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

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

РАЗРАБОТКА СЕТЕВОЙ ИГРЫ «МОРСКОЙ БОЙ»

БГУИР КР 6 - 05 - 06 12 01 011 ПЗ

Выполнил: студент группы 310901 Клещёв В. А.

Проверила: Болтак С. В.

Минск 2025

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc199964504)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 4](#_Toc199964505)

[1.1 Обзор аналогов 4](#_Toc199964506)

[1.2 Постановка задачи 6](#_Toc199964507)

[2 ПРОЕКТИРОВАНИЕ ЗАДАЧИ. 7](#_Toc199964508)

[2.1 Структура программы 7](#_Toc199964509)

[2.2 Проектирование интерфейса программного средства 8](#_Toc199964510)

[2.2.1 Регистрация и авторизация 8](#_Toc199964511)

[2.2.2 Процесс игры 10](#_Toc199964512)

[2.3 Проектирование функционала программного средства 12](#_Toc199964513)

[3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА 19](#_Toc199964514)

[3.1 Система аутентификации 19](#_Toc199964515)

[3.2 Клиент-серверная модель 19](#_Toc199964516)

[3.3 Создание игровой сессии 20](#_Toc199964517)

[3.4 Игровой процесс 21](#_Toc199964518)

[4 ТЕСТИРОВАНИЕ 22](#_Toc199964519)

[5 ПРИМЕНЕНИЕ ПРОГРАММЫ 24](#_Toc199964520)

[5.1 Руководство пользователя 24](#_Toc199964521)

[ЗАКЛЮЧЕНИЕ 26](#_Toc199964522)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 28](#_Toc199964523)

[ПРИЛОЖЕНИЕ А 29](#_Toc199964524)

# **ВВЕДЕНИЕ**

Интернет и современные технологии стремительно развиваются, оказывая значительное влияние на все сферы нашей жизни. Одной из наиболее популярных областей их применения являются компьютерные игры, которые давно вышли за рамки простого развлечения и стали важным способом взаимодействия между людьми по всему миру. Особенно востребованы сетевые игры, позволяющие игрокам соревноваться и сотрудничать в режиме реального времени, что делает игровой процесс более увлекательным и социально ориентированным.

Сетевые игры представляют собой сложные программные продукты, требующие использования современных технологий, глубоких знаний в области программирования и сетевого взаимодействия. Интерес к их разработке неуклонно растёт благодаря доступности языков программирования, инструментов и платформ, которые упрощают создание интерактивных приложений.

Одной из таких игр, которая сохраняет свою популярность на протяжении многих лет, является «Морской бой». Эта классическая стратегическая игра, известная многим с детства, изначально существовала в бумажной версии, но с развитием технологий обрела множество цифровых интерпретаций. Привлекательность «Морского боя» заключается в его простых правилах, увлекательном игровом процессе и возможности соревноваться с другими игроками, что делает её одинаково интересной как для детей, так и для взрослых.

Данная курсовая работа посвящена разработке сетевой версии игры «Морской бой». Особенностью этой реализации станет возможность взаимодействия двух игроков через локальную сеть или интернет. В рамках работы предполагается создание удобного пользовательского интерфейса и реализация основных игровых механик, таких как расстановка кораблей, пошаговые ходы, отображение результатов попаданий и определение победителя.

Разработка сетевой версии «Морского боя» представляет собой интересную задачу, которая объединяет в себе программирование, изучение сетевых технологий и проектирование пользовательских интерфейсов. Этот проект демонстрирует, как классические настольные игры могут быть адаптированы к современным условиям с использованием современных инструментов разработки, что позволяет создавать новые формы взаимодействия и досуга для пользователей.

# **АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

## **Обзор аналогов**

Сегодня существует множество реализаций игры «Морской бой» в различных форматах: от мобильных приложений до онлайн-платформ. Все они предоставляют игрокам возможность соревноваться друг с другом, используя классические правила игры.

Одним из самых популярных аналогов является игра «*Sea Battle 2*». Это мобильное приложение, доступное для платформ *iOS* и *Android*, которое предлагает игрокам расширенную версию классического «Морского боя». В «*Sea Battle 2*» доступно несколько режимов игры, включая одиночную игру с искусственным интеллектом, локальный мультиплеер через *Bluetooth* и сетевую игру через интернет.

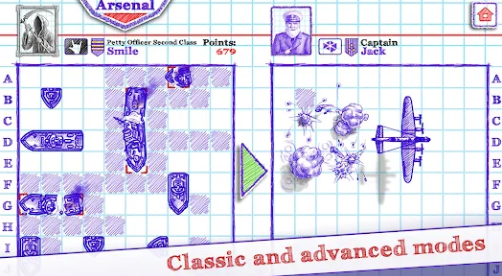


Рисунок 1.1 –

Одной из ключевых особенностей приложения является наличие дополнительных элементов, таких как самолеты, мины, радары и другие боевые единицы, которые значительно разнообразят игровой процесс и требуют от игроков более глубокой стратегической подготовки. Также приложение предоставляет возможность персонализации: игроки могут настраивать внешний вид игрового поля, кораблей и выбирать стиль анимации для максимального погружения в атмосферу морского сражения. Встроенные социальные функции, такие как чат, списки друзей и рейтинговые таблицы, делают игру более интерактивной и увлекательной. Благодаря своему современному подходу и динамичному игровому процессу «*Sea Battle 2*» идеально подходит для игроков, которые ищут обновленную версию классической игры.

Другим популярным аналогом является браузерная игра «*BattleShips Online*», которая позволяет играть без необходимости установки программного обеспечения. Для игры требуется только браузер и доступ к интернету, что делает платформу доступной для пользователей с любыми устройствами, включая персональные компьютеры и мобильные телефоны. Эта версия «Морского боя» реализует классические правила без дополнительных элементов, предлагая игрокам традиционный игровой процесс.



Рисунок 1.2 –

На платформе также можно создавать комнаты для игры с друзьями или присоединяться к открытым комнатам для игры с случайными противниками. Ведется статистика побед и поражений, что добавляет соревновательный элемент. Интерфейс платформы выполнен в минималистичном стиле, что позволяет игрокам сосредоточиться на игровом процессе без отвлечения на лишние визуальные элементы. Простота и доступность делают «*BattleShips Online*» идеальным выбором для тех, кто предпочитает классическую версию игры без дополнительных усложнений.

Таким образом, оба аналога предоставляют игрокам возможность насладиться увлекательными морскими сражениями, но ориентированы на разные аудитории. «*Sea Battle 2*» предлагает современный и разнообразный игровой опыт, тогда как «*BattleShips Online*» фокусируется на доступности и простоте традиционного игрового процесса.

## **Постановка задачи**

В рамках данной курсовой работы необходимо разработать сетевую игру «Морской бой». В перечень задач при разработке игры входит:

* изучить правила и механики классической игры «Морской бой»;
* провести анализ существующих реализаций игры в онлайн-формате;
* спроектировать и реализовать игровой интерфейс;
* определить функциональные требования к игре.

Основные функциональные требования игры включают реализацию механики игрового процесса:

* создание игровой комнаты для двух игроков;
* возможность подключения игроков через локальную сеть или интернет;
* расстановку кораблей на игровом поле с соблюдением правил;
* пошаговый игровой процесс с фиксацией результатов попаданий и промахов;
* отображение изменений на игровом поле в реальном времени;
* проверку условий завершения игры;
* реализация завершения игры с отображением победителя.

Помимо основных функций игрового процесса необходимо реализовать интуитивно понятный интерфейс, который включает:

* основной игровой экран с отображением двух игровых полей, а также индикатора текущего хода;
* интерфейс для расстановки кораблей перед началом игры с визуальными подсказками;
* экран завершения игры с итогами.

За синхронизацию и автоматизацию процессов будет отвечать серверная часть, которая реализует следующие функции:

* подключение двух игроков к одной игровой комнате;
* управление игровыми данными;
* синхронизацию игровых событий между клиентами;
* проверку условий победы и завершения игры.

# **ПРОЕКТИРОВАНИЕ ЗАДАЧИ**.

## **Структура программы**

В данном пункте, будут приведены все модули, а также классы и функции, которые были реализованы в модулях.

*Auth* – модуль отвечает за авторизацию пользователей в игре, включая регистрацию, вход и обработку данных. Он обеспечивает проверку логинов и паролей, хранение информации в файле и графический интерфейс с интерактивными элементами.

*Constans* – модуль определяет константы и параметры интерфейса для игры, включая цвета, размеры экрана, шрифты и интервалы. Эти данные используются в различных частях игрового проекта, обеспечивая единообразный стиль и удобство работы с визуальными элементами.

*Game\_Objects* – модуль отвечает за отрисовку интерактивных элементов игры, включая кнопки и игровую сетку. Он добавляет эффекты анимации, обработку пользовательских действий и визуализацию состояния игрового поля.

*GameLogic* – модуль реализует основную игровую логику для проекта «Морской бой». Он состоит из двух ключевых классов: *GameLogic* (главная игровая логика) и *ComputerAI* (искусственный интеллект противника). Эти классы управляют размещением кораблей, обработкой ходов, взаимодействием игрока с интерфейсом и реализацией алгоритмов для ИИ;

*GUI* – модуль предназначен для создания главного меню игры с визуальными эффектами и анимацией. Он обеспечивает интерактивный интерфейс с кнопками, заголовком и динамическим фоном, создавая атмосферу морского приключения.

*Main* – модуль является основным управляющим компонентом игры "Морской бой". Он управляет авторизацией, главным меню, сетевым подключением, процессом расстановки кораблей и игровым циклом, включая битву. управление игровым процессом, включая ходы ИИ, сетевое взаимодействие, проверку победы и отображение результатов. Он связывает игровые механики, сетевое соединение и визуальные элементы. Ключевые компоненты:

*Network* – модуль отвечает за сетевое взаимодействие в игре, обеспечивая соединение между игроками и передачу данных на сервер. Он реализует подключение к серверу, отправку игровых действий и получение состояния игры.

*Config* – модуль содержит набор серверных параметров, необходимых для настройки сетевой игры. Он определяет ключевые характеристики соединения, допустимые игровые условия и ограничения.

*Server* – модуль реализует серверную логику для сетевой игры "Морской бой", обрабатывая соединения между игроками, управление игровыми сессиями и обмен данными.

## **Проектирование интерфейса программного средства**

Интерфейс игрового проекта "Сетевой Морской бой" основан на *2D*-графике, где используются динамичные элементы для взаимодействия пользователей. Он создан с применением библиотеки *Pygame*, что позволяет реализовать плавные анимации, интерактивные кнопки и детализированную игровую среду.

Приложение работает в полноэкранном режиме и поддерживает анимированные визуальные эффекты, такие как движение волн, корабли, всплески воды при попадании снарядов. Визуальное оформление помогает создать атмосферу морского сражения, делая игру более захватывающей.

В интерфейсе предусмотрены несколько экранов:

1. Экран авторизации – содержит поля ввода логина и пароля, а также кнопки для входа и регистрации.
2. Главное меню – здесь игрок может выбрать режим игры: против ИИ или с реальным соперником по сети.
3. Экран расстановки кораблей – предлагает удобное размещение флота перед началом боя.
4. Игровое поле – показывает сетку с кораблями игроков, позволяет выполнять ходы и видеть результаты выстрелов.
5. Экран завершения игры – объявляет победителя и предлагает вернуться в главное меню.

### **Регистрация и авторизация**

После запуска программы игроки проходят этап регистрации или авторизации. Для этого предусмотрен отдельный экран, содержащий два поля для ввода логина и пароля, а также кнопки **"Войти"** и **"Регистрация"**.

При регистрации система проверяет логин на уникальность и допустимые символы, а пароль – на надежность и соответствие требованиям безопасности, также предусмотрен повтор пароля. Если введенные данные корректны, они хранятся в локальном файле *JSON* в зашифрованном виде.

Для авторизации игроки вводят свой логин и пароль, после чего их данные сверяются с хранимыми записями. В случае успешного входа пользователь переходит в главное меню. Если данные не совпадают, система выводит сообщение об ошибке.

Визуальное оформление экрана авторизации выполнено в морском стиле, с анимированным фоном и волнами. Поля ввода имеют *placeholders*, а кнопки реагируют на наведение, создавая приятную интерактивность. Вид окна регистрации и авторизации представлен на рисунках 2.1 и 2.2.

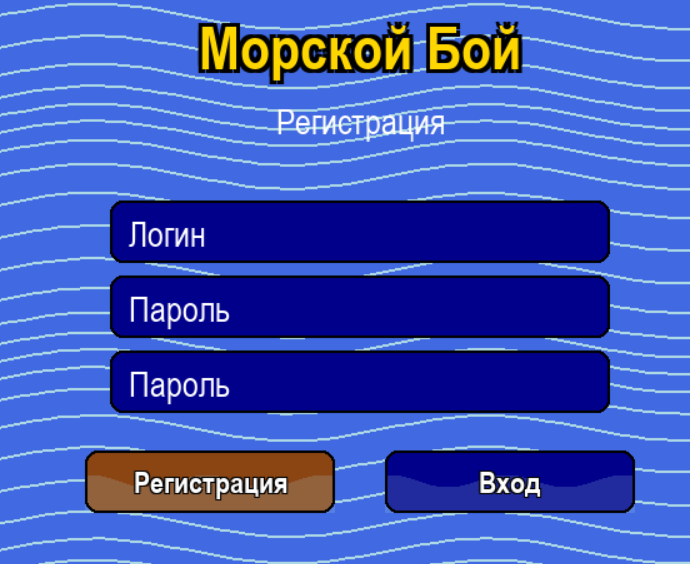


Рисунок 2.1 – Окно регистрации

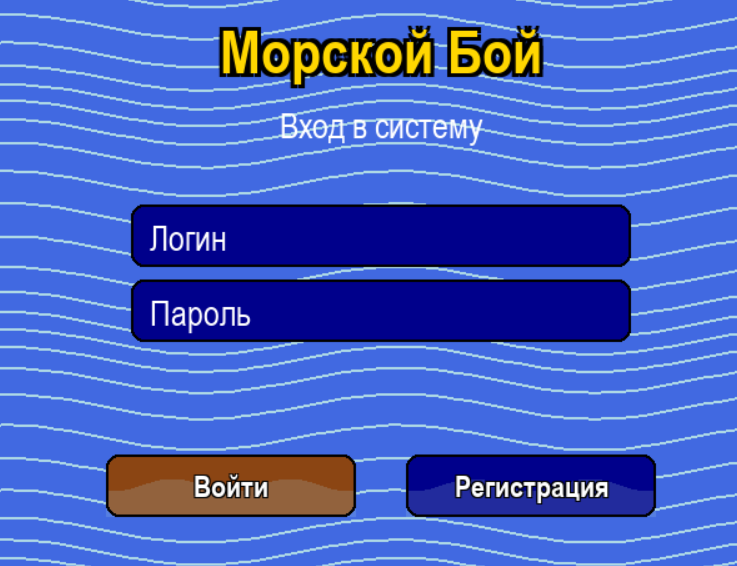


Рисунок 2.2 – Окно авторизации

### **Процесс игры**

После успешного входа в систему игрок попадает в главное меню, которое предоставляет три основных варианта продолжения:

1. Играть с ИИ – режим для одиночной игры, где пользователь соревнуется против компьютерного противника.
2. Сетевая игра – подключение к онлайн-матчу, где можно сразиться с другим игроком.
3. Настройки – раздел, где можно изменить параметры игры, такие как графика и управление.

Главное меню оформлено в морском стиле, с анимированным фоном и волнами, создающими эффект движения. Кнопки имеют интерактивную анимацию: они меняют цвет при наведении, а при нажатии запускают соответствующий игровой режим.

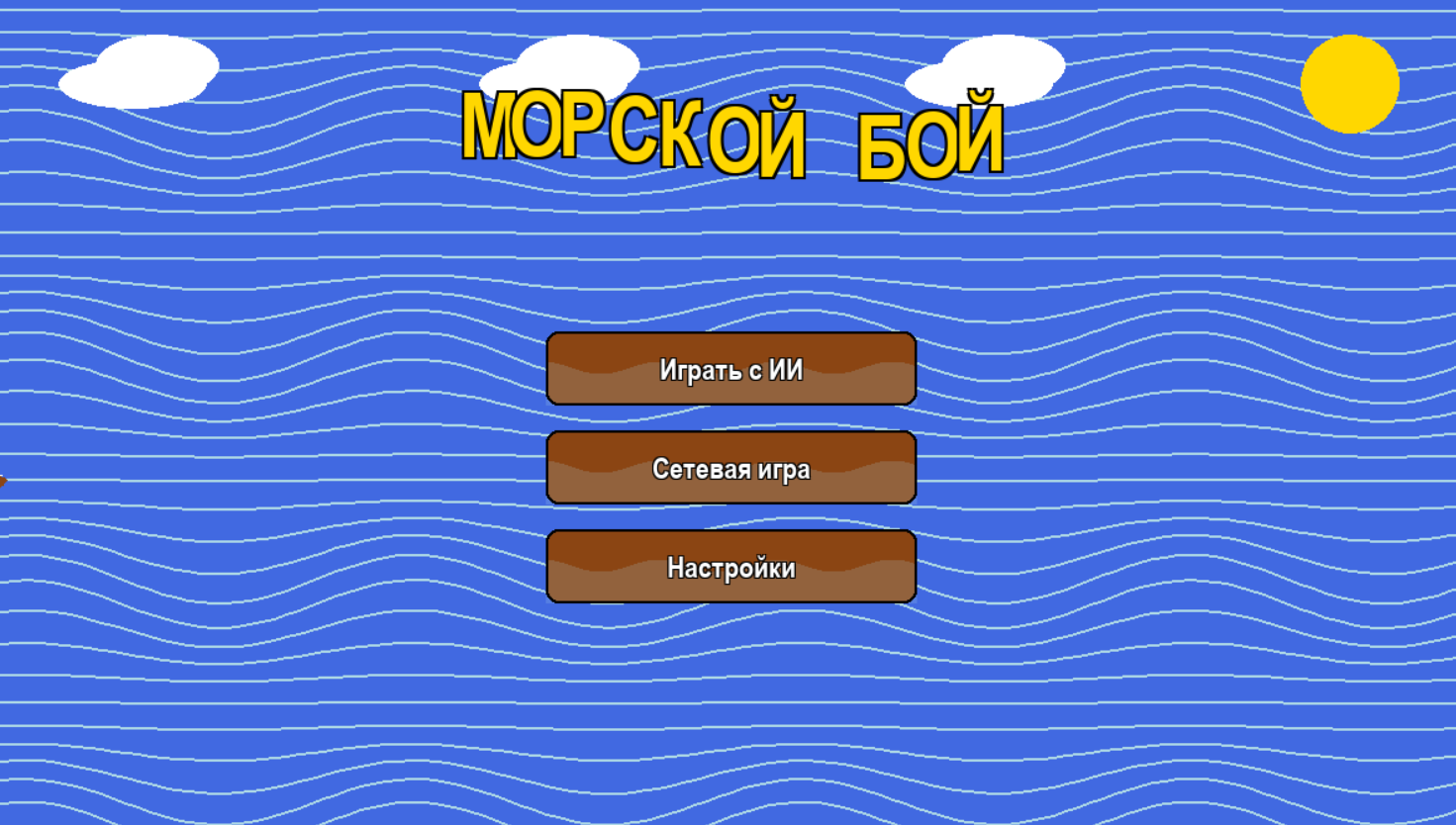


Рисунок 2.3 – Главное меню

Режим игры против искусственного интеллекта предоставляет возможность одиночного сражения в "Морском бою". В этом режиме пользователь играет против компьютерного противника, который использует алгоритмы для принятия решений по размещению кораблей и стрельбе.

После выбора режима "Играть с ИИ", игрок переходит на экран расстановки кораблей. Здесь он вручную размещает свой флот, выбирая размеры судов и их ориентацию на игровом поле. Интерфейс предусматривает удобное управление: корабли можно поворачивать, перемещать и проверять на возможность размещения.

Когда пользователь завершает расстановку флота, начинается игровой процесс. Искусственный интеллект делает ходы, анализируя предыдущие попадания и стратегически выбирая цели. Алгоритм ИИ работает на основе эвристик:

* При промахе — он выбирает новую случайную клетку.
* При попадании — пытается определить направление расположения корабля и продолжает атаку в этом направлении.
* При уничтожении корабля — помечает соседние клетки как недоступные для дальнейших выстрелов.

Режим "Сетевая игра" позволяет пользователю сыграть против реального соперника через интернет. Это добавляет элемент стратегии и динамичности, так как противник может действовать непредсказуемо.

После выбора сетевого режима запускается процесс подключения к серверу. Игрок автоматически соединяется с сервером, который распределяет его в игровую комнату и назначает уникальный *player\_id*. Если на сервере уже присутствует другой игрок, матч начинается сразу же. В противном случае игрок ожидает соперника.

Когда оба участника подключены, они переходят к этапу расстановки кораблей, аналогично одиночному режиму. После завершения расстановки начинается морской бой, в котором игроки по очереди атакуют друг друга.

При каждом выстреле:

1. Игрок выбирает координаты атаки.
2. Сообщение с координатами отправляется на сервер.
3. Сервер анализирует ситуацию и отвечает результатом (hit, miss, sunk).
4. В зависимости от ответа игрок видит анимацию попадания или промаха, а ход передаётся сопернику.

Игровая логика полностью синхронизирована между клиентами, а сервер следит за порядком ходов и честностью матчей. При победе одного из участников сервер отправляет финальное сообщение с результатом, а игроки могут вернуться в главное меню.

## **Проектирование функционала программного средства**

Для описания были отобраны 3 основные алгоритма:

* **Алгоритм обработки выстрела** – определяет, попал ли игрок в корабль противника, потоплен ли он и передает ход другому игроку.
* **Алгоритм размещения кораблей** – проверяет корректность расположения флота, предотвращая ошибки и недопустимые расстановки.
* **Алгоритм управления сетевой игрой** – координирует взаимодействие между клиентами, синхронизирует игровое состояние и обеспечивает честность матчей.

Одним из ключевых алгоритмов игрового процесса является **обработка выстрела**, которая определяет попадание, уничтожение корабля и смену хода между игроками.

При атаке игрок выбирает координаты выстрела и отправляет данные либо серверу (в сетевой игре), либо алгоритму ИИ (в одиночном режиме). После получения координат проверяется состояние клетки на игровом поле соперника:

* Если клетка **пустая**, выстрел считается **промахом**, и ход передаётся другому игроку.
* Если в клетке **находится часть корабля**, она помечается как **попадание**, но ход остаётся у атакующего.
* Если попадание затрагивает **последнюю целую часть корабля**, судно считается **уничтоженным**, и алгоритм отмечает его вокруг.

После попадания алгоритм анализирует соседние клетки и проверяет, остались ли у корабля неповреждённые части. Если все его сегменты были подбиты, корабль считается **потопленным**, а соседние клетки вокруг него блокируются для последующих атак, исключая возможность ложных выстрелов. После каждого выстрела система обновляет состояние игры, отправляя информацию игрокам. В сетевом режиме это реализовано через сервер, который фиксирует состояние полей, передаёт результаты и контролирует очередь ходов. Каждое попадание сопровождается анимацией всплеска воды, изменением цвета клетки, а при уничтожении – эффектами взрыва корабля. Это делает игровой процесс динамичным и зрелищным.

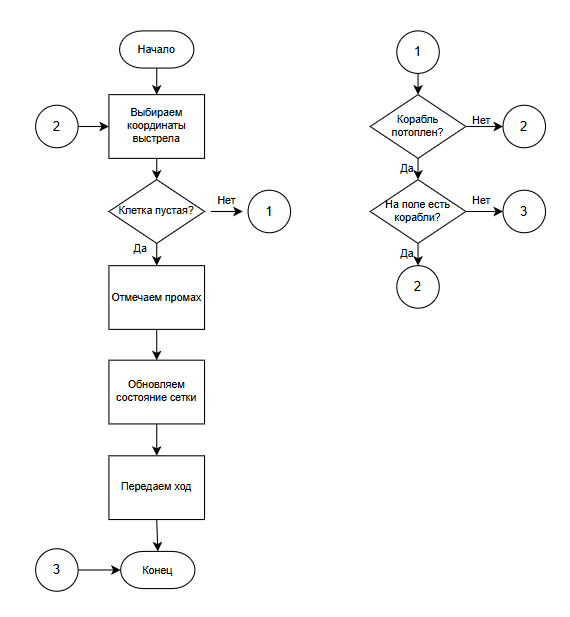


Рисунок 2.4 – Блок-схема **алгоритм обработки выстрела**

Перед началом морского боя игроки расставляют свои корабли на игровом поле. Этот процесс требует продуманного подхода, так как удачное размещение флота может существенно повлиять на исход битвы.

Игроку доступны корабли разного размера:

* **4-палубные (линкоры)** – самые крупные и редкие корабли.
* **3-палубные (крейсеры)** – универсальный вариант.
* **2-палубные (эсминцы)** – манёвренные, легко размещаются.
* **1-палубные (катера)** – небольшие, трудно обнаружить.

При размещении корабля система проверяет несколько условий:

* Корабль **не должен выходить за границы поля**.
* Корабли **не могут располагаться рядом** – между ними должен быть минимум одна клетка расстояния.
* **Ориентация судна** (горизонтальная или вертикальная) выбирается игроком.

При установке корабля алгоритм выполняет анализ доступных клеток. Он проверяет, есть ли рядом другие корабли, выходит ли судно за рамки сетки и соответствует ли оно игровым правилам. Если размещение невозможно, игрок получает визуальное предупреждение.

В режиме одиночной игры **искусственный интеллект** расставляет свои корабли случайным образом, соблюдая все правила. Чтобы сделать игру сложнее, его размещение не всегда идеально, а некоторые корабли могут находиться в неожиданных местах.

Игровой интерфейс предоставляет удобные элементы управления:

* Корабли можно **перемещать и вращать** перед размещением.
* При наведении **подсвечиваются допустимые позиции.**
* После окончательной расстановки флот игрока закрепляется на поле.

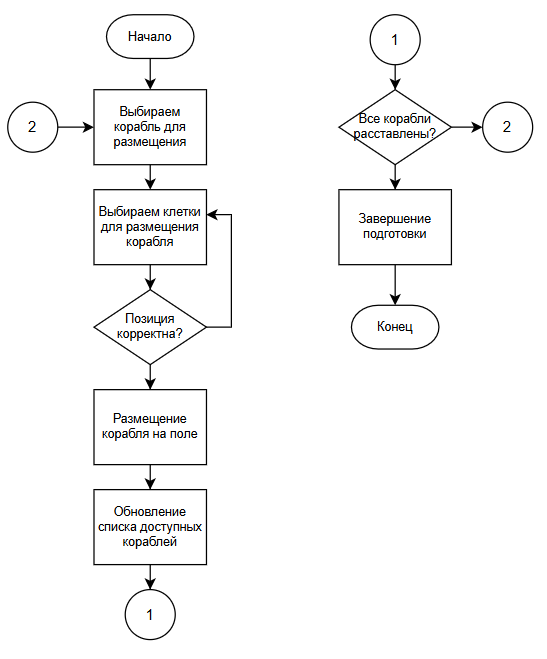


Рисунок 2.5 – Блок-схема **алгоритм размещения кораблей**

Сетевая игра требует чётко организованного обмена данными между игроками, а также координации ходов, сохранения состояния игры и обеспечения честности матчей.

При запуске сетевого режима игроки соединяются с сервером через ***TCP*-соединение**. Сервер фиксирует каждого подключившегося игрока, назначает **уникальный идентификатор (***player\_id***)**, а затем ожидает второго участника. Когда оба игрока в сети, система переходит к этапу расстановки кораблей.

После завершения подготовки сервер начинает управлять ходами:

* Отслеживает **очерёдность выстрелов**, не позволяя игрокам ходить одновременно.
* Получает данные о **координатах выстрела**, анализирует их и отправляет результат другому участнику.
* Обрабатывает **попадания**, уничтожение кораблей и передачу хода.

Каждый шаг синхронизируется через сервер, который следит за тем, чтобы игроки получали актуальную информацию о состоянии боя.

После каждого хода сервер анализирует состояние игрового поля и проверяет, остались ли у противника корабли. Если все корабли уничтожены, он отправляет сообщение о завершении матча и объявляет победителя.

Чтобы избежать ошибок и лагов, сервер работает в **многопоточной среде**, где каждый игрок обрабатывается в отдельном потоке. В случае разрыва соединения система уведомляет второго участника и завершает игру.

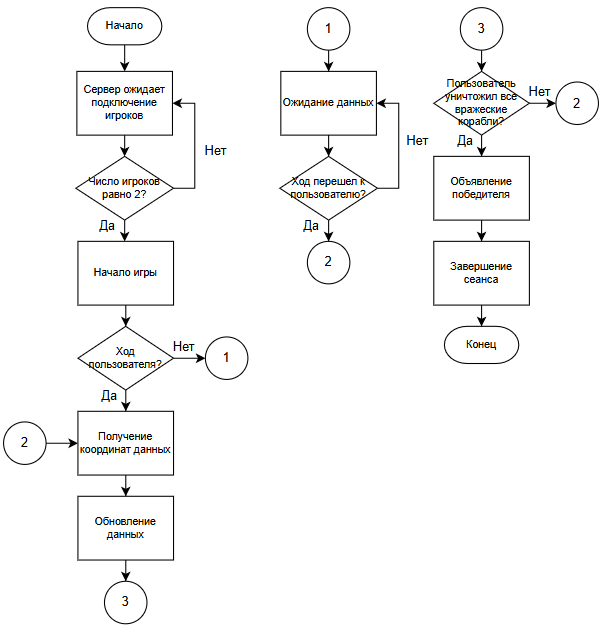


Рисунок 2.6 – Блок-схема **алгоритм управления сетевой игрой**

# **РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА**

## **Система аутентификации**

Перед началом игрового процесса пользователи проходят этап аутентификации, который включает регистрацию новых игроков и авторизацию уже существующих. Эти процессы обеспечивают безопасность данных и идентификацию игроков в игровой системе.

При регистрации пользователю необходимо ввести логин и пароль, которые затем проверяются на соответствие установленным требованиям. Валидация данных включает проверку длины логина, допустимых символов и его уникальности среди уже зарегистрированных пользователей. Пароли также проходят контроль на соответствие безопасности.

После успешной проверки данных пароль хешируется с использованием алгоритма *SHA*-256 и сохраняется в локальном файле *JSON* (*users.json*). В процессе аутентификации введенные пользователем логин и пароль сверяются с сохранёнными в файле. Если данные совпадают, игрок получает доступ к системе и переходит в главное меню.

Этот механизм авторизации реализуется с использованием файловой системы вместо базы данных, что делает программу автономной и упрощает ее использование без необходимости развертывания серверного окружения.

Для удобства взаимодействия с пользователем в программе предусмотрен графический интерфейс авторизации. Визуальные элементы, такие как поля ввода, кнопки и сообщения об ошибках, помогают игроку корректно пройти регистрацию или вход в систему. Анимация фона и интерактивные кнопки улучшают пользовательский опыт, делая процесс аутентификации удобным и интуитивно понятным.

## **Клиент-серверная модель**

Ключевым элементом сетевой игры является связь между клиентами и сервером, обеспечивающая обмен данными и координацию игровых действий.

Игроки подключаются к серверу через *TCP*-соединение, которое гарантирует надежную передачу данных. После успешного соединения сервер назначает каждому пользователю уникальный *player\_id*, который используется для управления очередностью ходов.

Перед началом игровой сессии клиенты передают на сервер данные о расположении своих кораблей, а сервер, в свою очередь, обрабатывает информацию и проверяет корректность размещения. В процессе игры клиенты отправляют серверу команды, такие как *shot*, обозначающие выстрел по вражескому полю. После получения запроса сервер анализирует состояние противника и возвращает результат – *hit*, *miss* или *sunk*, если корабль полностью уничтожен.

Для обработки сетевых событий сервер работает в многопоточной среде: на каждого подключенного игрока создается отдельный поток, который принимает и передает данные без задержек. Сервер также управляет игровым процессом, отслеживая состояние каждой партии, а при завершении игры оповещает всех участников о победителе.

Взаимодействие между клиентом и сервером реализовано с использованием сериализации данных с помощью *pickle*, что позволяет эффективно передавать сложные объекты, такие как игровая сетка и статусы игроков.

## **Создание игровой сессии**

После успешной авторизации игрок переходит в главное меню, где ему предлагается создать новую игровую комнату или подключиться к уже существующей.

При создании новой комнаты инициатор запускает локальный сервер с использованием *TCP*-соединения, которое привязывается к его текущему *IP*-адресу. Первый игрок, создавший комнату, подключается к серверу как клиент, но с другого порта, обеспечивая двустороннюю связь.

Другие пользователи могут подключаться к комнате, используя сетевой модуль *Network*, который устанавливает соединение с хостом. При успешном подключении сервер обновляет список активных клиентов и рассылает обновленные данные всем участникам. Это позволяет динамически обновлять элементы интерфейса, отображающие список игроков.

Для каждого клиента сервер создаёт отдельный поток, используя метод *handle\_client()*, который отвечает за обработку входящих сообщений. Этот поток принимает данные от игроков, анализирует их и пересылает соответствующему оппоненту. Клиенты, в свою очередь, ожидают информацию от сервера в методе *send()*, который получает игровые обновления, такие как начало матча, изменение статуса или выполнение игрового действия.

После того как необходимое число игроков подключилось, хост может начать игру, что переводит всех участников из режима подготовки в сам игровой процесс. В этот момент сервер фиксирует расположение кораблей каждого игрока и готовится к координации ходов.

## **Игровой процесс**

После подключения всех игроков к игровой комнате и завершения этапа подготовки, начинается сам игровой процесс.

Игровая логика управляет ходами, отслеживает состояние каждого игрока и регулирует взаимодействие между клиентами и сервером. Система ходов организована так, что игроки стреляют по полю соперника, ожидая результатов попадания.

Ходы обрабатываются через сетевое взаимодействие: когда игрок делает выстрел, его клиент отправляет сообщение серверу с координатами цели. Сервер анализирует состояние игровой сетки противника и возвращает результат (*hit*, *miss*, *sunk*). При попадании проверяется, не уничтожен ли корабль полностью. Если весь корабль затоплен, сервер автоматически помечает соседние клетки как недоступные для дальнейших атак.

Игра продолжается до тех пор, пока один из игроков не уничтожит все корабли соперника. По окончании сервер отправляет всем клиентам сообщение с объявлением победителя, а интерфейс обновляется, демонстрируя результат.

Для одиночной игры предусмотрен встроенный ИИ, который принимает решения по стрельбе, анализируя предыдущие попадания и выбирая стратегически выгодные цели. Алгоритм автоматически выбирает направление атак, стремясь к уничтожению кораблей противника.

Весь игровой процесс сопровождается визуальными эффектами: анимацией волн, выстрелами, эффектами попаданий и уничтожений. Это делает игру более захватывающей и динамичной.

# **ТЕСТИРОВАНИЕ**

Для обеспечения стабильной работы приложения «Сетевой Морской Бой» и предотвращения ситуаций, способных привести к нарушению логики игры или к её аварийному завершению, была реализована система обработки исключений и уведомлений.

Первым будет представлена на рисунке 4.1 обработка на корректность заданных данных, а именно логина и пароля.



Рисунок 4.1 – Обработка некорректно заданного логина и пароля

Если во время сетевой игры один из игроков внезапно отключается, система автоматически сообщает об этом остальным участникам. Если отключается игрок, не являющийся хостом, текущая сессия завершается с сохранением игровых данных, и игра запускается заново с оставшимися участниками. Сообщение об отключении игрока представлено на рисунке 4.2.



Рисунок 4.2 – Отключение игрока

Если же игрок, пытаются подключится в комнату, но сервер при этом не включен, система информирует об этом, выводя соответствующее уведомление. Такие уведомления представлены в виде программы и консоли на рисунках 4.3 и 4.4.

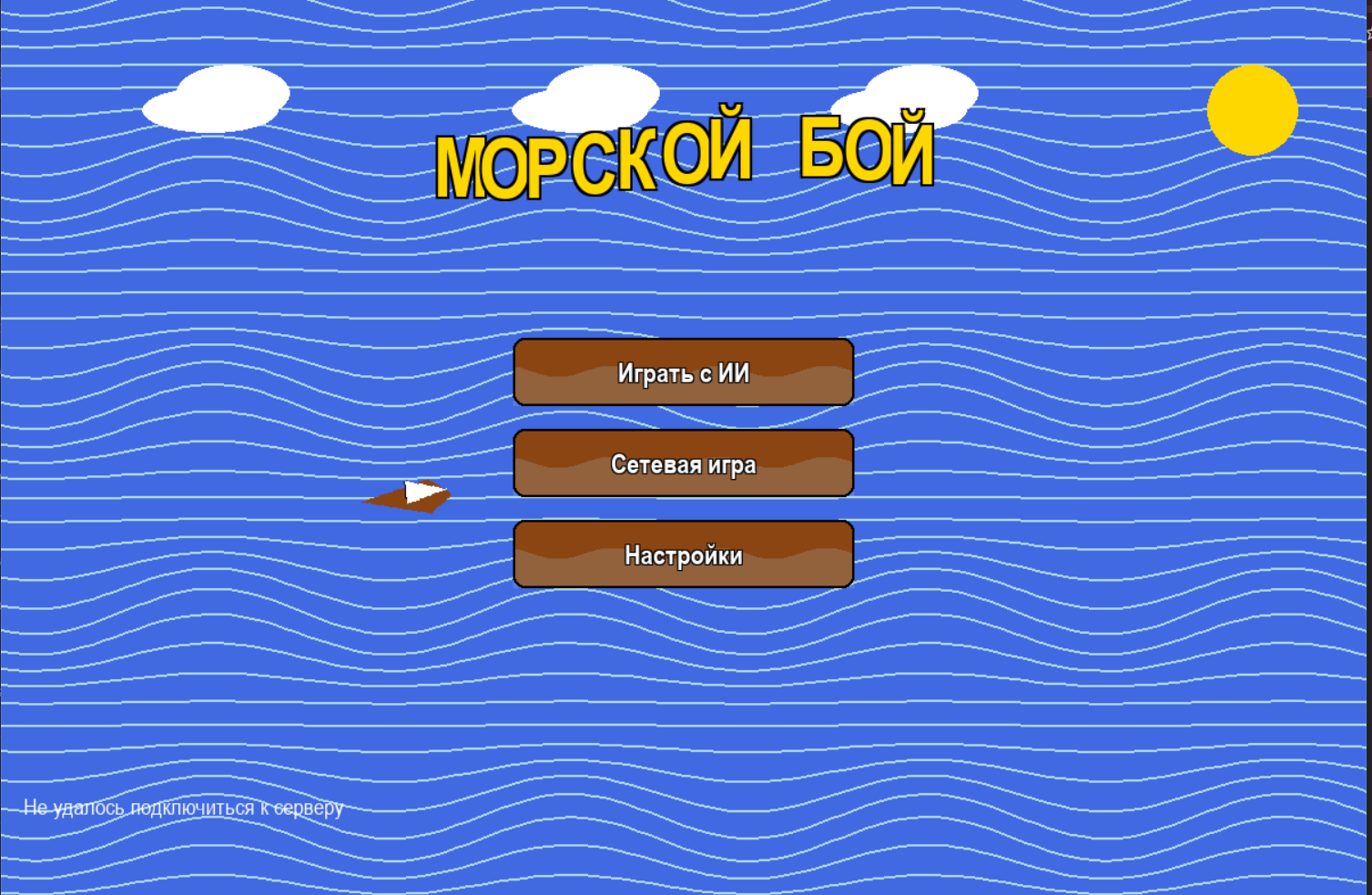


Рисунок 4.3 – Результат подключения пользователя к не включенному серверу в виде программы

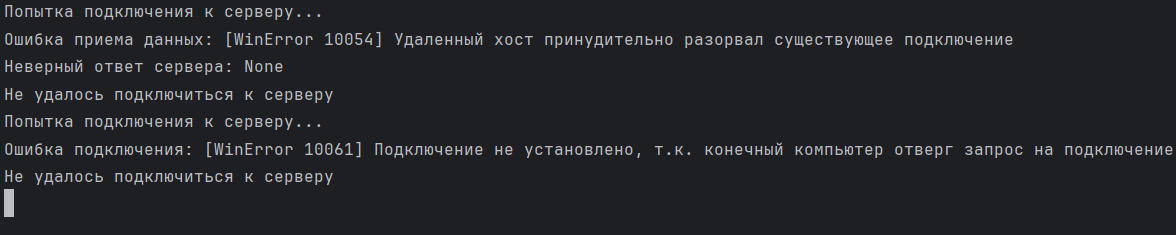


Рисунок 4.4 – Результат подключения пользователя к не включенному серверу в виде консоли

# **ПРИМЕНЕНИЕ ПРОГРАММЫ**

Данное руководство пользователя направлено на ознакомление с функционалом программы «Сетевой Морской Бой». Проект предусматривает два режима игры: против искусственного интеллекта и в сетевом режиме против другого игрока. Программа включает систему регистрации и авторизации с использованием *JSON*-файлов, после чего пользователь попадает в главное меню с выбором одного из режимов.

## **Руководство пользователя**

После запуска программы пользователь видит окно авторизации, где он может либо зарегистрироваться, либо войти в уже существующий аккаунт. При регистрации пользователь вводит корректные логин, пароль и повторный пароль в соответствующие поля и нажимает кнопку «Регистрация». При некорректном вводе появляется сообщение с указанием ошибки. Интерфейс окна регистрации был показан ранее.

Если у пользователя уже есть аккаунт, он может перейти в режим авторизации, нажав кнопку «Войти». После ввода данных и нажатия кнопки «Вход» происходит проверка введенной информации, и в случае успешной авторизации пользователь перенаправляется в главное меню. Образец интерфейса окна авторизации также показан ранее

Главное меню предоставляет две основные опции: «Играть с ИИ» и «Сетевая игра». Выбрав режим «Играть с ИИ», пользователь погружается в однопользовательскую игру, где процесс боя происходит против встроенного искусственного интеллекта. Все элементы управления, от расстановки кораблей до осуществления выстрелов, интуитивно понятны и сопровождаются визуальными эффектами, способствующими полному погружению в игровой процесс.

При выборе режима «Сетевая игра» программа инициирует подключение к серверу, уже запущенному на локальном хосте или в сети. В случае успешного подключения, пользователь переходит в режим ожидания, где программа информирует о статусе соединения и ожидает подключения второго игрока. Как только оба участника готовы, начинается сетевая игровая сессия, в ходе которой данные о состоянии игровых полей обновляются в режиме реального времени. Интерфейс “Сетевой игры” ничего не отличается от “Игры против ИИ”.

В процессе игры пользователям предоставляется возможность контролировать собственное игровое поле, делать выстрелы по вражеским кораблям и получать мгновенную обратную связь о попаданиях или промахах. Выделение активного игрока осуществляется посредством визуальных подсветок, а результаты каждого выстрела сопровождаются соответствующими сообщениями.

Для завершения игры и возврата в главное меню предусмотрены специальные кнопки, позволяющие в любой момент прекратить игровую сессию. Это обеспечивает удобный способ выхода из игры и дальнейшую навигацию по приложению.

# **ЗАКЛЮЧЕНИЕ**

В рамках выполнения курсовой работы была разработана сетевая версия игры «Морской бой», реализующая основные механики классического игрового процесса с возможностью выбора режима одиночной игры против искусственного интеллекта или многопользовательского матча. Основной целью разработки было создание стабильной и увлекательной игровой системы с простым и интуитивно понятным интерфейсом, позволяющим пользователям удобно управлять игровыми процессами.

Для реализации проекта использовались современные технологии, обеспечивающие стабильность и корректную работу сетевых механик. Взаимодействие между игроками организовано с применением TCP-соединений, обеспечивающих надёжную передачу игровых данных. Серверная архитектура построена на многопоточной обработке запросов, что позволяет динамически отслеживать состояние игры, корректировать статус участников и синхронизировать их взаимодействие в режиме реального времени. Графическая составляющая создана с использованием библиотеки Pygame, благодаря чему удалось реализовать анимацию волн, плавные переходы между игровыми экранами и эффекты стрельбы, попаданий и уничтожений кораблей. Авторизация и регистрация игроков выполнены с применением локального хранилища JSON, что делает систему удобной и автономной, не требующей подключения к внешним базам данных.

Разработанная версия игры предоставляет несколько основных режимов: одиночную игру против искусственного интеллекта, многопользовательский матч, а также настройку параметров игрового процесса. В режиме игры против ИИ используется алгоритм, анализирующий предыдущие попадания и промахи, формирующий стратегию атак и адаптирующий поведение компьютера в зависимости от действий пользователя. Многопользовательский режим организован с применением серверной логики, распределяющей очередность ходов между игроками и координирующей взаимодействие, позволяя пользователям соревноваться друг с другом в стратегическом морском сражении.

Разработанный программный продукт имеет ряд преимуществ, среди которых лёгкость и доступность интерфейса, минимальные системные требования и удобная навигация. В отличие от сложных 3D-игр, он не требует мощных аппаратных ресурсов, а также предоставляет игрокам анимированный и тематически оформленный интерфейс, создающий атмосферу реального морского боя. В игре предусмотрена динамическая смена статуса участников, визуальные эффекты, а также удобное управление игровыми процессами, позволяющее комфортно взаимодействовать с механиками игры.

Среди недостатков текущей версии можно отметить отсутствие соревновательного рейтингового режима и полноценной статистики игроков, что снижает уровень долгосрочного взаимодействия пользователей. Кроме того, программа пока не поддерживает облачные серверы, что ограничивает возможность игры за пределами локальной сети. Однако заложенные основы позволяют масштабировать проект в дальнейшем, добавляя новые функциональные возможности.

Проведённый анализ аналогичных игровых проектов позволил выделить сильные и слабые стороны различных реализаций сетевых игр, на основе чего был разработан баланс между производительностью, игровыми механиками и удобством интерфейса. На текущем этапе игра представляет собой полностью функционирующий прототип, демонстрирующий перспективы развития в коммерческом направлении. В дальнейшем возможны улучшения сетевой инфраструктуры, расширение игровых режимов и добавление новых механик, которые могут сделать проект ещё более интересным и востребованным среди пользователей.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Стандарт предприятия. Общие требования. СТП 01–2024 / Доманов А. Т., Сорока Н. И. – Минск : БГУИР, 2024. – 178 с.

[2] Python Software Foundation. Python 3 Documentation [Электронный ресурс]. – Режим доступа: https://docs.python.org/3, 2023.

[3] Pygame Community. Pygame Official Documentation [Электронный ресурс]. – Режим доступа: https://www.pygame.org/docs, 2023

[4] MySQL [Электронный ресурс]. – Режим доступа : https://www.mysql.com

[5] Эльмуродов, Ш. Ш. Принцип построения баз данных = Principle of building the database / Электронные системы и технологии : сборник материалов 59-й научной конференции аспирантов, магистрантов и студентов БГУИР, Минск, 17–21 апреля 2023 г. / Эльмуродов Ш. Ш. – Минск, 2023. – С. 371–373.

[6] Microsoft [Электронный ресурс]. – Режим доступа : https://visualstudio.microsoft.com/ru/downloads/

[7] Metanit, Python [Электронный ресурс]. – Режим доступа : https://metanit.com/python/tutorial/

[8] Грокаем алгоритмы: пособие / А. Бхаргава – Минск, 2022.

[9] LocalHost [Электронный ресурс]. – Режим доступа : http://localhost/MAMP/phpmyadmin.php?lang=en.

[10] Metanit, TCP-сервер [Электронный ресурс]. – Режим доступа : https://metanit.com/sharp/net/4.2.php

# **ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Листинг кода**

from auth import AuthScreen  
from gui import MainMenu, draw\_grid\_labels  
from game\_objects import Button, GameGrid  
from gamelogic import GameLogic  
from constants import \*  
from network import Network  
  
  
def main():  
 pygame.init()  
 screen = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT))  
 pygame.display.set\_caption("Морской Бой")  
 clock = pygame.time.Clock()  
  
 auth\_screen = AuthScreen(screen)  
 if not auth\_screen.run():  
 pygame.quit()  
 return  
  
 menu = MainMenu()  
 game\_logic = None  
 network = None  
 current\_screen = "menu"  
 game\_winner = None  
 player\_id = None  
 ready\_btn = None  
 cancel\_btn = None  
 back\_to\_menu\_btn = None  
 waiting\_time = 0  
  
 running = True  
 while running:  
 mouse\_pos = pygame.mouse.get\_pos()  
 screen.fill(OCEAN\_BLUE)  
  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 running = False  
 if network:  
 network.disconnect()  
 pygame.quit()  
 return  
  
 if current\_screen == "menu":  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 for i, btn in enumerate(menu.buttons):  
 if btn.rect.collidepoint(mouse\_pos):  
 if i == 0: # Игра с ИИ  
 game\_logic = GameLogic(screen)  
 current\_screen = "placement"  
 print("Начата игра с ИИ")  
 elif i == 1:  
 try:  
 game\_logic = GameLogic(screen)  
 network = Network()  
 print("Попытка подключения к серверу...")  
  
 if network.connect():  
 print("Подключение успешно")  
 # Получаем данные от сервера  
 if hasattr(network, 'player\_id') and network.player\_id is not None:  
 player\_id = network.player\_id  
 game\_logic.player\_id = player\_id  
  
 if player\_id % 2 == 0: # Первый игрок в паре  
 current\_screen = "waiting"  
 game\_logic.add\_message("Ожидание второго игрока...")  
 waiting\_time = time.time()  
 cancel\_btn = Button("Отмена", SCREEN\_WIDTH // 2 - 100,  
 SCREEN\_HEIGHT - 100, 200, 50)  
 print(f"Игрок {player\_id}: Ожидание второго игрока")  
 else: # Второй игрок в паре  
 current\_screen = "placement"  
 game\_logic.add\_message("Соперник найден! Расставьте корабли.")  
 ready\_btn = Button("Готов", SCREEN\_WIDTH // 2 - 100,  
 SCREEN\_HEIGHT - 100, 200, 50)  
 print(f"Игрок {player\_id}: Начало расстановки кораблей")  
 else:  
 game\_logic.add\_message("Ошибка: не получен ID игрока")  
 print("Ошибка: не получен ID игрока")  
 else:  
 game\_logic.add\_message("Не удалось подключиться к серверу")  
 print("Не удалось подключиться к серверу")  
 except Exception as e:  
 print(f"Ошибка подключения: {str(e)}")  
 if game\_logic:  
 game\_logic.add\_message(f"Ошибка: {str(e)}")  
  
 elif current\_screen == "waiting":  
 if event.type == pygame.MOUSEBUTTONDOWN and cancel\_btn and cancel\_btn.rect.collidepoint(mouse\_pos):  
 if network:  
 network.disconnect()  
 current\_screen = "menu"  
 menu = MainMenu()  
 print("Отмена ожидания")  
  
 elif current\_screen == "placement":  
 if event.type == pygame.KEYDOWN and event.key == pygame.K\_r:  
 if game\_logic.selected\_ship\_size:  
 game\_logic.ship\_orientation = 1 - game\_logic.ship\_orientation  
 print("Корабль повернут")  
  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 selected = game\_logic.ship\_selector.handle\_event(  
 mouse\_pos,  
 game\_logic.get\_available\_ships\_list()  
 )  
 if selected:  
 game\_logic.selected\_ship\_size = selected  
 print(f"Выбран корабль размером {selected}")  
  
 elif (left\_margin <= mouse\_pos[0] <= left\_margin + 10 \* block\_size and  
 upper\_margin <= mouse\_pos[1] <= upper\_margin + 10 \* block\_size and  
 game\_logic.selected\_ship\_size):  
  
 grid\_x = (mouse\_pos[0] - left\_margin) // block\_size  
 grid\_y = (mouse\_pos[1] - upper\_margin) // block\_size  
  
 if game\_logic.validate\_placement(grid\_x, grid\_y, game\_logic.selected\_ship\_size):  
 game\_logic.place\_ship(grid\_x, grid\_y, game\_logic.selected\_ship\_size)  
 print(f"Корабль размещен на ({grid\_x}, {grid\_y})")  
  
 elif ready\_btn and ready\_btn.rect.collidepoint(mouse\_pos):  
 if all(count == 0 for count in game\_logic.ships\_available):  
 try:  
 response = network.send({  
 'type': 'ready',  
 'player\_id': player\_id,  
 'grid': game\_logic.player\_grid  
 })  
 if response and response.get('status') == 'waiting':  
 game\_logic.add\_message("Ожидаем готовности соперника...")  
 print("Ожидание готовности соперника")  
 elif response and response.get('status') == 'battle':  
 current\_screen = "battle"  
 game\_logic.add\_message("Игра начинается!")  
 game\_logic.current\_turn = (player\_id % 2 == 0)  
 print("Начало игры")  
 except Exception as e:  
 game\_logic.add\_message(f"Ошибка: {str(e)}")  
 print(f"Ошибка отправки готовности: {str(e)}")  
  
 elif current\_screen == "battle":  
 if event.type == pygame.MOUSEBUTTONDOWN and game\_logic.current\_turn:  
 if (left\_margin + 15 \* block\_size <= mouse\_pos[0] <= left\_margin + 25 \* block\_size and  
 upper\_margin <= mouse\_pos[1] <= upper\_margin + 10 \* block\_size):  
  
 grid\_x = (mouse\_pos[0] - (left\_margin + 15 \* block\_size)) // block\_size  
 grid\_y = (mouse\_pos[1] - upper\_margin) // block\_size  
  
 if game\_logic.ai\_grid[grid\_y][grid\_x] in [2, 3]:  
 game\_logic.add\_message("Сюда уже стреляли!")  
 continue  
  
 if network:  
 try:  
 print(f"Отправка выстрела на ({grid\_x}, {grid\_y})")  
 response = network.send({  
 'type': 'shot',  
 'x': grid\_x,  
 'y': grid\_y,  
 'player\_id': player\_id  
 })  
  
 if response:  
 if response.get('hit'):  
 game\_logic.ai\_grid[grid\_y][grid\_x] = 2  
 if response.get('sunk'):  
 game\_logic.add\_message("Вы потопили корабль!")  
 else:  
 game\_logic.add\_message("Вы попали!")  
 else:  
 game\_logic.ai\_grid[grid\_y][grid\_x] = 3  
 game\_logic.add\_message("Вы промахнулись!")  
 game\_logic.current\_turn = False  
  
 if response.get('game\_over'):  
 current\_screen = "game\_over"  
 game\_winner = "player" if response.get('winner') == player\_id else "enemy"  
 except Exception as e:  
 game\_logic.add\_message(f"Ошибка: {str(e)}")  
 print(f"Ошибка выстрела: {str(e)}")  
  
 else:  
 result = game\_logic.handle\_shot(grid\_x, grid\_y, True)  
  
 if result == "miss":  
 game\_logic.current\_turn = False  
 game\_logic.add\_message("Ваш ход завершен. Ход ИИ...")  
  
 x, y = game\_logic.ai.make\_move()  
 ai\_result = game\_logic.handle\_shot(x, y, False)  
  
 if ai\_result == "hit":  
 game\_logic.add\_message("ИИ попал в ваш корабль!")  
 elif ai\_result == "sunk":  
 game\_logic.add\_message("ИИ потопил ваш корабль!")  
 elif ai\_result == "miss":  
 game\_logic.add\_message("ИИ промахнулся!")  
  
 game\_logic.current\_turn = True  
  
 winner = game\_logic.check\_victory()  
 if winner:  
 current\_screen = "game\_over"  
 game\_winner = winner  
 game\_logic.add\_message("Вы победили!" if winner == "player" else "ИИ победил!")  
  
 elif current\_screen == "game\_over":  
 if event.type == pygame.MOUSEBUTTONDOWN and back\_to\_menu\_btn and back\_to\_menu\_btn.rect.collidepoint(  
 mouse\_pos):  
 if network:  
 network.disconnect()  
 network = None  
 current\_screen = "menu"  
 menu = MainMenu()  
 print("Возврат в главное меню")  
  
 if network and network.connected:  
 message = network.check\_data()  
 if message:  
 print(f"Получено сетевое сообщение: {message}")  
  
 if message.get('status') == 'start\_placement' and current\_screen == "waiting":  
 current\_screen = "placement"  
 game\_logic.add\_message("Оба игрока подключены! Расставьте корабли.")  
 ready\_btn = Button("Готов", SCREEN\_WIDTH // 2 - 100, SCREEN\_HEIGHT - 100, 200, 50)  
 print("Начало расстановки кораблей")  
  
 elif message.get('status') == 'battle' and current\_screen == "placement":  
 current\_screen = "battle"  
 game\_logic.add\_message("Игра начинается!")  
 game\_logic.current\_turn = (player\_id % 2 == 0)   
 print("Начало игры")  
  
 elif message.get('type') == 'shot\_result' and current\_screen == "battle":  
 x = message.get('x')  
 y = message.get('y')  
 opponent\_id = message.get('player\_id')  
  
 if opponent\_id != player\_id:   
 result = game\_logic.handle\_shot(x, y, False)  
  
 if result == "hit":  
 game\_logic.add\_message("Противник попал в ваш корабль!")  
 elif result == "sunk":  
 game\_logic.add\_message("Противник потопил ваш корабль!")  
 elif result == "miss":  
 game\_logic.add\_message("Противник промахнулся!")  
 game\_logic.current\_turn = True  
  
 winner = game\_logic.check\_victory()  
 if winner:  
 current\_screen = "game\_over"  
 game\_winner = "player" if winner == "player" else "enemy"  
  
 if current\_screen == "waiting" and (time.time() - waiting\_time > 30):  
 game\_logic.add\_message("Таймаут подключения")  
 print("Таймаут подключения")  
 if network:  
 network.disconnect()  
 current\_screen = "menu"  
  
 if current\_screen == "menu":  
 menu.update(mouse\_pos)  
 menu.draw(screen)  
 if game\_logic and game\_logic.messages:  
 y\_offset = SCREEN\_HEIGHT - 100  
 for msg in game\_logic.messages[-3:]:  
 text\_surf = pygame.font.SysFont('Arial', 20).render(msg, True, WHITE)  
 screen.blit(text\_surf, (20, y\_offset))  
 y\_offset -= 30  
  
 elif current\_screen == "waiting":  
 menu.background = menu.create\_background()  
 screen.blit(menu.background, (0, 0))  
  
 font = pygame.font.SysFont('Arial', 36)  
 text = font.render("Ожидание второго игрока...", True, WHITE)  
 screen.blit(text, (SCREEN\_WIDTH // 2 - text.get\_width() // 2, SCREEN\_HEIGHT // 2 - 50))  
  
 if cancel\_btn:  
 cancel\_btn.draw(screen)  
  
 if game\_logic and game\_logic.messages:  
 y\_offset = SCREEN\_HEIGHT - 100  
 for msg in game\_logic.messages[-3:]:  
 text\_surf = pygame.font.SysFont('Arial', 20).render(msg, True, WHITE)  
 screen.blit(text\_surf, (20, y\_offset))  
 y\_offset -= 30  
  
 elif current\_screen == "placement":  
 GameGrid(left\_margin, upper\_margin, True).draw(screen, game\_logic.player\_grid, game\_logic)  
 GameGrid(left\_margin + 15 \* block\_size, upper\_margin, False).draw(screen, [[0] \* 10 for \_ in range(10)],  
 None)  
  
 game\_logic.ship\_selector.draw(screen, game\_logic.get\_available\_ships\_list())  
  
 draw\_grid\_labels(screen)  
  
 if ready\_btn:  
 ready\_btn.draw(screen)  
  
 game\_logic.draw\_messages(screen)  
  
 elif current\_screen == "battle":  
 GameGrid(left\_margin, upper\_margin, True).draw(screen, game\_logic.player\_grid, game\_logic)  
 GameGrid(left\_margin + 15 \* block\_size, upper\_margin, False).draw(screen, game\_logic.ai\_grid, game\_logic)  
  
 draw\_grid\_labels(screen)  
  
 status\_text = "Ваш ход" if game\_logic.current\_turn else "Ход соперника"  
 status\_color = GREEN if game\_logic.current\_turn else RED  
 status\_surf = pygame.font.SysFont('Arial', 24).render(status\_text, True, status\_color)  
 pygame.draw.rect(screen, (\*DARK\_BLUE, 200), (CHAT\_X, CHAT\_Y - 40, CHAT\_WIDTH, 30))  
 screen.blit(status\_surf, (CHAT\_X + 10, CHAT\_Y - 35))  
  
 game\_logic.draw\_messages(screen)  
  
 elif current\_screen == "game\_over":  
 menu.background = menu.create\_background()  
 screen.blit(menu.background, (0, 0))  
  
 font = pygame.font.SysFont('Arial', 48)  
 text = font.render("Вы победили!" if game\_winner == "player" else "Вы проиграли!",  
 True, GREEN if game\_winner == "player" else RED)  
 screen.blit(text, (SCREEN\_WIDTH // 2 - text.get\_width() // 2, SCREEN\_HEIGHT // 2 - 50))  
  
 back\_to\_menu\_btn = Button("В меню", SCREEN\_WIDTH // 2 - 100, SCREEN\_HEIGHT // 2 + 50, 200, 50)  
 back\_to\_menu\_btn.draw(screen)  
  
 pygame.display.flip()  
 clock.tick(60)  
  
 if network:  
 network.disconnect()  
 pygame.quit()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

import random  
from game\_objects import GameGrid  
from ship\_selector import ShipSelector  
from constants import \*  
from network import Network  
  
  
class GameLogic:  
 def \_\_init\_\_(self, screen):  
 self.screen = screen  
 self.player\_grid = [[0] \* 10 for \_ in range(10)]  
 self.ships = [4, 3, 3, 2, 2, 2, 1, 1, 1, 1]  
 self.available\_ships = self.ships.copy()  
 self.placement\_mode = True  
 self.current\_turn = True  
 self.game\_over = False  
 self.ship\_selector = ShipSelector(SCREEN\_WIDTH - 250, 100)  
 self.ai\_ships = self.place\_ai\_ships()  
 self.ai\_grid = [[0] \* 10 for \_ in range(10)]  
 self.messages = []  
 self.message\_font = pygame.font.SysFont('Arial', 24)  
 self.message\_rect = pygame.Rect(SCREEN\_WIDTH - 300, 20, 280, 200)  
 self.status\_font = pygame.font.SysFont('Arial', 28, bold=True)  
 self.status\_rect = pygame.Rect(SCREEN\_WIDTH - 300, 250, 280, 150)  
 self.rotation\_angle = 0  
 self.max\_rotation\_angle = 90  
 self.ships\_available = [1, 2, 3, 4]  
 self.ship\_types = [4, 3, 2, 1]  
 self.selected\_ship\_size = None  
 self.ship\_orientation = 0  
 self.network = None  
 self.player\_id = None  
  
 def update(self):  
 if self.rotation\_angle > 0:  
 self.rotation\_angle = max(0, self.rotation\_angle - 10)  
  
 def add\_message(self, text):  
 if not hasattr(self, 'messages'):  
 self.messages = []  
 self.messages.append(text)  
 if len(self.messages) > 5:  
 self.messages.pop(0)  
  
 def get\_available\_ships\_list(self):  
 """Возвращает список доступных размеров кораблей"""  
 available = []  
 for i, count in enumerate(self.ships\_available):  
 if count > 0:  
 available.append(self.ship\_types[i])  
 return available  
  
 def draw\_messages(self, surface):  
 pygame.draw.rect(surface, (\*DARK\_BLUE, 200), self.message\_rect)  
 pygame.draw.rect(surface, BLACK, self.message\_rect, 2)  
  
 for i, msg in enumerate(self.messages[-5:]):  
 text = self.message\_font.render(msg, True, WHITE)  
 surface.blit(text, (self.message\_rect.x + 10,  
 self.message\_rect.y + 10 + i \* 30))  
  
 def draw\_status\_panel(self, surface):  
 pygame.draw.rect(surface, (\*DARK\_BLUE, 200), self.status\_rect)  
 pygame.draw.rect(surface, BLACK, self.status\_rect, 2)  
  
 status = "Ваш ход" if self.current\_turn else "Ход ИИ"  
 text = self.status\_font.render(status, True,  
 GREEN if self.current\_turn else RED)  
 surface.blit(text, (self.status\_rect.x + 20, self.status\_rect.y + 20))  
  
 def validate\_ai\_placement(self, x, y, size, orientation, grid):  
 # Проверяем границы  
 if orientation == 0 and x + size > 10:  
 return False  
 if orientation == 1 and y + size > 10:  
 return False  
  
 # Проверяем саму клетку и окружение  
 for i in range(-1, size + 1):  
 for j in range(-1, 2):  
 check\_x = x + (i if orientation == 0 else j)  
 check\_y = y + (j if orientation == 0 else i)  
  
 if 0 <= check\_x < 10 and 0 <= check\_y < 10:  
 if grid[check\_y][check\_x] != 0:  
 return False  
 return True  
  
 def init\_battle(self):  
 self.player\_grid = GameGrid(left\_margin, upper\_margin, is\_player=True)  
 self.ai\_grid = GameGrid(left\_margin + 15 \* block\_size, upper\_margin, is\_player=False)  
 self.battle\_initialized = True  
  
 def validate\_placement(self, x, y, size):  
 # Проверка границ  
 if self.ship\_orientation == 0 and x + size > 10:  
 return False  
 if self.ship\_orientation == 1 and y + size > 10:  
 return False  
  
 # Проверка соседних клеток  
 for i in range(-1, size + 1):  
 for j in range(-1, 2):  
 check\_x = x + (i if self.ship\_orientation == 0 else j)  
 check\_y = y + (j if self.ship\_orientation == 0 else i)  
  
 if 0 <= check\_x < 10 and 0 <= check\_y < 10:  
 if self.player\_grid[check\_y][check\_x] != 0:  
 return False  
 return True  
  
 def place\_ship(self, x, y, size):  
 for i in range(size):  
 dx = i if self.ship\_orientation == 0 else 0  
 dy = i if self.ship\_orientation == 1 else 0  
 self.player\_grid[y + dy][x + dx] = 1  
  
 # Уменьшаем количество доступных кораблей этого типа  
 ship\_index = [4, 3, 2, 1].index(size)  
 self.ships\_available[ship\_index] -= 1  
  
 if self.ships\_available[ship\_index] <= 0:  
 self.selected\_ship\_size = None  
  
 def handle\_shot(self, x, y, is\_player=True):  
 target\_grid = self.ai\_grid if is\_player else self.player\_grid  
  
 if target\_grid[y][x] in [2, 3]: # Уже стреляли сюда  
 return None  
  
 if target\_grid[y][x] == 1: # Попадание  
 target\_grid[y][x] = 2  
 sunk, ship\_cells = self.check\_ship\_sunk(x, y, target\_grid)  
  
 if sunk:  
 self.last\_sunk\_size = len(ship\_cells)  
 self.mark\_around\_ship(ship\_cells, target\_grid)  
 return "sunk"  
 return "hit"  
 else: # Промах  
 target\_grid[y][x] = 3  
 return "miss"  
  
 def mark\_around\_ship(self, ship\_cells, grid):  
 directions = [(-1, -1), (-1, 0), (-1, 1),  
 (0, -1), (0, 1),  
 (1, -1), (1, 0), (1, 1)]  
  
 for x, y in ship\_cells:  
 for dx, dy in directions:  
 nx, ny = x + dx, y + dy  
 if 0 <= nx < 10 and 0 <= ny < 10 and grid[ny][nx] == 0:  
 grid[ny][nx] = 3  
  
 def check\_ship\_sunk(self, x, y, grid):  
 ship\_cells = []  
 directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]  
  
 to\_check = [(x, y)]  
 while to\_check:  
 cx, cy = to\_check.pop()  
 if (cx, cy) not in ship\_cells and grid[cy][cx] == 2:  
 ship\_cells.append((cx, cy))  
 for dx, dy in directions:  
 nx, ny = cx + dx, cy + dy  
 if 0 <= nx < 10 and 0 <= ny < 10:  
 to\_check.append((nx, ny))  
  
 for cx, cy in ship\_cells:  
 if grid[cy][cx] != 2:  
 return False, []  
  
 return True, ship\_cells  
  
  
 def check\_victory(self):  
 player\_alive = any(1 in row for row in self.player\_grid)  
 ai\_alive = any(1 in row for row in self.ai\_grid)  
  
 if not ai\_alive:  
 return "player"  
 elif not player\_alive:  
 return "ai"  
 return None  
  
 def start\_battle(self):  
 self.current\_turn = True  
 self.ai\_grid = self.ai\_ships  
  
 def place\_ai\_ships(self):  
 grid = [[0] \* 10 for \_ in range(10)]  
 for size in self.ships:  
 placed = False  
 while not placed:  
 x = random.randint(0, 9)  
 y = random.randint(0, 9)  
 orient = random.randint(0, 1)  
 if self.validate\_ai\_placement(x, y, size, orient, grid):  
 for i in range(size):  
 dx = i if orient == 0 else 0  
 dy = i if orient == 1 else 0  
 grid[y + dy][x + dx] = 1  
 placed = True  
 return grid  
  
 def handle\_events(self, events, mouse\_pos):  
 for event in events:  
 if event.type == pygame.QUIT:  
 return False  
  
 if event.type == pygame.KEYDOWN and event.key == pygame.K\_r:  
 if self.selected\_ship\_size is not None:  
 self.ship\_orientation = 1 - self.ship\_orientation  
  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 selected = self.ship\_selector.handle\_event(mouse\_pos, self.available\_ships)  
 if selected is not None:  
 self.selected\_ship\_size = selected  
  
 if (self.selected\_ship\_size is not None and  
 left\_margin <= mouse\_pos[0] <= left\_margin + 10 \* block\_size and  
 upper\_margin <= mouse\_pos[1] <= upper\_margin + 10 \* block\_size):  
  
 grid\_x = (mouse\_pos[0] - left\_margin) // block\_size  
 grid\_y = (mouse\_pos[1] - upper\_margin) // block\_size  
  
 if self.validate\_placement(grid\_x, grid\_y, self.selected\_ship\_size):  
 # Находим индекс только когда точно знаем, что selected\_ship\_size не None  
 ship\_index = self.ship\_types.index(self.selected\_ship\_size)  
 self.place\_ship(grid\_x, grid\_y, self.selected\_ship\_size)  
 self.ships\_available[ship\_index] -= 1  
  
 if self.ships\_available[ship\_index] <= 0:  
 self.selected\_ship\_size = None  
  
 if all(count == 0 for count in self.ships\_available):  
 self.start\_battle()  
 return "battle" # Возвращаем новое состояние  
 return "placement"  
  
  
 def handle\_ship\_placement(self, grid\_x, grid\_y):  
 if not self.selected\_ship\_size:  
 return False  
  
 if self.validate\_placement(grid\_x, grid\_y, self.selected\_ship\_size):  
 self.place\_ship(grid\_x, grid\_y, self.selected\_ship\_size)  
  
 # Уменьшаем количество доступных кораблей этого типа  
 ship\_index = [4, 3, 2, 1].index(self.selected\_ship\_size)  
 self.available\_ships[ship\_index] -= 1  
  
 # Если корабли этого типа закончились - сбрасываем выбор  
 if self.available\_ships[ship\_index] <= 0:  
 self.selected\_ship\_size = None  
  
 return True  
 return False  
  
 def process\_network\_response(self, response, x, y):  
 if response['hit']:  
 self.ai\_grid[y][x] = 2  
 if response.get('sunk'):  
 self.add\_message("Вы потопили корабль!")  
 else:  
 self.add\_message("Вы попали!")  
 else:  
 self.ai\_grid[y][x] = 3  
 self.add\_message("Вы промахнулись!")  
  
 self.current\_turn = (response['turn'] == self.player\_id)  
  
 if response.get('game\_over'):  
 return "player" if response['winner'] == self.player\_id else "enemy"  
 return None  
  
 def handle\_network\_shot(self, x, y):  
 try:  
 response = self.network.send({  
 'type': 'shot',  
 'coords': (x, y)  
 })  
  
 if response and response.get('type') == 'shot\_result':  
 if response['hit']:  
 self.ai\_grid[response['y']][response['x']] = 2  
 if response.get('sunk'):  
 self.mark\_around\_ship(response['x'], response['y'], self.ai\_grid)  
 self.add\_message("Вы попали!" if response['hit'] else "Вы промахнулись!")  
  
 if response.get('game\_over'):  
 return "player" if response['winner'] == self.player\_id else "enemy"  
  
 self.current\_turn = not response['hit'] or response.get('sunk', False)  
 return None  
  
 except Exception as e:  
 print(f"Ошибка при обработке выстрела: {e}")  
 self.add\_message("Ошибка соединения")  
 return "disconnected"  
  
  
 def handle\_network(self):  
 if not self.network or not self.network.connected:  
 return False  
  
 try:  
 # Проверяем состояние игры  
 state = self.network.send({"type": "get\_state"})  
 if state and 'status' in state:  
 if state['status'] == 'battle' and not self.started:  
 self.started = True  
 self.add\_message("Игра начинается!")  
 return 'start'  
 elif state['status'] == 'disconnected':  
 self.add\_message("Соперник отключился")  
 return 'disconnect'  
 return True  
 except:  
 return False  
class ComputerAI:  
 def \_\_init\_\_(self, grid):  
 self.grid = grid  
 self.possible\_targets = [(x, y) for x in range(10) for y in range(10)]  
 self.last\_hit = None  
 self.directions = [(0,1), (1,0), (0,-1), (-1,0)]  
 self.target\_mode = False  
  
 def make\_move(self):  
 if not self.possible\_targets:  
 return None  
  
 # Если было попадание, ищем вокруг  
 if self.last\_hit and not self.target\_mode:  
 self.target\_mode = True  
 self.directions = [(0,1), (1,0), (0,-1), (-1,0)]  
 random.shuffle(self.directions)  
  
 if self.target\_mode:  
 for dx, dy in self.directions:  
 x, y = self.last\_hit[0] + dx, self.last\_hit[1] + dy  
 if (x, y) in self.possible\_targets:  
 self.possible\_targets.remove((x, y))  
 return x, y  
 self.target\_mode = False  
 self.last\_hit = None  
  
 # Случайный выстрел  
 x, y = random.choice(self.possible\_targets)  
 self.possible\_targets.remove((x, y))  
 return x, y

import re

**ВЕДОМОСТЬ ДОКУМЕНТОВ**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Зона* | *Обозначение* | | | | | *Наименование* | | | Дополнительные сведения | | | |
|  |  | | | | |  | | |  | | | |
|  |  | | | | | *Текстовые документы* | | |  | | | |
|  | *ГУИР ДП 1-58 01 01 014 ПЗ* | | | | | *Пояснительная записка* | | | *30 с.* | | | |
|  | *ГУИР.502520.011 ПД* | | | | | *Схема алгоритма работы системы* | | | *Лист 1* | | | |
|  |  | | | | |  | | |  | | | |
|  |  | | | | |  | | |  | | | |
|  |  | | | | |  | | |  | | | |
|  |  | | | | |  | | |  | | | |
|  |  | | | | |  | | |  | | | |
|  |  | | | | |  | | |  | | | |
|  |  | | | | |  | | |  | | | |
|  |  | | | | |  | | |  | | | |
|  |  |  |  |  | ГУИР КП 1-58 01 01 011 КП | | | | | | | |
|  |  |  |  |  |
| *Изм.* | *Лист* | *№ докум.* | *Подп.* | *Дата* |
| *Разраб.* | | *Клещёв* |  |  | *Сетевая игра ”Морской бой”* | | *Лит.* | | | | *Лист* | *Листов* |
| *Пров.* | | *Болтак* |  |  |  | *У* | |  | *30* |  |
| *Т.контр.* | |  |  |  | *Кафедра ИПиЭ,*  *группа 310901* | | | | | |
| *Н.контр.* | |  |  |  |
| *Утв.* | |  |  |  |