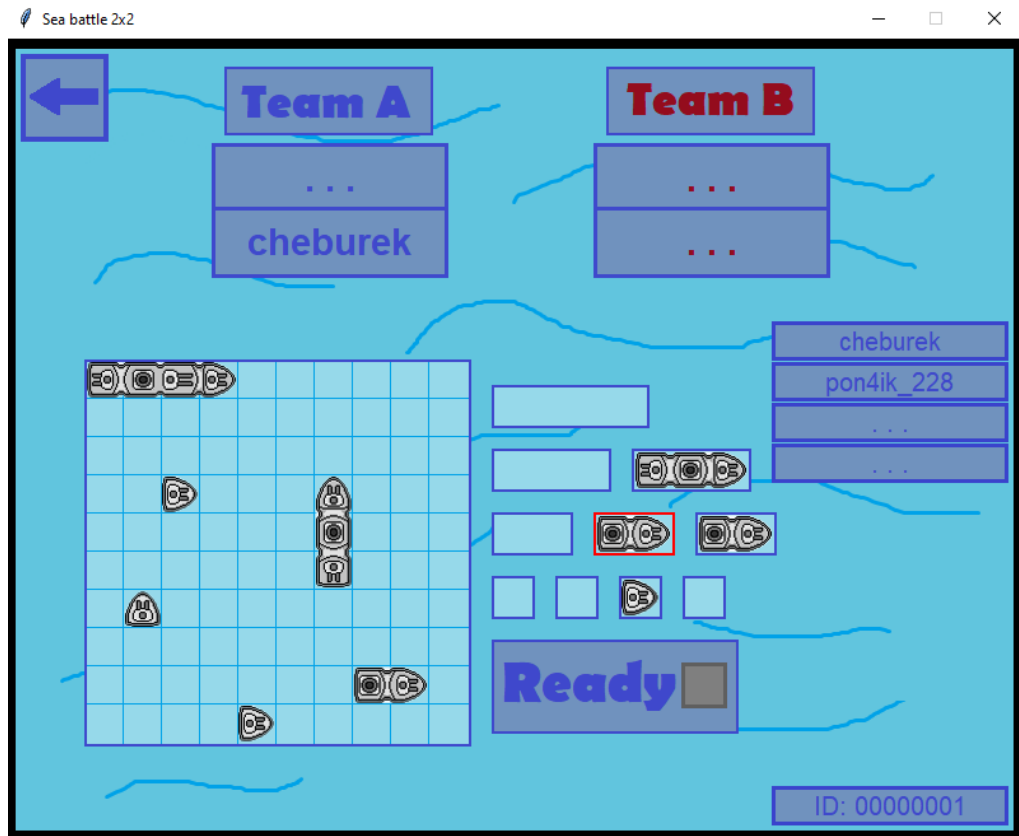


Рисунок 2.6. Страница выбора игры.

На странице боя игроки ходят по очереди, делая по 4 выстрела за ход. Количество выстрелов у игрока уменьшается, когда ему сбивают корабли. Игрок, который делает ход, подсвечивается зелёным.

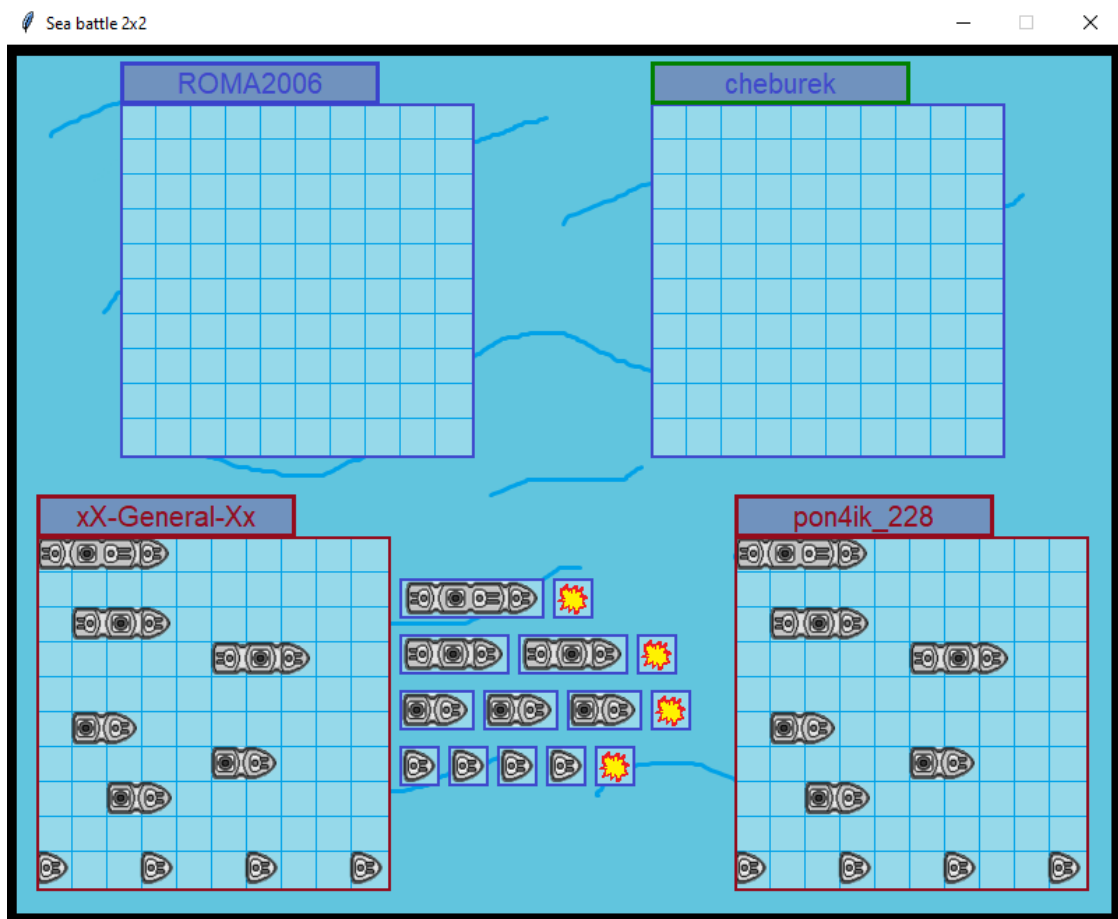


Рисунок 2.7. Страница боя.

ГЛАВА 3

ТЕСТИРОВАНИЕ ПРОГРАММЫ

3.1 Тестирование исходного кода

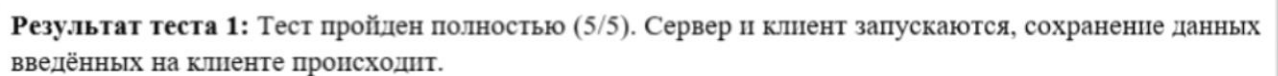
Проект состоит из двух модулей: клиента и сервера. Тестами был покрыт основной функционал клиентской части, и соответственно функции сервера, связанные с этим функционалом. Всего было проведено пять тестов. Полное содержание тест-кейсов для проекта приведено в приложении 2.

Список проведённых тестов:

- Проверка выполнения запросов и сохранения данных
- Проверка возможности создания и присоединения к играм
- Проверка алгоритмов расстановки кораблей
- Проверка возможности распределения по командам
- Проверка процесса боя

С помощью этих тестов работоспособность приложения была проверена. Была доказана правильная работа всех его необходимых функций. Все тесты были успешно пройдены.

Результат теста 1, направленного на проверку выполнения запросов и сохранения данных, приведён на рисунке 3.1.



Результат теста 1: Тест пройден полностью (5/5). Сервер и клиент запускаются, сохранение данных введённых на клиенте происходит.

Рисунок 3.1. Результат теста 1.

Результат теста 2, направленного на проверку возможности создания и присоединения к играм, приведён на рисунке 3.2.

Результат теста 2: Тест пройден полностью (9/9). Игры создаются и отображаются в списке. К играм можно присоединиться как кликнув на них в списке, так и с помощью ввода ID.

Рисунок 3.2. Результат теста 2.

Результат теста 3, направленного на проверку алгоритмов расстановки кораблей, приведён на рисунке 3.3.

Результат теста 3: Тест пройден полностью (6/6). Корабли расставляются и поворачиваются правильно. Не допускается расстановка кораблей в недоступные места.

Рисунок 3.3. Результат теста 3.

Результат теста 4, направленного на проверку алгоритмов распределения по командам, приведён на рисунке 3.4.

Результат теста 4: Тест пройден полностью (4/4). Присоединение к командам и выход из них, а также синхронизация информации между клиентами происходят правильно.

Рисунок 3.4. Результат теста 4.

Результат теста 5, направленного на проверку алгоритмов, отвечающих за проведение боя, приведён на рисунке 3.5.

Результат теста 5: Тест пройден полностью (4/4). Выстрелы, перемена ходов и синхронизация действий игроков происходят правильно.

Рисунок 3.5. Результат теста 5.

3.2 Тестирование интерфейса пользователя и юзабилити

Тестирование показало, что пользовательский интерфейс довольно удобен и в основном интуитивно понятен. Однако следует добавить больше подсказок для пользователя при его неправильных действиях. Например, если он вводит ID игры, которой не существует.

Интерфейс выглядит минималистично и довольно приятно. Однако для того, чтобы улучшить впечатление пользователя от игры, следует добавить плавные переходы между страницами и другие анимации. Также не лишним будет добавление звукового сопровождения.

ГЛАВА 4

ПЕРСПЕКТИВЫ ДАЛЬНЕЙШЕГО РАЗВИТИЯ ПРОЕКТА

4.1 Перспективы технического развития

Были реализованы все требуемые функции для проекта: пользователь может создавать игры, присоединится к игре из списка активных игр или по ID, свободно выбрать команду. Однако у него остаётся масса возможностей для дальнейшего развития. Одни из самых важных функций, которые стоило бы добавить – это возможность добавлять ботов вместо реальных игроков. Также можно добавить режимы игры для двоих и для одного человека. Ещё одним перспективным дополнением может стать добавление в игру валюты и магазина скинов и бонусов.

Важно отметить, что в последней версии проекта так и не был реализован автоматический алгоритм для игры по сети. На данный момент игра по сети с разных устройств возможна только с использованием сторонних программ (ngrok).

Ещё стоит улучшить систему расстановки кораблей перед боем. Так как на данный момент корабли можно вращать только против часовой стрелки. Также их не всегда удобно перемещать мышкой. Поэтому следует добавить способ, позволяющий вращать и перемещать корабли с помощью кнопок на клавиатуре.

4.2 Перспективы монетизации

На данный момент не реализовано никаких способов монетизации проекта. Чтобы стало возможным хоть немного монетизировать проект, его нужно доработать и добавить больше функционала. На данный момент проект слишком прост, чтобы его можно было монетизировать.

ЗАКЛЮЧЕНИЕ

В результате было создано приложение, отвечающее всем заявленным требованиям. На выходе проекта, как и планировалось, было получено два модуля: клиент и сервер. Функционал приложения точно соответствует требованиям технического задания.

Приложение «Sea Battle 2x2» позволяет пользователям играть в морской бой в командах по два человека друг против друга. Среди функций, выполняемых приложением можно выделить:

- Создание пользователем игры
- Просмотр списка активных игр
- Возможность присоединиться к игре из списка активных игр или по ID
- Возможность свободно распределяться по командам
- Расстановка кораблей и возможность её изменения
- Ведение командного боя два против двух

В процессе создания проекта был получен новый опыт и закреплены старые навыки по разработке приложений с клиент-серверной архитектурой. Были получены практические умения по использованию вебхуков. Также были получены базовые умения по использованию микрофреймворка flask на Python. Также было получено много новых знаний и закреплены старые умения по созданию приложений с графическим интерфейсом с помощью tkinter и работы с изображениями в библиотеке Pillow.

ЛИТЕРАТУРА

1. Документация по C++ [Электронный ресурс] // Docs.Microsoft.com 2021 – URL: <https://docs.microsoft.com/ru-ru/cpp/cpp/?view=msvc-160>
2. Документация по Python 3.9.2 [Электронный ресурс] // docs.python.org 2021 – URL: <https://docs.python.org/3/>
3. Оформление выпускной квалификационной работы на соискание квалификационного уровня «Магистр» («Бакалавр»): методические рекомендации. / сост. Бержанский В.Н., Дзедолик И.В., Полулях С.Н. – Симферополь: КФУ им. В.И.Вернадского, 2019. – 31 с.
4. Документация по библиотеке `cpp_httplib` для работы с сетью в C++ [Электронный ресурс] // github.com/yhirose – URL: <https://github.com/yhirose/cpp-httplib>
5. Документация по библиотеке `tkinter` для Python 3.9.5 [Электронный ресурс] // docs.python.org 2021 – URL: <https://docs.python.org/3/library/tkinter.html>

ПРИЛОЖЕНИЕ 1

КЛАСС GAME СЕРВЕРНОЙ ЧАСТИ

```

class Game {
public:
    //enum ERR {};    <-- in future
    class Ship {
    public:
        int x, y, size;
        std::string orient;
        std::vector<bool> parts;
        Ship(int x, int y, int size, std::string orient = "e") : x(x), y(y), size(size),
orient(orient) {}
        Ship() : x(0), y(0), size(1), orient("e") {}
    };
    class Player {
    public:
        class Field {
        public:
            class Cell {
            public:
                int x, y;
                bool shoted = false;
                bool contains_ship = false;
                std::string ship_key = "";
                Cell(int x = 0, int y = 0) : x(x), y(y) {}
            };
            Cell battle_field[10][10];
            std::map<std::string, Ship> ships;
            Field() {
                for (int x = 0; x < 10; x++) {
                    for (int y = 0; y < 10; y++) {
                        battle_field[x][y] = Cell(x, y);
                    }
                }
            }
            bool check_area(int x, int y) {
                if (battle_field[x][y].contains_ship) return false;
                if (x < 9 && battle_field[x + 1][y].contains_ship) return false;
                if (x < 9 && y > 0 && battle_field[x + 1][y - 1].contains_ship) return false;
                if (y > 0 && battle_field[x][y - 1].contains_ship) return false;
                if (x > 0 && y > 0 && battle_field[x - 1][y - 1].contains_ship) return false;
                if (x > 0 && battle_field[x - 1][y].contains_ship) return false;
                if (x > 0 && y < 9 && battle_field[x - 1][y + 1].contains_ship) return false;
                if (y < 9 && battle_field[x][y + 1].contains_ship) return false;
                if (x < 9 && y < 9 && battle_field[x + 1][y + 1].contains_ship) return false;
                return true;
            }
            bool check_for_add(Ship ship) {
                if (ship.orient == "e") {
                    if ((ship.x + ship.size - 1) < 10) {
                        for (int i = 0; i < ship.size; i++) {
                            if (!(this->check_area(ship.x + i, ship.y))) return false;
                        }
                    }
                    else return false;
                }
                if (ship.orient == "n") {
                    if ((ship.y - ship.size + 1) >= 0) {
                        for (int i = 0; i < ship.size; i++) {
                            if (!(this->check_area(ship.x, ship.y - i))) return false;
                        }
                    }
                }
            }
        };
    };
};

```

```

    }
    }
    else return false;
}
if (ship.orient == "w") {
    if ((ship.x - ship.size + 1) >= 0) {
        for (int i = 0; i < ship.size; i++) {
            if (!(this->check_area(ship.x - i, ship.y))) return false;
        }
    }
    else return false;
}
if (ship.orient == "s") {
    if ((ship.y + ship.size - 1) < 10) {
        for (int i = 0; i < ship.size; i++) {
            if (!(this->check_area(ship.x, ship.y + i))) return false;
        }
    }
    else return false;
}
return true;
}
bool add(Ship ship) {
    if (!(this->check_for_add(ship))) return false;
    std::string ship_key = std::to_string(ship.x) + std::to_string(ship.y);
    this->ships[ship_key] = ship;
    if (ship.orient == "e") {
        for (int i = 0; i < ship.size; i++) {
            battle_field[ship.x + i][ship.y].contains_ship = true;
            battle_field[ship.x + i][ship.y].ship_key = ship_key;
        }
    }
    if (ship.orient == "n") {
        for (int i = 0; i < ship.size; i++) {
            battle_field[ship.x][ship.y - i].contains_ship = true;
            battle_field[ship.x][ship.y - i].ship_key = ship_key;
        }
    }
    if (ship.orient == "w") {
        for (int i = 0; i < ship.size; i++) {
            battle_field[ship.x - i][ship.y].contains_ship = true;
            battle_field[ship.x - i][ship.y].ship_key = ship_key;
        }
    }
    if (ship.orient == "s") {
        for (int i = 0; i < ship.size; i++) {
            battle_field[ship.x][ship.y + i].contains_ship = true;
            battle_field[ship.x][ship.y + i].ship_key = ship_key;
        }
    }
    return true;
}
};
std::string player_id;
std::string player_name;
std::string webhook;
bool ready = false;
Field field;
Player(std::string id, std::string name, std::string webhook): player_id(id),
player_name(name), webhook(webhook) {}

int fill_field(std::vector<Ship> ships) {
    int count[4] = { 0,0,0,0 };
    for (Ship ship : ships) {

```

```

        if (ship.size > 0 && ship.size < 5) count[ship.size - 1]++;
        else return 1; // incorrect ship size
    }
    if (!(count[0] == 4 && count[1] == 3 && count[2] == 2 && count[3] == 1)) return 2;
// wrong amount of ships
    for (Ship ship : ships) {
        if (!(this->field.add(ship))) {
            this->field = Field();
            return 3; // wrong placement of ships
        }
    }
    return 0;
}
};
private:
    std::string game_id;
    std::vector<Player> players;
    std::vector<std::string> team_a{ "0", "0" };
    std::vector<std::string> team_b{ "0", "0" };
    bool started = false;
    int turn; // 1, 2, 3, 4;
    std::string gen_id() {
        return id_up(last_id);
    }
public:
    static std::string last_id;
    Game() {
        game_id = gen_id();
        last_id = game_id;
        std::cout << "created game with ID: " << game_id << "\n";
    }
    std::string get_id() {
        return game_id;
    }
    int add_player(std::string player_id, std::string player_name, std::string player_webhook)
    {
        for (Player p : players) {
            if (p.player_id == player_id) return 2;
        }
        if (player_webhook[player_webhook.size()-1] == '/')
            player_webhook.erase(player_webhook.size()-1, 1);
        Player p(player_id, player_name, player_webhook);
        if (players.size() < 4) {
            players.push_back(p);
            return 0;
        }
        return 1;
    }
    void remove_player(std::string player_id) {
        for (std::vector<Player>::iterator it = players.begin(); it != players.end(); ++it){
            if ((*it).player_id == player_id) {
                players.erase(it);
                break;
            }
        }
        if (this->team_a[0] == player_id) team_a[0] = "0";
        else if (this->team_a[1] == player_id) team_a[1] = "0";
        else if (this->team_b[0] == player_id) team_b[0] = "0";
        else if (this->team_b[1] == player_id) team_b[1] = "0";
    }
    int get_p_count() {
        return players.size();
    }
    std::string get_admin() {

```

```

        if (players.size()) return players[0].player_name;
        else return "";
    }
    int remove_from_team(std::string player_id) {
        for (Player& p : players) {
            if (p.player_id == player_id) {
                if (p.ready) return 1;
                break;
            }
        }
        if (team_a[0] == player_id) team_a[0] = "0";
        if (team_a[1] == player_id) team_a[1] = "0";
        if (team_b[0] == player_id) team_b[0] = "0";
        if (team_b[1] == player_id) team_b[1] = "0";
        return 0;
    }
    int add_to_team_a(std::string player_id, int num) {
        bool p_exist = false;
        for (Player& p : players) {
            if (p.player_id == player_id) {
                if (p.ready) return 4;
                p_exist = true;
                break;
            }
        }
        if (!p_exist) return 2;
        if (num > 1 || num < 0) return 3;
        this->remove_from_team(player_id);
        if (this->team_a[num] == "0") {
            this->team_a[num] = player_id;
            return 0;
        }
        return 1;
    }
    int add_to_team_b(std::string player_id, int num) {
        bool p_exist = false;
        for (Player& p : players) {
            if (p.player_id == player_id) {
                if (p.ready) return 4;
                p_exist = true;
                break;
            }
        }
        if (!p_exist) return 2;
        if (num > 1 || num < 0) return 3;
        this->remove_from_team(player_id);
        if (this->team_b[num] == "0") {
            this->team_b[num] = player_id;
            return 0;
        }
        return 1;
    }
    int set_ready(std::string player_id, std::vector<Ship> ships) {
        for (Player& p : players) {
            if (p.player_id == player_id) {
                if (p.ready) return 5; // player already ready
                p.field = Player::Field();
                int i = p.fill_field(ships);
                if (i == 0) p.ready = true;
                return i;
            }
        }
        return 4; // no player in game
    }
}

```



```

int deselect_ready(std::string player_id) {
    for (Player& p : players) {
        if (p.player_id == player_id) {
            p.ready = false;
            return 0;
        }
    }
    return 1; // no player in game
}

bool all_ready() {
    return (players.size() == 4
            && players[0].ready
            && players[1].ready
            && players[2].ready
            && players[3].ready);
}

bool have_started() {
    return started;
}

void start_game() {
    started = true;
    turn = (rand() % 4) + 1;
}

std::vector<std::string> get_webhooks(std::string exclude = "") {
    std::vector<std::string> webhooks;
    for (Player p : players) {
        if (p.player_id != exclude) webhooks.push_back(p.webhook);
    }
    return webhooks;
}

json get_data_for_new_player() {
    json data = json::object();
    data["players"] = json::array();
    int i = 0;
    for (Player p : players) {
        data["players"].push_back(json::object());
        data["players"][i]["player_id"] = p.player_id;
        data["players"][i]["player_name"] = p.player_name;
        i++;
    }
    data["team_a"] = json::array();
    data["team_a"].push_back(team_a[0]);
    data["team_a"].push_back(team_a[1]);

    data["team_b"] = json::array();
    data["team_b"].push_back(team_a[0]);
    data["team_b"].push_back(team_a[1]);
    return data;
}

json get_teams_data() {
    json data = json::object();
    data["team_a"] = json::array();
    data["team_a"].push_back(team_a[0]);
    data["team_a"].push_back(team_a[1]);

    data["team_b"] = json::array();
    data["team_b"].push_back(team_b[0]);
    data["team_b"].push_back(team_b[1]);
    return data;
}

std::map<std::string, json> get_data_for_start() { // {webhook: data}
    std::map<std::string, json> data;
    for (Player& p : players) {
        json j = json::object();

```

```

j["friend"] = json::object();
if (team_a[0] == p.player_id) j["friend"]["id"] = team_a[1];
if (team_a[1] == p.player_id) j["friend"]["id"] = team_a[0];
if (team_b[0] == p.player_id) j["friend"]["id"] = team_b[1];
if (team_b[1] == p.player_id) j["friend"]["id"] = team_b[0];

Player::Field friend_field;
for (Player& fr_player : players) {
    if (fr_player.player_id == j["friend"]["id"]) {
        friend_field = fr_player.field;
        break;
    }
}
j["friend"]["ships"] = json::array();
int i = 0;
for (auto& ship : friend_field.ships) {
    j["friend"]["ships"].push_back(json::object());
    j["friend"]["ships"][i]["x"] = ship.second.x;
    j["friend"]["ships"][i]["y"] = ship.second.y;
    j["friend"]["ships"][i]["size"] = ship.second.size;
    j["friend"]["ships"][i]["orient"] = ship.second.orient;
    i++;
}

if (team_a[0] == p.player_id || team_a[1] == p.player_id) {
    j["enemy_1"] = team_b[0];
    j["enemy_2"] = team_b[1];
}
else {
    j["enemy_1"] = team_a[0];
    j["enemy_2"] = team_a[1];
}

switch (turn)
{
case (1):
    j["turn"] = team_a[0];
    break;
case (2):
    j["turn"] = team_a[1];
    break;
case (3):
    j["turn"] = team_b[0];
    break;
case (4):
    j["turn"] = team_b[1];
    break;
}
data[p.webhook] = j;
}
return data;
}
};
std::string Game::last_id = "00000000";

```

ПРИЛОЖЕНИЕ 2

ТЕСТ-КЕЙСЫ ПРОЕКТА

Тест-план по тестированию проекта «Sea Battle 2x2»

Общая информация

Информация о проекте	
Название проекта	Sea Battle 2x2
Автор	Баламутин Егор Никитович ПИ-201(1)
Е-mail автора	Yegorbalamutin1@gmail.com
Связь с автором	Vk: https://vk.com/id252395328
Ссылка на проект	Github: https://github.com/Yegor-Balamutin/CFU_tasks/tree/master/Coursework
Дата релиза	31.05.2021

Содержание тестов

Тест	Описание
Тест 1	Проверка выполнения запросов и сохранения данных
Тест 2	Проверка возможности создания и присоединения к играм
Тест 3	Проверка алгоритмов расстановки кораблей
Тест 4	Проверка возможности распределения по командам
Тест 5	Проверка процесса боя

Тест 1

Шаг №	Действие	Ожидаемый результат	Реальный результат	Пройден / не пройден	Комментарий
1	Запустить сервер	Сервер запускается и выводится его адрес	Сервер запустился и вывел информацию	пройден	
2	Запустить клиент	Открывается окно интерфейса и консоль	Оба окна открылись	пройден	
3	Ввести в соответствии с полями адрес сервера, адрес клиента (строка «Running on» в консоли) и никнейм. Нажать «play».	На клиенте откроется окно со списком доступных игр (список пустой). На сервер придёт запрос от клиента, сервер выведет в консоль информацию (в том числе присвоенное клиенту ID)	Данные были введены. После нажатия кнопки «play» открылась страница с пустым списком, также содержащая кнопки «Create game» и «join».	пройден	
4	Закрыть окно клиента и консоль. Затем снова его запустить.	Откроется консоль и окно интерфейса. Данные, введённые в прошлый раз, будут автоматически занесены в соответствующие поля.	Окна открылись, данные вписались.	пройден	
5	Изменить данные (например, никнейм), нажать «play» и закрыть клиент. Затем снова его запустить.	При следующем запуске клиента в полях появятся данные, введённые в последний раз.	При повторном запуске в полях появились последние введённые данные.	пройден	

Результат теста 1: Тест пройден полностью (5/5). Сервер и клиент запускаются, сохранение данных, введённых на клиенте происходит.

Тест 2

Шаг №	Действие	Ожидаемый результат	Реальный результат	Пройден / не пройден	Комментарий
1	При запущенном сервере, открыть клиент и нажать «play». Затем нажать кнопку «Create game».	Пользователь попадёт на страницу подготовки к бою. Он увидит свой никнейм в списке игроков, а также ID игры.	Страница открылась. Никнейм и ID были видны.	пройден	
2	Запустить второй клиент, ввести данные и нажать «play»	Список активных игр будет содержать игру, созданную клиентом 1. Будут отображаться ник создателя, ID игры и количество игроков.	Игра была видна, данные были корректны.	пройден	
3	Запустить третий клиент, ввести данные и нажать «play»	Ожидаемый результат аналогичен результату из предыдущего шага.	Игра была видна, данные были корректны.	пройден	
4	На втором клиенте нажать на игру в списке.	Пользователь попадёт на страницу подготовки к бою. В списке игроков он будет видеть свой ник и ник создателя игры.	Список игроков содержал никнейм создателя и никнейм пользователя.	пройден	
5	На третьем клиенте нажать кнопку «обновить» (кнопка с двумя круглыми стрелками)	Список активных игр обновится. Количество игроков в игре станет равно двум.	Список обновился, количество игроков стало равно двум.	пройден	

6	На третьем клиенте нажать на игру в списке.	Пользователь присоединится к игре, в списке игроков он увидит себя и двух других игроков. Другие игроки также увидят появление нового игрока.	Пользователь присоединился к игре. На всех трёх клиентах в списке отобразилось по три игрока.	пройден	
7	Запустить четвёртый клиент, ввести данные и нажать «play»	Пользователь увидит в списке игру с тремя игроками	Пользователь увидел в списке игру с тремя игроками.	пройден	
8	На четвёртом клиенте ввести в поле «Game ID» ID игры, которое он видит в списке. Нажать «Join».	Пользователь присоединится к игре. В списке игроков будут ники всех четырёх участников.	Пользователь присоединился, его имя появилось в списке на всех четырёх клиентах.	пройден	
9	На любом из клиентов нажать стрелочку «назад».	Пользователь выйдет на страницу со списком игр. Остальные игроки увидят, что его ник исчез из списка.	Все пользователи увидели, что игрок покинул игру.	пройден	

Результат теста 2: Тест пройден полностью (9/9). Игры создаются и отображаются в списке. К играм можно присоединиться как кликнув на них в списке, так и с помощью ввода ID.

Тест 3

Шаг №	Действие	Ожидаемый результат	Реальный результат	Пройден / не пройден	Комментарий
1	На любом из клиентов, находясь на странице подготовки к бою, нажать на любой из кораблей рядом с полем.	Корабль выделится красной рамкой	Корабль выделился.	пройден	
2	Затем нажать на любую клетку в середине поля.	Выбранный корабль пропадёт со своего места появится на поле.	Корабль переместился на поле.	пройден	
3	Выбрать другой корабль и нажать на клетку рядом с поставленным кораблём.	В месте клика на пол секунды появится полупрозрачный корабль, показывающий, что на это место его поставить нельзя.	Полупрозрачный корабль появляется.	пройден	
4	Нажать правой кнопкой на поставленный на поле корабль.	Корабль должен повернуться на 90 градусов.	Корабль поворачивается.	пройден	
5	Нажать на корабль на поле. Затем нажать на место, на котором он находился до того, как был поставлен на поле.	Корабль должен пропасть с поля и вернуться на своё место вне поля.	Корабль вернулся в свою ячейку.	пройден	

6	Порасставлять корабли, поперемещать и покрутить их. Попробовать поставить корабль в недоступное место.	Корабли будут свободно изменять свою позицию, если им ничего не мешает.	Корабли свободно перемещаются в доступные для них места. Поставить корабль в недопустимое место не удалось	пройден	Не всегда удобно вращать и перемещать корабли. Было бы удобнее делать это с помощью кнопок на клавиатуре.
---	--	---	--	---------	---

Результат теста 3: Тест пройден полностью (6/6). Корабли расставляются и поворачиваются правильно. Не допускается расстановка кораблей в недоступные места.

Тест 4

Шаг №	Действие	Ожидаемый результат	Реальный результат	Прошел / не прошел	Комментарий
1	Находясь на странице подготовки к бою, нажать на свободную ячейку под надписью «Team A» или «Team B»	Имя игрока появится в этой ячейке. Другие игроки также увидят это.	Имя появилось в ячейке на всех клиентах.	пройден	
2	Повторно нажать на ту же самую ячейку.	Имя пользователя пропадет из ячейки для всех игроков.	Все игроки увидели, что имя пропало.	пройден	
3	Находясь в команде, нажать на другую свободную ячейку	Имя игрока пропадет из старой ячейки и появится в новой.	Имя пропало из старой ячейки и появилось в новой.	пройден	
4	С разных клиентов повступать и повыходить из команд.	Результаты должны соответствовать результатам предыдущих двух шагов.	Всё работало правильно.	пройден	

Результат теста 4: Тест пройден полностью (4/4). Присоединение к командам и выход из них, а также синхронизация информации между клиентами происходят правильно.

Тест 5

Шаг №	Действие	Ожидаемый результат	Реальный результат	Пройден / не пройден	Комментарий
1	После расстановки всех кораблей и вступления в команду, нажать на кнопку «Ready».	На кнопке появится галочка. Рамка вокруг имени игрока в списке станет зелёной для всех пользователей.	Галочка появилась. Имя в списке стало обведено зелёным.	пройден	
2	Четыре игрока должны находится в одной игре. Каждый должен расставить корабли, вступить в команду и заявить о готовности.	После того, как последний игрок заявляет о готовности, все игроки попадают на страницу боя. Каждый игрок видит свои корабли и корабли напарника.	Все пользователи попали на страницу боя. Расстановка кораблей была правильной.	пройден	
3	На клиенте, где имя пользователя находится в зелёной рамке, нажать на любое из полей противников.	Игрок выстрелит в клетку, на которую он нажал. Если в клетке не было корабля, то она заштрихуется синим. Если был – на ней появится красный крест. Результат выстрела увидят все пользователи.	Выстрел происходит. Результат виден всем.	пройден	
4	Выстрелить ещё 3 раза.	После четырёх выстрелов право хода передаётся следующему игроку.	Право хода передаётся.	пройден	

Результат теста 5: Тест пройден полностью (4/4). Выстрелы, перемена ходов и синхронизация действий игроков происходят правильно.