Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерный системы и сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему:

СЕТЕВАЯ КОМПЬЮТЕРНАЯ ИГРА «CODENAMES»

БГУИР КП 1-40 01 01 047 ПЗ

Студент Сорочук Д. Н.

Руководитель Болтак С.В.

Минск 2025

СОДЕРЖАНИЕ

Введение 5
1 Анализ предметной области 6
1.1 Обзор аналогов 6
1.2 Постановка задачи 8
2 Проектирование программного средства 11
2.1 Архитектура приложения 11
2.2 Проектирование сетевого взаимодействия 12
2.3 Проектирование игрового процесса 14
2.4 Проектирование пользовательского интерфейса 16
3 Разработка программного средства 18
3.1 Реализация серверной части 18
3.2 Реализация игровой логики 19
3.3 Разработка пользовательского интерфейса 20
4 Тестирование программного средства 23
5 Руководство пользователя 24
5.1 Описание пользовательского интерфейса 24
5.2 Управление игровым процессом 24
5.3 Правила игры 26
Заключение 28
Список использованных источников 29
Приложение А. исходный код программы 30

ВВЕДЕНИЕ

Современные цифровые технологии предоставляют широкие возможности для реализации настольных игр в формате сетевых приложений, обеспечивая удобство взаимодействия между игроками независимо от их местоположения. Одной из популярных настольных игр, получивших широкое признание, является Codenames — игра на ассоциации и командное взаимодействие, в которой игроки делятся на команды и пытаются угадать слова на игровом поле, руководствуясь подсказками капитанов. Перевод этой игры в цифровой формат открывает новые перспективы, такие как автоматизация игрового процесса, поддержка многопользовательского режима в реальном времени и доступность через веб-браузеры.

Целью данной работы является разработка сетевой версии игры Codenames в виде веб-приложения, обеспечивающего возможность проведения игровых сессий в реальном времени с использованием современных веб-технологий. Основное внимание уделено созданию удобного пользовательского интерфейса, реализации стабильного сетевого взаимодействия и обеспечению корректной игровой логики, соответствующей правилам оригинальной настольной игры.

Для достижения поставленной цели были сформулированы следующие задачи:

1. Провести анализ предметной области, включая изучение аналогов и постановку требований к программному средству.
2. Спроектировать архитектуру приложения, включая клиентскую и серверную части, а также механизм сетевого взаимодействия.
3. Реализовать программное средство с использованием технологий FastAPI для серверной части, WebSocket для обмена данными в реальном времени и HTML/CSS/JavaScript для клиентской части.
4. Провести тестирование разработанного приложения для выявления и устранения ошибок.
5. Разработать руководство пользователя, описывающее интерфейс и правила игры.

В результате выполнения работы создано веб-приложение, позволяющее игрокам создавать игровые сессии, присоединяться к ним, распределяться по командам и играть в Codenames через браузер. Приложение поддерживает ключевые аспекты игры, такие как формирование игрового поля, распределение ролей (капитаны и агенты), предоставление подсказок и раскрытие слов, а также обеспечивает синхронизацию действий между всеми участниками в реальном времени.

Данная пояснительная записка описывает процесс разработки программного средства, включая анализ, проектирование, реализацию, тестирование и инструкции для пользователей. Работа направлена на демонстрацию возможностей современных веб-технологий в создании интерактивных многопользовательских приложений.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Обзор аналогов

В рамках анализа предметной области были рассмотрены две существующие сетевые реализации игры Codenames: веб-сервисы Codenames.me и Horsepaste.com. Цель анализа — изучить их функциональные возможности, интерфейс, доступность и сетевые особенности, чтобы определить преимущества и недостатки, а также выделить уникальные характеристики разрабатываемого программного средства.

Codenames.me — это браузерная платформа для игры в Codenames, разработанная Денисом Ольшиным. Сервис позволяет создавать игровые сессии с использованием уникального ключа, который определяет раскладку поля и распределение команд. Недавно платформа переехала на новый адрес cdna.me, где добавлены комнаты для хранения истории игр и упрощения повторных сессий.

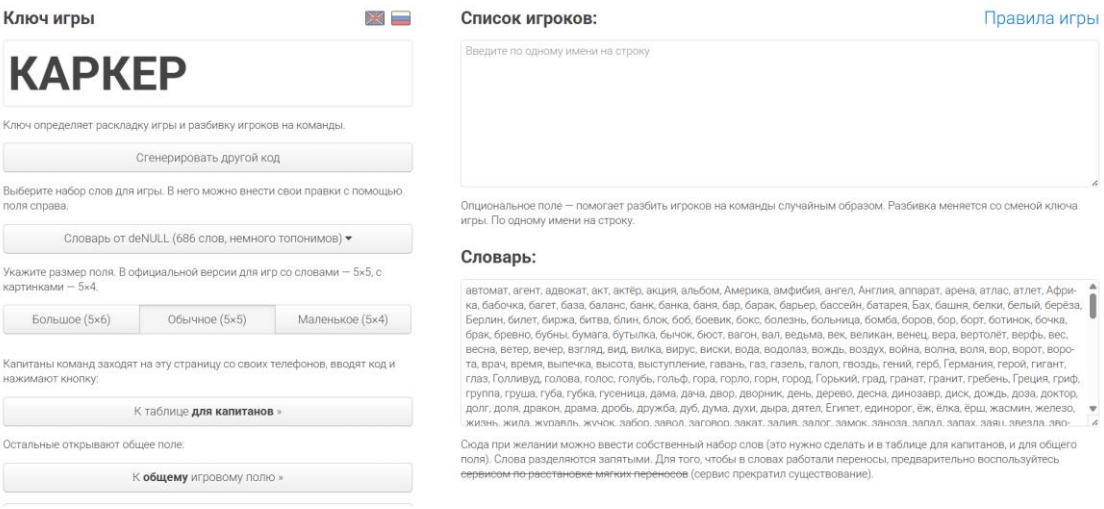

Рисунок 1.1 — Интерфейс Codenames.me

Основные функции:

- Создание игрового поля с настройкой размера (по умолчанию 5×5 для слов, 5×4 для картинок).
- Поддержка кастомных наборов слов, которые можно ввести через специальное поле (слова разделяются запятыми).
- Разделение интерфейсов для капитанов (с отображением ролей слов) и игроков (общее поле без ролей).
- Режим наблюдателя, предотвращающий случайные действия на поле.
- Возможность случайного распределения игроков по командам на основе списка имен.
- Хранение истории игр в комнатах (в новой версии на cdna.me).

Платформы: Браузер (ПК и мобильные устройства), не требует установки.

Преимущества:

- Бесплатный доступ без обязательной регистрации.
- Гибкость настройки: поддержка кастомных слов и разных размеров поля.
- Удобство для капитанов и игроков благодаря разделённым интерфейсам.
- Хранение истории игр в комнатах упрощает организацию повторных сессий.

Недостатки:

- Отсутствие встроенного чата или голосовой связи, что требует использования сторонних сервисов (например, Zoom).
- Необходимость ручного ввода ключа для каждой новой игры (в старой версии), что может быть неудобно.
- Ограниченная визуальная привлекательность интерфейса, ориентированного на функциональность.

Horsepaste.com — это бесплатная браузерная версия Codenames, разработанная на основе открытого исходного кода (GitHub: jbowens/codenames). Платформа ориентирована на простоту и доступность, позволяя игрокам создавать и делиться игровыми полями, которые обновляются в реальном времени по мере раскрытия слов.

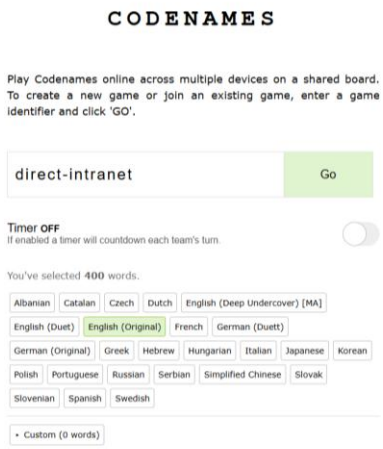

Рисунок 1.2 — Интерфейс Horsepaste.com

Основные функции:

- Генерация игрового поля (5×5) с возможностью просмотра в режимах спаймастера (с ролями) и обычного игрока (без ролей).
- Поддержка общего доступа к игровому полю через уникальную ссылку.
- Обновление поля в реальном времени при раскрытии слов.

– Минималистичный интерфейс, ориентированный на быстрый старт игры.

Платформы: Браузер (ПК и мобильные устройства), не требует установки.

Преимущества:

– Полностью бесплатный доступ без регистрации.
– Простота использования: игра начинается сразу после создания поля.
– Открытый исходный код, что позволяет сообществу вносить улучшения.
– Популярность среди игроков, подтверждённая отзывами на Reddit.

Недостатки:

– Ограниченные возможности кастомизации (например, нет поддержки пользовательских слов).
– Периодические проблемы с синхронизацией (например, сброс поля), о чём сообщали пользователи.

Сравнение с разрабатываемым программным средством:

Разработанное веб-приложение для Codenames, основанное на FastAPI и WebSocket, предлагает ряд преимуществ по сравнению с рассмотренными аналогами:

– Сетевая производительность: Использование WebSocket обеспечивает стабильную синхронизацию игрового состояния в реальном времени, устраняя проблемы с задержками или сбросами, как на Horsepaste.com.
– Интерфейс: Современный интерфейс на HTML/CSS/JavaScript обеспечивает визуальную привлекательность и интуитивность, превосходя минималистичный дизайн обоих аналогов.
– Встроенные функции: Возможность интеграции чата или уведомлений (если реализовано) устраняет необходимость в сторонних сервисах, как в обоих аналогах.

Разработанное приложение сочетает простоту и доступность Horsepaste.com с гибкостью Codenames.me, предлагая более надёжное и функциональное решение для сетевой игры в Codenames.

1.2 Постановка задачи

На основе анализа предметной области и обзора существующих аналогов была сформулирована задача разработки сетевой версии настольной игры Codenames в формате веб-приложения. Основная цель — создание функционального и удобного программного средства, которое позволит игрокам проводить игровые сессии в реальном времени через браузер, сохраняя ключевые механики оригинальной игры и обеспечивая стабильное сетевое взаимодействие.

Для реализации поставленной цели необходимо разработать веб-приложение, отвечающее следующим функциональным требованиям:

1. Создание и управление игровыми сессиями: Пользователи должны иметь возможность создавать новые игровые комнаты с уникальным

идентификатором и присоединяться к существующим сессиям по ссылке или коду.

2. Формирование игрового поля: Приложение должно генерировать игровое поле размером 5×5, состоящее из 25 слов, случайно распределённых по ролям (9 слов для одной команды, 8 — для другой, 7 нейтральных, 1 чёрное слово).
3. Поддержка ролей игроков: Реализовать разделение игроков на две команды с ролями капитанов (спаймастеров) и игроков (оперативников), обеспечивая разные интерфейсы: капитаны видят роли слов, игроки — только слова.
4. Игровой процесс: Обеспечить механику выбора слов игроками, смены хода и завершения игры при выполнении условий (все слова команды раскрыты или выбрано чёрное слово).
5. Кастомизация: Дать возможность пользователям выбирать наборы слов по языку слов, а также цвет команды, которая ходит первой.
6. Синхронизация в реальном времени: Реализовать механизм обновления игрового состояния (раскрытие слов, смена хода) для всех участников через WebSocket.
7. Пользовательский интерфейс: Разработать интуитивный интерфейс с главным меню, страницами правил, игровым экраном и настройками сессии, используя HTML/CSS/JavaScript.

Кроме того, приложение должно соответствовать следующим нефункциональным требованиям:

1. Кроссплатформенность: Приложение должно быть доступно через современные веб-браузеры (Chrome, Firefox, Safari) на ПК и мобильных устройствах без необходимости установки.
2. Производительность: Обеспечить минимальное время отклика сервера (менее 100 мс для WebSocket-соединений) и быструю загрузку игрового поля (менее 2 секунд).
3. Надёжность: Гарантировать стабильность соединения и корректную обработку ошибок, таких как отключение игрока или некорректный ввод.
4. Масштабируемость: Поддерживать одновременное участие до 20 игроков в одной сессии без значительного снижения производительности.
5. Удобство использования: Обеспечить интуитивный интерфейс, понятный новичкам, с минимальным количеством действий для начала игры.

Для реализации приложения выбраны следующие технологии:

– Серверная часть: FastAPI для обработки API-запросов и управления игровыми сессиями.
– Сетевое взаимодействие: WebSocket для синхронизации игрового состояния в реальном времени.

– Клиентская часть: HTML, CSS и JavaScript для создания динамического и адаптивного интерфейса.
– Хранение данных: База данных для хранения наборов слов и, при необходимости, истории сессий.

Разрабатываемое программное средство должно устранить недостатки аналогов и предоставить пользователям удобное и надёжное решение для сетевой игры в Codenames.

2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

2.1 Архитектура приложения

Архитектура программного средства для сетевой игры Codenames планируется реализовать на основе клиент-серверной модели, обеспечивающей разделение ответственности между сервером и клиентом. Основная цель проектирования — создать масштабируемую, надёжную и удобную систему, которая позволит игрокам участвовать в игре в реальном времени через веб-браузеры. Ниже описан общий план архитектуры, включая основные компоненты и их взаимодействие.

Архитектура будет включать следующие ключевые элементы:

– Серверная часть: Предусматривается использование фреймворка FastAPI для обработки запросов и управления игровыми сессиями. Сервер должен генерировать игровое поле, распределять роли игроков и обеспечивать синхронизацию через WebSocket. Для хранения наборов слов планируется использовать текстовые файлы или простую базу данных.
– Клиентская часть: Предполагается разработка интерфейса на HTML, CSS и JavaScript, который будет включать главное меню, страницу правил и игровой экран. Интерфейс должен быть адаптивным и поддерживать разделение ролей (капитаны и агенты) с разными представлениями данных.
– Сетевое взаимодействие: Планируется внедрение WebSocket для обмена данными в реальном времени, а также использование HTTP для начальной настройки сессий.

Планируемое взаимодействие компонентов

– Клиент отправляет запрос на создание игры через HTTP, сервер генерирует уникальный идентификатор сессии и возвращает данные для отображения игрового поля.
– Игроки подключаются к сессии через WebSocket, сервер рассылает обновления состояния (раскрытие слов, смена хода) всем участникам.
– Данные о словах и ролях хранятся на сервере и передаются клиентам в зашифрованном формате для предотвращения несанкционированного доступа.

Процесс создания игры представлен на рисунке 2.1:

Рисунок 2.1 — Диаграмма создания игры

Модульная структура приложения:

- Сервер:
  - Модуль «main.py» для обработки запросов и логики игры.
  - Директория «assets» для хранения файлов слов.
- Клиент:
  - Директория «templates» для HTML-страниц («home.html», «rules.html», «index.html»).
  - Директория «static» для стилей («styles.css») и предполагаемого скрипта («script.js»).

Архитектура проектируется с учётом возможности масштабирования (поддержка нескольких сессий) и интеграции дополнительных функций, таких как выбор языков.

2.2 Проектирование сетевого взаимодействия

Сетевое взаимодействие в программном средстве для сетевой игры Codenames будет реализовано с использованием комбинации HTTP-запросов и WebSocket-соединений, обеспечивая как начальную настройку игровых сессий, так и синхронизацию данных в реальном времени между всеми

участниками. Ниже описан процесс проектирования сетевого взаимодействия, включая используемые протоколы, маршруты и форматы данных.

Сетевое взаимодействие будет организовано следующим образом:

1. HTTP-запросы:
   - Маршрут «GET /» будет отображать главную страницу для настройки параметров игры. Клиент отправит запрос на создание сессии с параметрами, какая команда ходит первой и какой язык слов. Сервер сгенерирует уникальный «game_id» и перенаправит клиента на маршрут «GET /game/{game_id}» для отображения игрового экрана.
   - Маршрут «GET /api/game/{game_id}» будет возвращать состояние игры в формате JSON, содержащем список слов и их ролей, статус раскрытия слов, текущий ход, оставшиеся слова, флаг окончания игры, победитель, язык слов.
   - Маршрут «POST /game/{game_id}/reveal/{index}» будет обрабатывать раскрытие слова по индексу, обновлять состояние игры и рассылать изменения через WebSocket.
   - Маршрут «POST /game/{game_id}/end-turn» будет завершать ход текущей команды, передавая ход другой команде, и уведомлять участников через WebSocket.
2. WebSocket:
   - Маршрут «/ws/{game_id}» будет устанавливать WebSocket-соединение для каждого игрока в сессии. При подключении сервер отправит сообщение с необходимыми переменными. Все изменения состояния игры (раскрытие слова, смена хода, конец игры) будут передаваться в формате JSON. Клиент будет обновлять интерфейс на основе этих данных.

Процесс сетевого взаимодействия:

Клиент инициирует создание игровой сессии через HTTP-запрос, получая уникальный идентификатор игры. Затем он запрашивает начальное состояние игры для отображения игрового поля. После этого клиент подключается к WebSocket для получения обновлений состояния в реальном времени. Сервер рассылает всем игрокам изменения, например, при раскрытии слова или смене хода, обеспечивая синхронизацию данных, таких как текущий ход и счёт команд.

Блок-схема сетевого взаимодействия показана на рисунке 2.2.

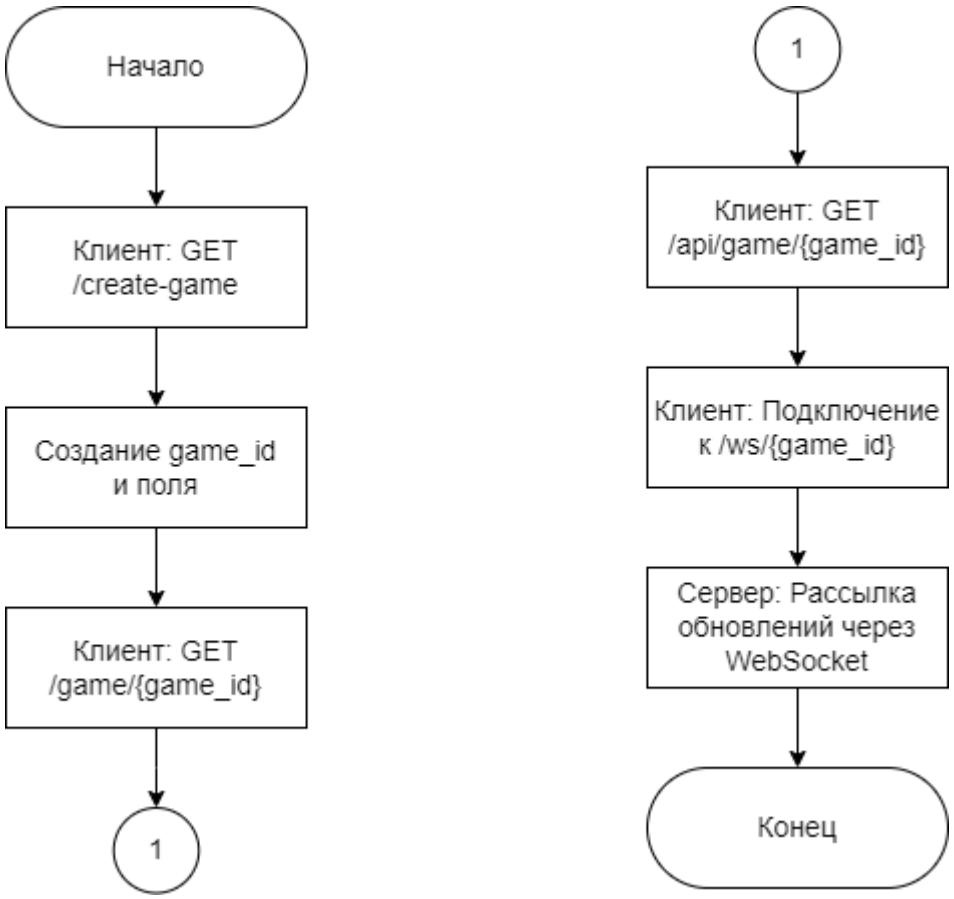

Рисунок 2.2 — Диаграмма сетевого взаимодействия

Все данные в приложении передаются в формате JSON. HTTP-ответы, такие как запрос на получение состояния игры, включают информацию о словах, их статусе, текущем ходе, количестве оставшихся карточек для каждой команды, статусе завершения игры и выбранном языке. WebSocket-сообщения также используют JSON для передачи событий, например, при раскрытии слова, включая его индекс, статус, идентификатор игры, текущий ход, счёт команд, состояние игры и информацию о победителе.

Сетевое взаимодействие будет спроектировано с учётом надёжности и масштабируемости (поддержка нескольких одновременных сессий).

2.3 Проектирование игрового процесса

Игровой процесс в сетевой игре Codenames будет реализован как командная игра на ассоциации, разделённая на два основных этапа: подготовку и ход игры, с чётко определёнными правилами победы и завершения. Ниже описан проектируемый процесс, включая роли игроков, этапы игры и условия окончания.

Подготовка к игре:

– Игра будет начинаться с создания сессии через маршрут GET, где будут задаваться параметры: первая команда и язык слов.

– Сервер сгенерирует игровое поле из 25 слов, распределённых следующим образом: 8-9 карт для первой команды, 8-9 для второй, 7 нейтральных и 1 «убийца». Распределение ролей будет случайным.
– Игроки распределятся на две команды: красную и синюю, каждая из которых будет включать ведущего (Spymaster) и полевых игроков.

Ход игры:

– Капитан команды, чей ход сейчас, будет давать подсказку, состоящую из одного слова и числа, указывающего количество связанных слов. Например, «Природа, 3» будет означать три слова, связанных с природой.
– Полевые агенты будут обсуждать подсказку и отправлять запрос для выбора слова.
– Если выбранное слово соответствует цвету команды, агенты смогут продолжить угадывать (до числа из подсказки плюс одна дополнительная попытка). Если цвет не совпадает, ход перейдёт к другой команде.
– Команда сможет завершить ход в любой момент, отправив запрос, после чего сервер изменит текущий ход на противоположную команду.

Игра будет завершаться в следующих случаях:

– Если команда раскроет все свои слова (счёт красных или синих станет равен 0), эта команда объявит победителем.
– Если команда раскроет слово «убийца», текущая команда проиграет, а противоположная станет победителем.

Процесс игры будет представлен на рисунке 2.3.

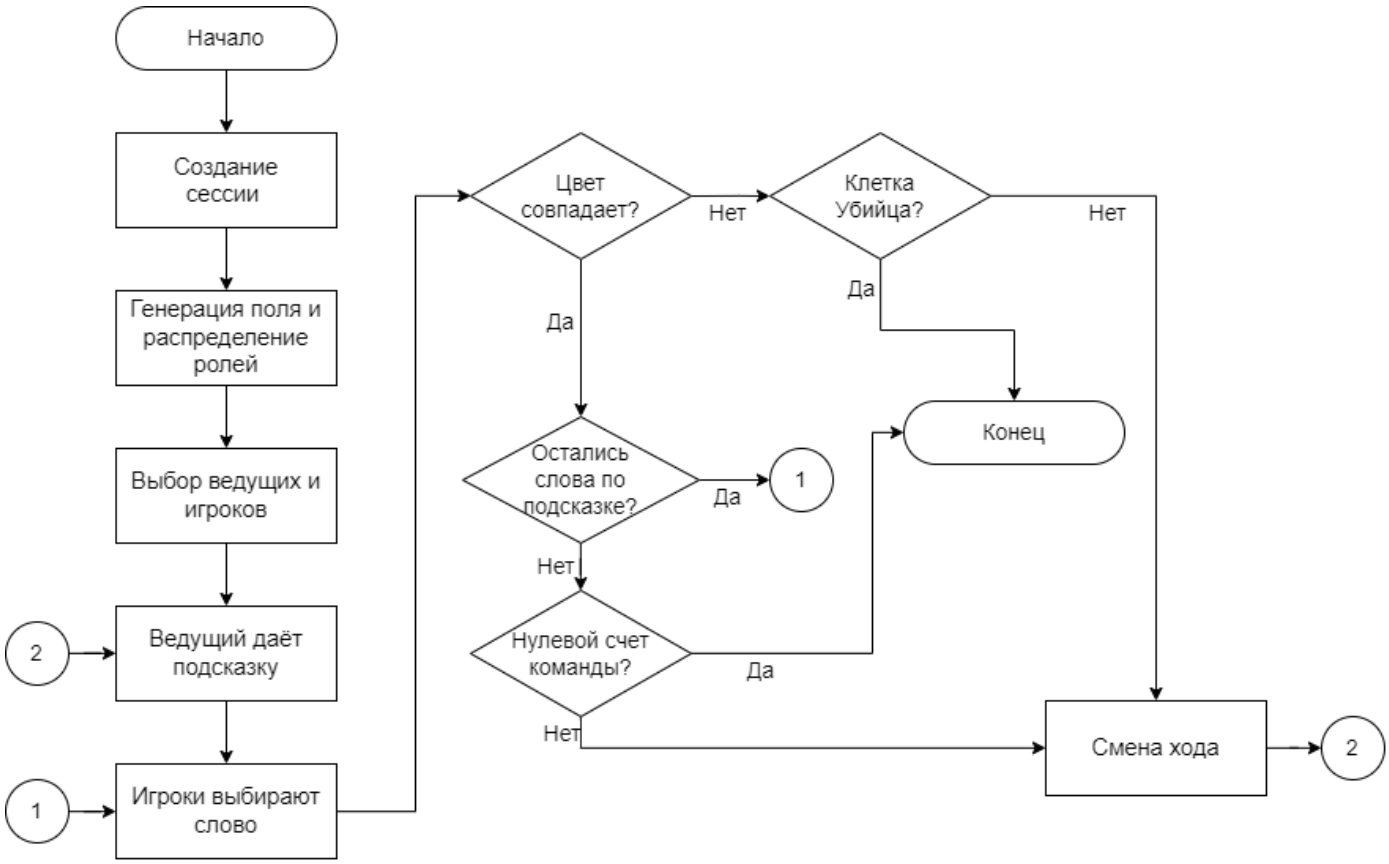

Рисунок 2.3 — Диаграмма игрового процесса

Особенности и ограничения

– Подсказки ведущего будут ограничиваться одним словом, без использования жестов или слов, присутствующих на поле.
– Система будет учитывать ошибки, такие как раскрытие «убийцы», и автоматически завершит игру с определением проигравшей команды.

Игровой процесс будет спроектирован с учётом интуитивности для пользователей и поддержки многопользовательского режима через сетевое взаимодействие.

2.4 Проектирование пользовательского интерфейса

Пользовательский интерфейс сетевой игры Codenames будет реализован как веб-приложение, обеспечивающее интуитивное управление и адаптивный дизайн для взаимодействия игроков через браузер. Интерфейс будет включать несколько ключевых страниц, каждая из которых будет выполнять определённую функцию в игровом процессе. Ниже описан проектируемый интерфейс, включая его структуру и функциональность.

Структура интерфейса:

– Главная страница: Будет отображаться при входе. Она будет содержать форму для настройки игры, включающую выбор первой

команды и языка слов. Кнопка «Начать игру» отправит запрос для создания новой сессии.

– Игровой экран: Будет отображать номер игры, текущий счёт (например, 9-8) и информацию о чьём ходе (например, «Ход красных»). Игровое поле с 25 словами будет динамически генерироваться и обновляться через JavaScript. Интерфейс включит кнопки «Завершить ход», «Поделиться игрой», «Новая игра» и «Правила игры». Также будет предусмотрено переключение ролей между «Игрок» и «Ведущий» с разным отображением данных (например, капитаны увидят цвета слов).
– Страница правил: Предоставит описание правил игры, включая роли, ходы, условия победы и стратегические советы. Страница будет содержать ссылки для возврата на главную или к текущей игре.

Функциональность интерфейса:

– Интерфейс будет адаптивным, поддерживая различные размеры экранов, чтобы обеспечить удобство на мобильных устройствах и настольных компьютерах.
– Динамическое обновление игрового поля и состояния (счёт, ход).
– Кнопка «Завершить ход» и выбор слов через POST–запросы.
– Переключение ролей (игрок/ведущий) будет менять видимость данных, где ведущий увидит цвета слов, а игроки — только раскрытые карты.

Интерфейс будет спроектирован с учётом удобства использования и поддержки мультиплеерного режима через сетевое взаимодействие.

3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

3.1 Реализация серверной части

Сетевая часть программного средства для игры Codenames реализована с использованием комбинации HTTP-запросов и WebSocket-соединений, обеспечивая как начальную настройку игровых сессий, так и синхронизацию данных в реальном времени. Реализация основана на фреймворке FastAPI, который позволит эффективно обрабатывать запросы и управлять мультиплеером. Ниже описан процесс реализации сетевой части, включая маршруты, протоколы и взаимодействие с клиентом.

Используемые маршруты и протоколы

1. HTTP-запросы:
   - Маршрут «GET /» отображает главную страницу для настройки параметров игры. Клиент может выбрать первую команду («first_team»: «random», «red», «blue») и язык слов («language»: «russian», «english»).
   - Маршрут «GET /create-game» создавает новую игровую сессию, генерируя уникальный «game_id» (случайное число от 1000 до 9999), формируя поле из 25 слов с распределением ролей (8-9 карт для одной команды, 8-9 для другой, 7 нейтральных, 1 «убийца») и перенаправляя клиента на «GET /game/{game_id}».
   - Маршрут «GET /game/{game_id}» отображает игровой экран с номером игры, счётом и текущим ходом. Если «game_id» не найден, клиент будет перенаправлен на главную страницу.
   - Маршрут «GET /api/game/{game_id}» возвращает состояние игры в формате JSON, включая поля: «words» (слова и роли), «revealed» (статус раскрытия), «turn» (текущий ход), «red_count» (оставшиеся красные слова), «blue_count» (оставшиеся синие слова), «game_ended» (флаг окончания), «winner» (победитель), «language» (язык).
   - Маршрут «POST /game/{game_id}/reveal/{index}» обрабатывает раскрытие слова по индексу, обновляя счёт, проверяя условия окончания игры (победа по счёту или «убийца») и рассылая изменения через WebSocket.
   - Маршрут «POST /game/{game_id}/end-turn» завершает ход текущей команды, переключая ход на противоположную команду и уведомляя игроков через WebSocket.
2. WebSocket:
   - Маршрут «/ws/{game_id}» устанавливает WebSocket-соединение для каждого игрока. При подключении сервер отправляет сообщение «{«type»: «connected», «game_id»: game_id}». Все обновления состояния (раскрытие слова, смена хода, конец игры) передаются в формате JSON с полями: «index», «revealed»,

«game_id», «turn», «red_count», «blue_count», «game_ended», «winner».

Процесс сетевого взаимодействия:

– Клиент начинает взаимодействие с отправки запроса «GET /create-game», который создаст сессию и перенаправит на игровой экран через «GET /game/{game_id}».
– На игровом экране клиент отправляет запрос «GET /api/game/{game_id}» для получения начального состояния и подключится к WebSocket через «/ws/{game_id}» для получения обновлений.
– При выборе слова клиент отправляет запрос «POST /game/{game_id}/reveal/{index}», сервер обновит состояние (например, уменьшит счёт, сменит ход) и разошлёт изменения через WebSocket.
– Кнопка «Завершить ход» отправляет «POST /game/{game_id}/end-turn», сервер переключит ход и уведомит игроков через WebSocket.

Форматы данных и обработка ошибок:

– HTTP-ответы возвращаются в формате JSON. Например, ответ на «GET /api/game/{game_id}» будет содержать поля: «words», «revealed», «turn», «red_count», «blue_count», «game_ended», «winner», «language».
– WebSocket-сообщения передаются в формате JSON. Пример сообщения при раскрытии слова: «{«index»: 5, «revealed»: true, «game_id»: «1234», «turn»: «blue», «red_count»: 8, «blue_count»: 7, «game_ended»: false, «winner»: null}».
– Обработка ошибок включает: проверку существования «game_id» (иначе ошибка 404), валидацию индекса слова (иначе ошибка 400), обработку отключения WebSocket-клиентов (удаление из списка игроков).

Сетевая часть реализована с учётом надёжности (обработка ошибок) и масштабируемости (поддержка нескольких сессий).

3.2 Реализация игровой логики

Игровая логика сетевой игры Codenames реализована в серверной части с использованием фреймворка FastAPI. Она обеспечивает создание игровых сессий, управление ходами, раскрытие слов и определение условий завершения игры. Реализация опирается на программные модули, которые обрабатывают действия игроков и синхронизируют состояние между участниками.

Сессия создаётся через маршрут GET /create-game. Генерируется уникальный идентификатор game_id (случайное число от 1000 до 9999), который сохраняется в словаре games. Выбираются 25 слов из словаря, соответствующего выбранному языку (language: «russian» или «english»).

Определяется первая команда (first_team). Если параметр first_team равен «random», команда выбирается случайным образом («red» или «blue»). В зависимости от первой команды распределяются роли: если первыми ходят красные, они получают 9 карт, а синие — 8; если первыми ходят синие, то наоборот. Оставшиеся карты включают 7 нейтральных и 1 «убийцу». Роли перемешиваются случайным образом и связываются с выбранными словами.

Игровой процесс управляется через поле turn, которое указывает текущую команду («red» или «blue»). Игроки раскрывают слова, отправляя запрос POST /game/{game_id}/reveal/{index}. Если слово принадлежит текущей команде (например, красное для команды «red»), счёт команды уменьшается (red_count или blue_count), и ход продолжается. Если слово принадлежит другой команде, является нейтральным или «убийцей», ход переходит к противоположной команде. Игроки также могут завершить ход, отправив запрос POST /game/{game_id}/end-turn, что переключает поле turn на противоположную команду.

Игра завершается в двух случаях. Первый случай: одна из команд раскрывает все свои слова, то есть red_count или blue_count достигает 0. В этом случае команда объявляется победителем, поле game_ended устанавливается в true, а winner принимает значение «red» или «blue». Второй случай: раскрывается слово с ролью «убийца». Тогда текущая команда проигрывает, а противоположная объявляется победителем. После завершения игры дальнейшие действия (например, раскрытие слов) блокируются.

Состояние игры хранится в словаре games, где каждая сессия содержит данные о словах, ролях, счёте и текущем ходе. Реализация учитывает синхронизацию через WebSocket, что позволяет обновлять состояние у всех игроков в реальном времени. Обработка ошибок включает проверку валидности индекса слова и статуса игры (например, игра уже завершена).

3.3 Разработка пользовательского интерфейса

Пользовательский интерфейс сетевой игры Codenames реализован как веб-приложение, обеспечивающее удобное взаимодействие игроков через браузер. Интерфейс включает несколько ключевых страниц, каждая из которых выполняет свою роль в игровом процессе. Реализация основана на шаблонах Jinja2, которые интегрированы с серверной частью через FastAPI.

Основные страницы интерфейса:

1. Главная страница (рисунок 3.1) отображается по маршруту GET / и содержит форму для настройки игры. Форма позволяет выбрать первую команду через параметр first_team с опциями «random», «red» или «blue» и язык слов через параметр language с опциями «russian» или «english». Кнопка «Начать игру» отправляет запрос GET /create-game, который создаёт новую игровую сессию и перенаправляет пользователя на игровой экран.

2. Игровой экран (рисунок 3.2) доступен по маршруту GET /game/{game_id}. На экране отображается номер игры (game_id), текущий счёт (например, 9-8) и информация о текущем ходе (например, «Ход красных»). Игровое поле из 25 слов генерируется динамически с помощью JavaScript, который обрабатывает данные, полученные через запрос GET /api/game/{game_id}. Экран включает кнопки: «Завершить ход» (отправляет запрос POST /game/{game_id}/end-turn), «Поделиться игрой», «Новая игра» (перенаправляет на главную страницу) и «Правила игры» (ведёт на страницу правил). Также реализован переключатель ролей между «Игрок» и «Ведущий», который позволяет изменять отображение данных: ведущий видит цвета всех слов, а игрок — только раскрытые карты.
3. Страница правил (рисунок 3.3) доступна по маршруту GET /rules. Она содержит описание правил игры, включая роли (капитан и агенты), ход игры, условия победы и стратегические советы. Страница предоставляет ссылки для возврата на главную страницу или к текущей игре, если указан параметр game_id.

Интерфейс использует шаблоны Jinja2 для рендеринга страниц. Динамическое обновление игрового поля и состояния (счёт, ход) реализуется через JavaScript, который взаимодействует с сервером через WebSocket-соединение по маршруту /ws/{game_id} и получает начальные данные через GET /api/game/{game_id}. Выбор слова осуществляется отправкой запроса POST /game/{game_id}/reveal/{index}, что обновляет состояние игры и синхронизирует его через WebSocket. Переключение ролей («Игрок»/»Ведущий») реализовано с использованием HTML-радиокнопок, а логика отображения обрабатывается на стороне клиента.

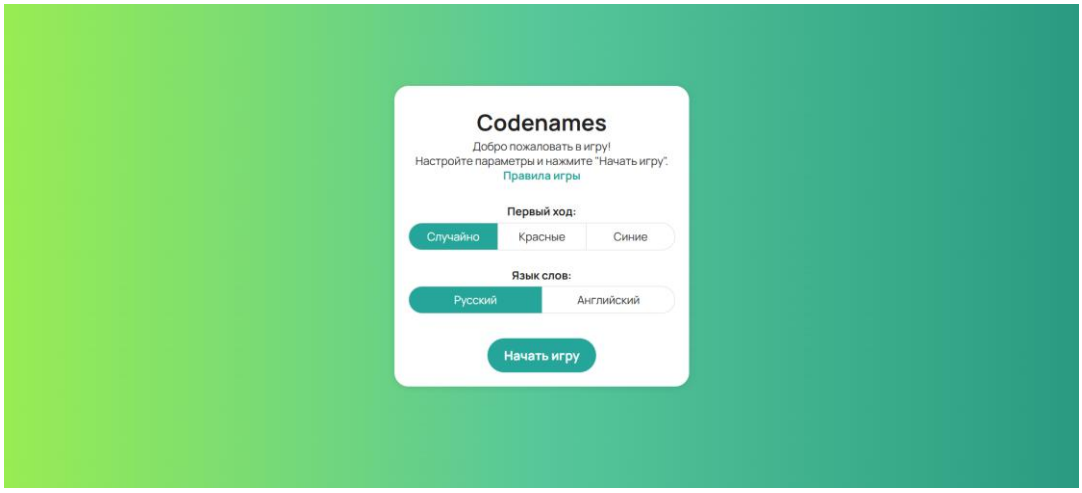

Рисунок 3.1 — Главная страница Codenames

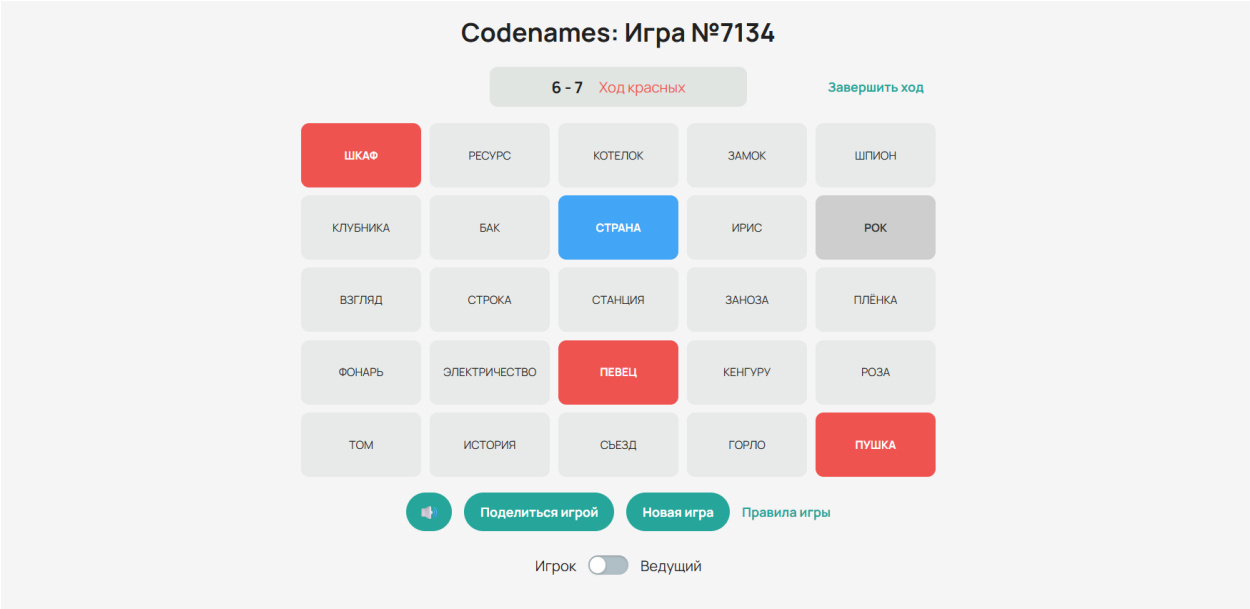

Рисунок 3.2 — Игровой экран Codenames

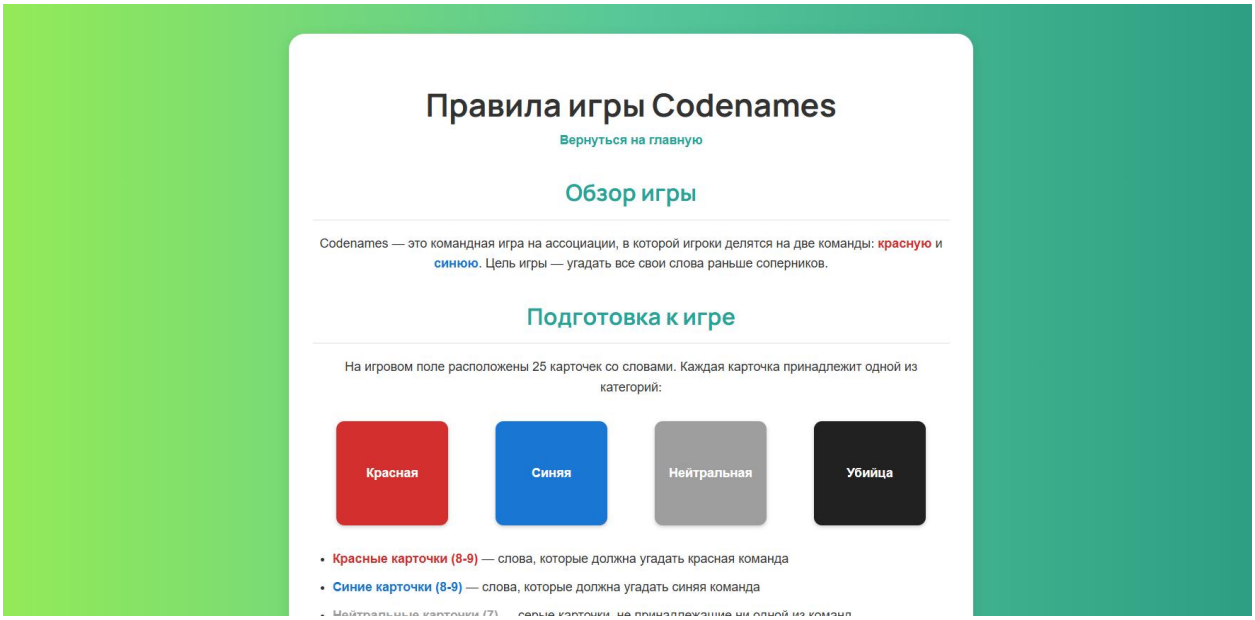

Рисунок 3.3 — Страница с правилами Codenames

Пользовательский интерфейс реализован с учётом базовых требований к многопользовательской игре, обеспечивая удобное взаимодействие и синхронизацию данных.

4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

На этапе тестирования программного средства проводилась проверка корректности функционирования всех основных компонентов веб-приложения: клиентской части (frontend), серверной логики (backend) и сетевого взаимодействия через WebSocket. В процессе тестирования были выявлены и устранены следующие недочёты:

1. Ошибка синхронизации игрового поля
   - Во время тестирования была обнаружена ошибка, при которой состояние игрового поля (25 слов с их ролями) не синхронизировалось корректно между игроками после раскрытия слова. Это происходило из-за отсутствия автоматического обновления данных на клиентской стороне после получения сообщения от сервера через WebSocket.
   - Решение: Для устранения проблемы была реализована функция обработки сообщений WebSocket на клиентской стороне. После получения события word_revealed от сервера клиент вызывает функцию обновления состояния игрового поля, используя JavaScript (функция updateBoard). Это обеспечивает немедленное отображение изменений для всех подключённых игроков.
2. Проблема с подключением новых игроков
   - При тестировании сценария подключения игрока к уже начатой игровой сессии возникала ошибка: новый игрок не получал текущее состояние игры (распределение ролей, раскрытые слова, текущий ход). Это было связано с тем, что сервер не отправлял полное состояние игры при новом подключении.
   - Решение: На серверной стороне (FastAPI) была добавлена логика отправки текущего состояния игры через WebSocket при событии player_joined. После успешного подключения нового игрока сервер формирует JSON-объект с данными об игровом поле, ролях и текущем ходе, который отправляется клиенту. Это позволило новым игрокам сразу видеть актуальное состояние игры.
3. Оптимизация загрузки слов для игрового поля
   - Изначально сервер загружал полный список слов из базы данных при каждом создании новой сессии, что приводило к задержкам, особенно при большом объёме слов. Это ухудшало опыт пользователя, так как создание игры занимало больше времени.
   - Решение: Была реализована оптимизация, заключающаяся в загрузке только 25 случайных слов для текущей игровой сессии. На серверной стороне добавлена функция, которая выбирает случайный поднабор слов из базы данных с помощью запроса random.sample. Это сократило время создания игры и улучшило производительность.

5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Настоящее руководство описывает использование сетевой игры Codenames, включая структуру пользовательского интерфейса, управление игровым процессом и правила игры. Руководство предназначено для игроков, желающих создать сессию, подключиться к игре или ознакомиться с её правилами.

5.1 Описание пользовательского интерфейса

Интерфейс игры Codenames представляет собой веб-приложение, доступное через браузер. Он включает несколько ключевых страниц, каждая из которых выполняет определённые функции.

Главное меню отображается по умолчанию при входе на сайт. На этой странице находится форма для настройки игры, включающая выпадающие списки для выбора первой команды («first_team» с опциями «random», «red», «blue») и языка слов («language» с опциями «russian», «english»). Кнопка «Начать игру» запускает процесс создания новой сессии. Интерфейс прост и интуитивен, позволяя быстро приступить к игре.

Игровой интерфейс доступен после создания сессии по уникальному идентификатору «game_id». Он показывает номер игры («game_id»), текущий счёт (например, 9-8) и чей ход (например, «Ход красных»). Игровое поле из 25 слов обновляется динамически, а кнопки «Завершить ход», «Поделиться игрой», «Новая игра» и «Правила игры» обеспечивают управление. Переключатель ролей («Игрок» и «Ведущий») позволяет выбрать режим отображения: игроки видят только раскрытые карты, а ведущий — цвета всех слов. Интерфейс адаптируется к различным размерам экрана для удобства использования.

Рекомендуемые скриншоты:
- Рисунок 6.1.1 — Главное меню: форма с настройками игры.
- Рисунок 6.1.2 — Игровой интерфейс: отображение счёта, хода, игрового поля и кнопок.

5.2 Управление игровым процессом

Раздел описывает шаги по созданию и подключению к игре, а также основные действия во время игрового процесса.

Процедура создания игровой сессии:

1. Откройте браузер и перейдите на главную страницу приложения.
2. В выпадающих списках выберите первую команду ходящих («Случайно», «Красные» или «Синие») и язык слов («Русский» или «Английский»). Пример выбора показан на рисунке 5.1

Рисунок 5.1 — Выбор параметров игры

3. Нажмите кнопку «Начать игру».
4. Система создаёт сессию с уникальным «game_id» и перенаправляет на игровой интерфейс. Поделитесь ссылкой, чтобы поделиться им с другими игроками. Пример созданной игры показан на рисунке 5.2.

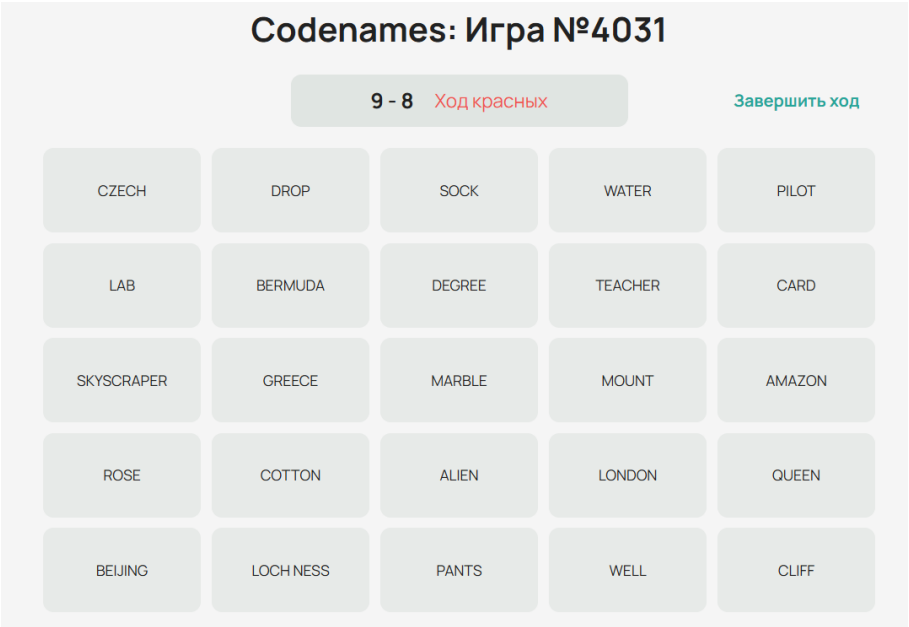

Рисунок 5.2 — Созданная игра с заданными параметрами

Процедура подключения к игре:

1. Получите «game_id» от создателя сессии.

2. Введите «game_id» в адресной строке браузера в формате «/game/{game_id}» (например, «/game/1234»).
3. Дождитесь загрузки игрового интерфейса и подключения через WebSocket.
4. Начните играть, следуя указаниям текущего хода и ролей.

Либо же просто получите готовую ссылку от создателя и перейдите по ней с помощью браузера. У вас на экране появится готовое игровое поле, в котором Вы сможете играть с друзьями.

Для выбора режима игрока необходимо использовать переключатель «Игрок – Ведущий», который находится снизу игрового экрана после игровых кнопок (рисунок 5.3).

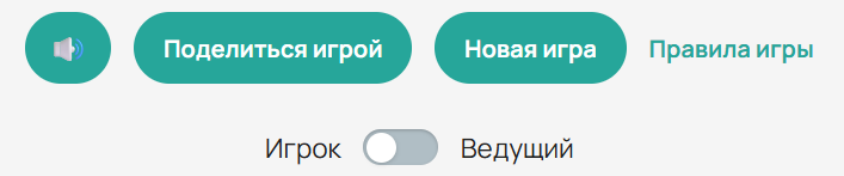

Рисунок 5.3 — Переключатель «Игрок – Ведущий»

Во время игрового процесса у команд может возникнуть необходимость завершить ход и передать его следующей команде. Для этого необходимо использовать специализированную кнопку, которая находится возле поля «Статус игры» (рисунок 5.4).

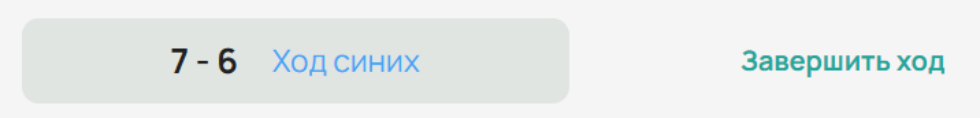

Рисунок 5.4 — Кнопка передачи игрового хода

5.3 Правила игры

Игра Codenames — это командная игра на ассоциации, где две команды (красная и синяя) соревнуются за раскрытие своих слов.

– Роли: Каждая команда включает капитана (ведущего) и полевых агентов. Ведущий видит цвета всех 25 слов, а агенты — только раскрытые карты.
– Ход игры: Ведущий даёт подсказку (одно слово и число, например, «Природа, 3»), указывая количество связанных слов. Агенты выбирают слова, отправляя запрос на раскрытие. Если цвет слова совпадает с командой, ход продолжается (до числа плюс одна попытка). Если цвет не совпадает, ход переходит к другой команде.

Игроки могут завершить ход вручную. Пример игры показан на рисунке 5.5.

– Окончание игры: Команда выигрывает, раскрыв все свои слова (9 или 8 в зависимости от первой команды). Если раскрыт «убийца», текущая команда проигрывает, а противоположная побеждает.

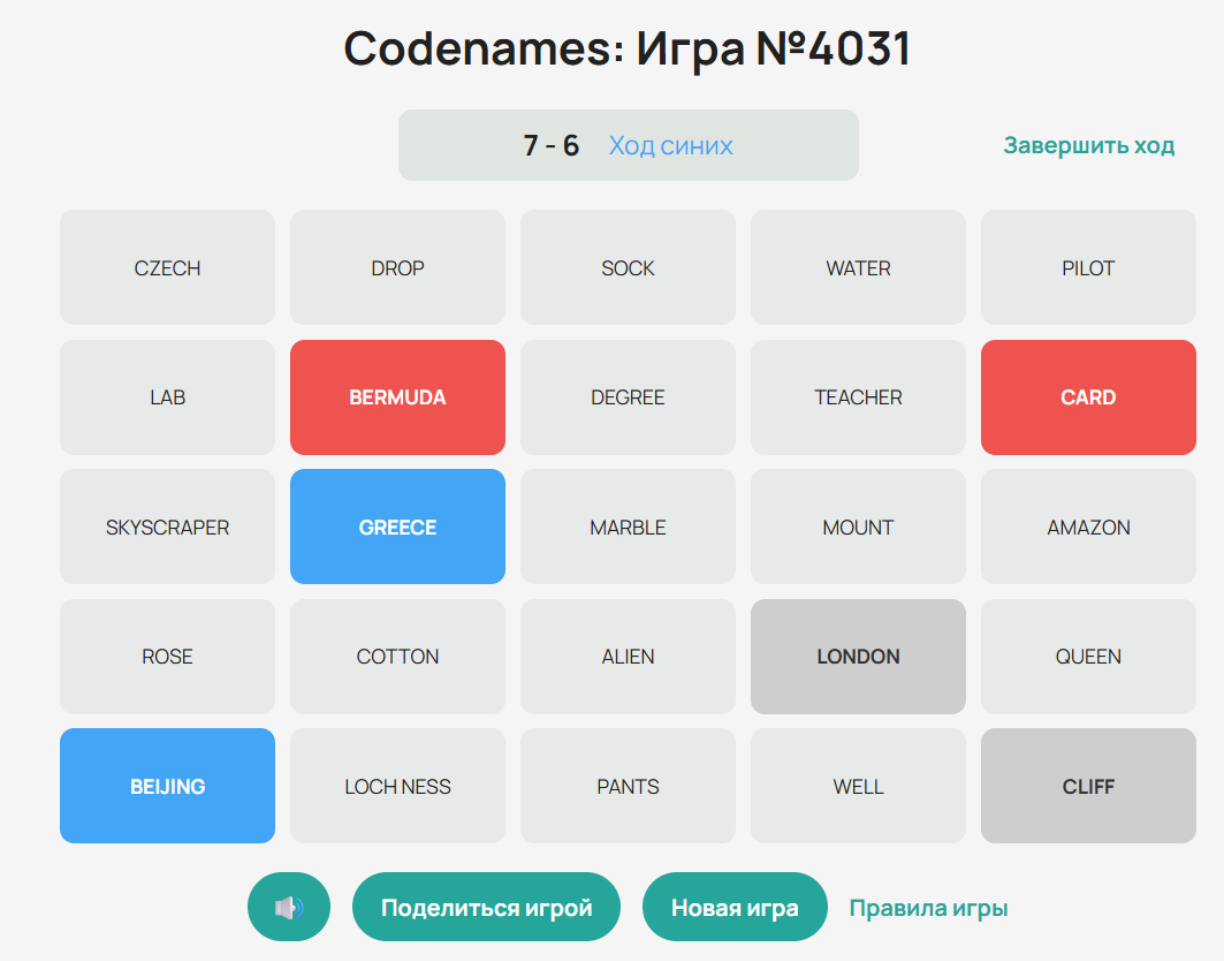

Рисунок 5.5 — Пример процесса игры

Руководство предоставляет полное описание для комфортного использования игры, включая настройку, подключение и соблюдение правил.

ЗАКЛЮЧЕНИЕ

В рамках данной работы разработано программное средство для сетевой игры Codenames, обеспечивающее командное взаимодействие игроков через веб-интерфейс с поддержкой мультиплеерного режима. Реализованы все запланированные этапы проектирования и разработки, включая проектирование и реализацию сетевой части, игровой логики, пользовательского интерфейса, а также составление руководства пользователя.

Сетевая часть функционирует на основе фреймворка FastAPI, который эффективно обрабатывает HTTP-запросы и WebSocket-соединения. Реализация позволяет создавать игровые сессии с уникальным идентификатором, синхронизировать состояние игры между игроками в реальном времени и обрабатывать действия пользователей. Для обеспечения надёжности добавлена обработка ошибок, включая проверку валидности и индексов слов, а также корректное управление отключением клиентов.

Игровая логика реализована с учётом правил Codenames: корректно распределяются роли (8-9 карт для каждой команды, 7 нейтральных, 1 «убийца»), управляются ходы команд, определяются условия победы (раскрытие всех слов команды) и проигрыша (раскрытие «убийцы»). Логика интегрирована с сетевой частью, что обеспечивает мгновенное обновление состояния игры для всех участников.

Пользовательский интерфейс реализован с использованием шаблонов Jinja2 для рендеринга страниц и JavaScript для динамического обновления данных. Интерфейс включает главную страницу для настройки игры, игровой экран с отображением поля, счёта и хода, а также страницу правил. Интуитивное расположение элементов управления интерфейса обеспечивают удобство использования.

Разработанное программное средство полностью соответствует поставленным требованиям, предоставляя функциональность для мультиплеерной игры, синхронизацию данных и удобный интерфейс. В ходе разработки достигнута стабильная работа приложения, а также реализована базовая масштабируемость, позволяющая поддерживать несколько одновременных игровых сессий.

В перспективе возможно расширение функциональности приложения. Среди возможных улучшений: добавление встроенного чата для общения между игроками, внедрение таймеров для ограничения времени на ход, что сделает игру более динамичной, а также улучшение адаптивности интерфейса для более комфортного использования на мобильных устройствах. Дополнительно можно рассмотреть локализацию интерфейса на другие языки, помимо русского и английского, и внедрение системы статистики, чтобы игроки могли отслеживать свои результаты и прогресс.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Официальная документация FastAPI [Электронный ресурс]. – Режим доступа: https://fastapi.tiangolo.com/.

[2] Документация Jinja2 [Электронный ресурс]. – Режим доступа: https://jinja.palletsprojects.com/.

[3] Руководство по WebSocket в Python [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/ru/docs/Web/API/WebSockets_API.

[4] GitHub Repositories for Multiplayer Game Development [Электронный ресурс]. – Режим доступа: https://github.com/topics/multiplayer-game.

[5] "Основы разработки сетевых игр на Python" [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/326144/.

[6] Официальная документация Python [Электронный ресурс]. – Режим доступа: https://docs.python.org/3/.

[7] Сайт для создания технической документации: Read the Docs [Электронный ресурс]. – Режим доступа: https://readthedocs.org/.

[8] "Введение в разработку веб-приложений с FastAPI" [Электронный ресурс]. – Режим доступа: https://www.digitalocean.com/community/tutorials/build-a-web-application-with-fastapi.

[9] Стандарт предприятия. Дипломные проекты (работы). [Электронный ресурс]. – Режим доступа: СТП_2024.pdf

[10] "Советы по созданию пользовательских интерфейсов для игр" [Электронный ресурс]. – Режим доступа: https://www.gamedeveloper.com/design/creating-intuitive-ui-for-games.

ПРИЛОЖЕНИЕ А. Исходный код программы

Main.py:

```
from fastapi import FastAPI, WebSocket, Request, HTTPException
from fastapi.templating import Jinja2Templates
from fastapi.staticfiles import StaticFiles
from fastapi.responses import RedirectResponse
from typing import Optional
import random
import json
import os

app = FastAPI()
app.mount("/static", StaticFiles(directory="static"), name="static")
templates = Jinja2Templates(directory="templates")

# Словари слов
WORDS = {
    "russian": [],
    "english": []
}

# Читаем русские слова
RUSSIAN_WORDS_FILE = "assets/russian-words-caps.txt"
if not os.path.exists(RUSSIAN_WORDS_FILE):
    raise RuntimeError(f"File {RUSSIAN_WORDS_FILE} not found")
with open(RUSSIAN_WORDS_FILE, encoding="utf-8") as f:
    WORDS["russian"] = [line.strip() for line in f if line.strip()]
if len(WORDS["russian"]) < 25:
    raise RuntimeError(f"Not enough words in {RUSSIAN_WORDS_FILE}, found
{len(WORDS['russian'])}, need at least 25")

# Читаем английские слова
ENGLISH_WORDS_FILE = "assets/english-words-caps.txt"
if not os.path.exists(ENGLISH_WORDS_FILE):
    raise RuntimeError(f"File {ENGLISH_WORDS_FILE} not found")
with open(ENGLISH_WORDS_FILE, encoding="utf-8") as f:
    WORDS["english"] = [line.strip() for line in f if line.strip()]
if len(WORDS["english"]) < 25:
    raise RuntimeError(f"Not enough words in {ENGLISH_WORDS_FILE}, found
{len(WORDS['english'])}, need at least 25")

# Хранилище игр
games = {}
```

```
# Стартовый экран
@app.get("/")
async def home(request: Request):
    return templates.TemplateResponse("home.html", {"request": request})

# Страница с правилами игры
@app.get("/rules")
async def rules(request: Request, game_id: Optional[str] = None):
    return templates.TemplateResponse("rules.html", {"request": request, "game_id": game_id})

# Создание игры
@app.get("/create-game")
async def create_game(request: Request, first_team: str = "random", language: str = "russian"):
    game_id = str(random.randint(1000, 9999))
    while game_id in games:
        game_id = str(random.randint(1000, 9999))

    # Выбираем словарь в зависимости от языка
    if language not in WORDS:
        language = "russian"  # По умолчанию используем русский

    words = random.sample(WORDS[language], 25)

    # Определяем, какая команда ходит первой
    if first_team == "random":
        first_team = random.choice(["red", "blue"])

    # Если первыми ходят красные, то у них 9 карт, у синих 8
    # Если первыми ходят синие, то у них 9 карт, у красных 8
    if first_team == "red":
        red_count = 9
        blue_count = 8
        roles = ["red"] * 9 + ["blue"] * 8 + ["neutral"] * 7 + ["assassin"] * 1
    else:  # first_team == "blue"
        red_count = 8
        blue_count = 9
        roles = ["red"] * 8 + ["blue"] * 9 + ["neutral"] * 7 + ["assassin"] * 1

    random.shuffle(roles)

    games[game_id] = {
        "words": list(zip(words, roles)),
        "revealed": [False] * 25,
        "players": [],
        "turn": first_team,
```

```
        "red_count": red_count,
        "blue_count": blue_count,
        "game_ended": False,
        "winner": None,
        "language": language
    }

    print(f"Created game {game_id} with first team: {first_team}, language: {language}")
    return RedirectResponse(url=f"/game/{game_id}")

# Показ игры
@app.get("/game/{game_id}")
async def show_game(game_id: str, request: Request):
    if game_id not in games:
        return RedirectResponse(url="/")
    return templates.TemplateResponse("index.html", {"request": request, "game_id": game_id})

# API для состояния игры
@app.get("/api/game/{game_id}")
async def get_game(game_id: str):
    if game_id not in games:
        raise HTTPException(status_code=404, detail="Game not found")
    return {
        "words": games[game_id]["words"],
        "revealed": games[game_id]["revealed"],
        "turn": games[game_id]["turn"],
        "red_count": games[game_id]["red_count"],
        "blue_count": games[game_id]["blue_count"],
        "game_ended": games[game_id]["game_ended"],
        "winner": games[game_id]["winner"],
        "language": games[game_id].get("language", "russian")  # По умолчанию русский, если
не указано
    }

# WebSocket для мультиплеера
@app.websocket("/ws/{game_id}")
async def websocket_endpoint(websocket: WebSocket, game_id: str):
    await websocket.accept()
    if game_id not in games:
        await websocket.close(code=1008, reason="Game not found")
        return
    games[game_id]["players"].append(websocket)
    print(f"Player connected to {game_id}, total players: {len(games[game_id]['players'])}")
    try:
        await websocket.send_json({"type": "connected", "game_id": game_id})
```

```
        while True:
            data = await websocket.receive_text()
            print(f"Received WebSocket message in {game_id}: {data}")
            for player in games[game_id]["players"]:
                await player.send_text(data)
    except Exception as e:
        print(f"WebSocket error in {game_id}: {e}")
        games[game_id]["players"].remove(websocket)

# Раскрытие слова
@app.post("/game/{game_id}/reveal/{index}")
async def reveal_word(game_id: str, index: int):
    if game_id not in games:
        raise HTTPException(status_code=404, detail="Game not found")
    if not 0 <= index < 25:
        raise HTTPException(status_code=400, detail="Invalid index")
    if games[game_id]["revealed"][index] or games[game_id]["game_ended"]:
        return {"status": "ok"}  # Уже раскрыто или игра окончена
    games[game_id]["revealed"][index] = True
    color = games[game_id]["words"][index][1]
    current_turn = games[game_id]["turn"]

    # Обновляем счёт
    if color == "red":
        games[game_id]["red_count"] -= 1
    elif color == "blue":
        games[game_id]["blue_count"] -= 1

    # Проверяем конец игры
    if games[game_id]["red_count"] == 0:
        games[game_id]["game_ended"] = True
        games[game_id]["winner"] = "red"
    elif games[game_id]["blue_count"] == 0:
        games[game_id]["game_ended"] = True
        games[game_id]["winner"] = "blue"
    elif color == "assassin":
        games[game_id]["game_ended"] = True
        games[game_id]["winner"] = "blue" if current_turn == "red" else "red"  #
Противоположная команда выигрывает

    # Логика смены хода
    if not games[game_id]["game_ended"]:
        if (current_turn == "red" and color != "red") or (current_turn == "blue" and color !=
"blue"):
            games[game_id]["turn"] = "blue" if current_turn == "red" else "red"
```

```
    print(f"Revealed word {index} in game {game_id}, color: {color}, turn:
{games[game_id]['turn']}, game_ended: {games[game_id]['game_ended']}")
    message = json.dumps({
        "index": index,
        "revealed": True,
        "game_id": game_id,
        "turn": games[game_id]["turn"],
        "red_count": games[game_id]["red_count"],
        "blue_count": games[game_id]["blue_count"],
        "game_ended": games[game_id]["game_ended"],
        "winner": games[game_id]["winner"]
    })
    for player in games[game_id]["players"]:
        try:
            await player.send_text(message)
            print(f"Sent WebSocket message to player in {game_id}: {message}")
        except Exception as e:
            print(f"Failed to send WebSocket message in {game_id}: {e}")
    return {"status": "ok"}

# Передача хода
@app.post("/game/{game_id}/end-turn")
async def end_turn(game_id: str):
    if game_id not in games:
        raise HTTPException(status_code=404, detail="Game not found")
    if games[game_id]["game_ended"]:
        return {"status": "ok"}  # Игра уже окончена

    # Меняем ход на противоположную команду
    current_turn = games[game_id]["turn"]
    games[game_id]["turn"] = "blue" if current_turn == "red" else "red"

    print(f"Turn ended in game {game_id}, new turn: {games[game_id]['turn']}")
    message = json.dumps({
        "game_id": game_id,
        "turn": games[game_id]["turn"],
        "red_count": games[game_id]["red_count"],
        "blue_count": games[game_id]["blue_count"],
        "game_ended": games[game_id]["game_ended"],
        "winner": games[game_id]["winner"],
        "turn_changed": True
    })
    for player in games[game_id]["players"]:
        try:
```

```
            await player.send_text(message)
            print(f"Sent WebSocket message to player in {game_id}: {message}")
        except Exception as e:
            print(f"Failed to send WebSocket message in {game_id}: {e}")
    return {"status": "ok"}
```

Index.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Codenames</title>
    <script src="https://unpkg.com/htmx.org@1.9.10"></script>
    <script src="/static/js/ws.js"></script>
    <script src="/static/js/sounds.js"></script>
    <link rel="stylesheet" href="/static/styles.css">
</head>
<body hx-ext="ws" ws-connect="/ws/{{ game_id }}">
    <h1>Codenames: Игра №{{ game_id }}</h1>
    <div class="game-status-wrapper">
        <div id="game-status" class="game-status" data-game='{"turn": "red", "red_count": 9,
"blue_count": 8, "game_ended": false, "winner": null}'>
            <span id="score">9 - 8</span>
            <span id="turn" class="turn-red">Ход красных</span>
            <span id="game-end-message" style="display: none;"></span>
        </div>
        <a id="end-turn-button" class="end-turn-link"
          hx-post="/game/{{ game_id }}/end-turn"
          hx-swap="none"
          hx-target="#game-board" style="color: #26A69A; text-decoration: none; font-weight:
bold; margin: auto;;">
            Завершить ход
        </a>
    </div>

    <div id="game-board"
        hx-get="/api/game/{{ game_id }}"
        hx-trigger="load"
        hx-swap="innerHTML">
        <p>Загрузка игры...</p>
    </div>

    <div class="controls-container">
        <!-- <div class="role-switcher">
            <span>Игрок</span>
            <label class="toggle">
```

```
            <input type="checkbox" id="role-toggle" onchange="toggleRole()">
            <span class="slider"></span>
          </label>
          <span>Ведущий</span>
        </div> -->

        <div class="game-controls">
          <button class="btn" onclick="shareGame()">Поделиться игрой</button>
          <a href="/" class="btn">Новая игра</a>
          <a href="/rules?game_id={{ game_id }}" style="color: #26A69A; text-decoration: none;
font-weight: bold; margin: auto;;">Правила игры</a>
        </div>
        <div class="role-switcher">
          <span>Игрок</span>
          <label class="toggle">
            <input type="checkbox" id="role-toggle" onchange="toggleRole()">
            <span class="slider"></span>
          </label>
          <span>Ведущий</span>
        </div>
      </div>

      <script>
        let currentRole = 'player'; // По умолчанию игрок

        function toggleRole() {
          const toggle = document.getElementById('role-toggle');
          currentRole = toggle.checked ? 'spymaster' : 'player';
          console.log('Switched role to:', currentRole);
          updateGameBoard();
        }

        function updateGameBoard() {
          const role = currentRole;
          document.getElementById('role-toggle').checked = (role === 'spymaster');
          htmx.ajax('GET', '/api/game/{{ game_id }}', {
            target: '#game-board',
            swap: 'innerHTML',
            headers: {'X-Custom-Header': 'value'}
          }).then(response => {
            console.log('Game board updated for role:', role);
            const gameData = response.detail.xhr.responseText ?
JSON.parse(response.detail.xhr.responseText) : {};
            document.getElementById('game-board').dataset.game = JSON.stringify(gameData);
            updateGameStatus(gameData);
```

```
        });
    }

    function updateGameStatus(data) {
        const status = document.getElementById('game-status');
        const turnText = document.getElementById('turn');
        const score = document.getElementById('score');
        const gameEndMessage = document.getElementById('game-end-message');
        const endTurnButton = document.getElementById('end-turn-button');
        const gameData = data || JSON.parse(status.dataset.game || '{}');
        console.log('Game data in updateGameStatus:', gameData);

        if (gameData.game_ended) {
            // Скрываем счет и информацию о ходе
            score.style.display = 'none';
            turnText.style.display = 'none';
            // Показываем сообщение о победе
            gameEndMessage.style.display = 'block';
            if (gameData.winner) {
                gameEndMessage.textContent = gameData.winner === 'red' ? 'Победа красных!' :
'Победа синих!';
                gameEndMessage.className = gameData.winner === 'red' ? 'game-end-win-red' :
'game-end-win-blue';
            }
            // Скрываем кнопку завершения хода
            endTurnButton.style.display = 'none';
        } else {
            // Показываем счет и информацию о ходе
            score.style.display = 'inline';
            score.textContent = `${gameData.red_count || 0} - ${gameData.blue_count || 0}`;
            turnText.style.display = 'inline';
            turnText.textContent = gameData.turn === 'red' ? 'Ход красных' : 'Ход синих';
            turnText.className = (gameData.turn === 'red' ? 'turn-red' : 'turn-blue') || '';
            // Скрываем сообщение о победе
            gameEndMessage.style.display = 'none';
            console.log('Applied class:', turnText.className);

            // Показываем кнопку завершения хода и обновляем её стиль
            if (currentRole === 'spymaster') {
                // Ведущий не может завершать ход
                endTurnButton.style.display = 'none';
            } else {
                // Обновляем стиль кнопки в зависимости от текущего хода
                endTurnButton.style.display = 'inline';
                endTurnButton.className = 'end-turn-link ' +
```

```
                    (gameData.turn === 'red' ? 'red-turn' : 'blue-turn');
            }
        }
    }

    htmx.on("htmx:afterSwap", function(evt) {
        if (evt.detail.target.id === 'game-board') {
            try {
                const game = JSON.parse(evt.detail.xhr.responseText);
                let html = "<div class='grid'>";
                game.words.forEach((word, index) => {
                    const revealed = game.revealed[index];
                    const role = word[1]; // red, blue, neutral, assassin
                    let backgroundClass, textClass, clickable;

                    if (currentRole === 'spymaster') {
                        // Для ведущего: фон серый, если не раскрыто, иначе цвет роли
                        backgroundClass = revealed ? role : 'hidden';
                        // Текст всегда в цвете роли для нераскрытых, для раскрытых будет применен стиль из CSS
                        textClass = revealed ? '' : `text-${role}`;
                        clickable = 'non-clickable'; // Ведущий не может кликать
                    } else {
                        // Для игрока: как раньше
                        backgroundClass = revealed ? role : 'hidden';
                        textClass = revealed ? '' : 'text-hidden';
                        clickable = (currentRole === 'player' && !revealed && !game.game_ended) ? '' : 'non-clickable';
                    }

                    html += `<div class="card ${backgroundClass} ${clickable}"
                            ${currentRole === 'player' && !revealed && !game.game_ended ? `hx-post="/game/{{ game_id }}/reveal/${index}" hx-trigger="click" hx-swap="none" hx-target="#game-board"` : ''}>
                            <span class="${textClass}">${word[0]}</span>
                        </div>`;
                });
                html += "</div>";
                evt.detail.target.innerHTML = html;
                console.log("Game board updated with cards for role:", currentRole);
                htmx.process(document.getElementById('game-board'));
                document.getElementById('game-board').dataset.game = JSON.stringify(game);
                updateGameStatus(game);
            } catch (e) {
                console.error("Error parsing game data:", e);
```

```
            }
        }
    });

    htmx.on("htmx:wsOpen", function(evt) {
        console.log("HTMX WebSocket connected for game {{ game_id }}");
    });

    htmx.on("htmx:wsClose", function(evt) {
        console.log("HTMX WebSocket closed:", evt.detail);
    });

    htmx.on("htmx:wsMessage", function(evt) {
        console.log("HTMX WebSocket raw message:", evt.detail.message, typeof
evt.detail.message);
        try {
            const data = JSON.parse(evt.detail.message);
            console.log("HTMX WebSocket parsed data:", data);
            if (String(data.game_id) === "{{ game_id }}") {
                // Получаем текущее состояние игры
                const gameBoard = document.getElementById('game-board');
                const gameData = gameBoard.dataset.game ? JSON.parse(gameBoard.dataset.game)
: {};

                // Обработка раскрытия карточки
                if (data.index !== undefined) {
                    console.log("Updating game board for index:", data.index);

                    // Если есть данные о словах, воспроизводим звук в зависимости от цвета
карточки
                    if (gameData.words && gameData.words[data.index]) {
                        const cardColor = gameData.words[data.index][1];
                        playCardSound(cardColor);
                    }
                }

                // Проверяем, изменился ли ход (при раскрытии карточки или при явном
завершении хода)
                const prevTurn = gameData.turn;
                const newTurn = data.turn;
                if (prevTurn && newTurn && prevTurn !== newTurn) {
                    playTurnChangeSound();
                }

                // Проверяем, закончилась ли игра
```

```
                if (data.game_ended && !gameData.game_ended) {
                    // Определяем, выиграла ли текущая команда
                    // Здесь можно добавить логику определения команды игрока
                    const isWinner = true; // Упрощенно считаем, что игрок всегда в выигрышной
команде
                    playGameEndSound(isWinner);
                }

                // Обновляем игровое поле
                htmx.ajax('GET', '/api/game/{{ game_id }}', {
                    target: '#game-board',
                    swap: 'innerHTML'
                }).then(() => {
                    console.log("Game board updated via HTMX WebSocket");
                }).catch((e) => {
                    console.error("Error updating game board:", e);
                });
            } else if (data.type === "connected") {
                console.log("HTMX WebSocket connection confirmed for game:", data.game_id);
            }
        } catch (e) {
            console.error("Error parsing HTMX WebSocket message:", e);
        }
    });

    htmx.on("htmx:afterRequest", function(evt) {
        console.log("HTMX request:", evt.detail.xhr.status, evt.detail.xhr.response);
        if (evt.detail.xhr.status !== 200) {
            console.error("Request failed:", evt.detail.xhr.statusText);
        } else if (evt.detail.elt.classList.contains('card') || evt.detail.elt.id === 'end-turn-button') {
            console.log("Triggering game board update after POST");
            htmx.ajax('GET', '/api/game/{{ game_id }}', {
                target: '#game-board',
                swap: 'innerHTML'
            });
        }
    });

    function shareGame() {
        const gameUrl = window.location.href;

        if (navigator.share) {
            // Используем Web Share API, если доступно
            navigator.share({
                title: 'Codenames Game',
```

```
                text: 'Присоединяйтесь к игре Codenames!',
                url: gameUrl
            }).catch(err => {
                console.error('Ошибка при попытке поделиться:', err);
                fallbackShare();
            });
        } else {
            fallbackShare();
        }

        function fallbackShare() {
            // Резервный вариант - копирование в буфер обмена
            navigator.clipboard.writeText(gameUrl).then(() => {
                alert('Ссылка на игру скопирована в буфер обмена!');
            }).catch(err => {
                console.error('Не удалось скопировать ссылку:', err);
                prompt('Скопируйте ссылку на игру:', gameUrl);
            });
        }
    }

    document.addEventListener('DOMContentLoaded', function() {
        if (typeof htmx === 'undefined') {
            console.error('HTMX not loaded!');
        } else {
            console.log('HTMX loaded successfully');
            updateGameBoard();
        }
    });

    // Резервный WebSocket
    const ws = new WebSocket('ws://localhost:8000/ws/{{ game_id }}');
    ws.onopen = () => {
        console.log('JS WebSocket connected for game {{ game_id }}');
    };
    ws.onmessage = (msg) => {
        console.log('JS WebSocket raw message:', msg.data, typeof msg.data);
        try {
            const data = JSON.parse(msg.data);
            console.log('JS WebSocket parsed data:', data);
            if (String(data.game_id) === "{{ game_id }}") {
                // Получаем текущее состояние игры
                const gameBoard = document.getElementById('game-board');
                const gameData = gameBoard.dataset.game ? JSON.parse(gameBoard.dataset.game)
: {};
```

```
                // Обработка раскрытия карточки
                if (data.index !== undefined) {
                    console.log('JS WebSocket updating game board for index:', data.index);

                    // Если есть данные о словах, воспроизводим звук в зависимости от цвета
карточки
                    if (gameData.words && gameData.words[data.index]) {
                        const cardColor = gameData.words[data.index][1];
                        playCardSound(cardColor);
                    }
                }

                // Проверяем, изменился ли ход (при раскрытии карточки или при явном
завершении хода)
                const prevTurn = gameData.turn;
                const newTurn = data.turn;
                if (prevTurn && newTurn && prevTurn !== newTurn) {
                    playTurnChangeSound();
                }

                // Проверяем, закончилась ли игра
                if (data.game_ended && !gameData.game_ended) {
                    // Определяем, выиграла ли текущая команда
                    // Здесь можно добавить логику определения команды игрока
                    const isWinner = true; // Упрощенно считаем, что игрок всегда в выигрышной
команде
                    playGameEndSound(isWinner);
                }

                htmx.ajax('GET', '/api/game/{{ game_id }}', {
                    target: '#game-board',
                    swap: 'innerHTML'
                }).then(() => {
                    console.log('Game board updated via JS WebSocket');
                });
            } else if (data.type === 'connected') {
                console.log('JS WebSocket connection confirmed for game:', data.game_id);
            }
        } catch (e) {
            console.error('Error parsing JS WebSocket message:', e);
        }
    };
    ws.onclose = () => {
        console.log('JS WebSocket closed');
```

```
        };
        ws.onerror = (e) => {
            console.error('JS WebSocket error:', e);
        };
    </script>
</body>
</html>
```

Home.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Codenames - Главная</title>
    <link rel="stylesheet" href="/static/styles.css">
    <style>
        html, body {
            height: 100%;
            margin: 0;
            padding: 0;
        }
        body {
            display: flex;
            justify-content: center;
            align-items: center;
            min-height: 100vh;
            background: #2A7B9B;
            background: linear-gradient(90deg, rgb(152, 237, 83) 0%, rgb(87, 199, 154) 50%,
rgb(42, 155, 129) 100%);
            background-attachment: fixed;
        }
        .container {
            max-width: 400px;
            padding: 20px;
            background-color: white;
            border-radius: 20px;
            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
            margin-bottom: 30px;
        }
        .form-group {
            margin-bottom: 20px;
        }
        .form-group label {
            display: block;
            margin-bottom: 5px;
            font-weight: bold;
```

```
}
.form-group select {
    width: 100%;
    padding: 8px;
    border: 1px solid #ddd;
    border-radius: 20px;
}
.btn {
    display: inline-block;
    padding: 10px 20px;
    background-color: #26A69A;
    color: white;
    border: none;
    border-radius: 30px;
    cursor: pointer;
    font-size: 16px;
    transition: background-color 0.3s;
}
.btn:hover {
    background-color: #2bbbad;
}
.btn-large {
    padding: 12px 24px;
    font-size: 18px;
}
.text-center {
    text-align: center;
    margin: 0;
}
.game-header {
    padding: 10px 10px 30px 10px;
}
.game-header h1 {
    margin-bottom: 2px; /* Увеличиваем расстояние между h1 и p */
    font-weight: 700 !important;
}
.game-header p {
    line-height: 1.3; /* Увеличиваем расстояние между строками в параграфе */
}
.form_radio_group {
    display: flex;
    width: 100%;
    overflow: hidden;
    margin-top: 5px;
}
```

```
        .form_radio_group-item {
            flex: 1;
        }
        .form_radio_group input[type=radio] {
            display: none;
        }
        .form_radio_group label {
            display: block;
            cursor: pointer;
            padding: 8px 5px;
            text-align: center;
            border: 1px solid #ddd;
            border-right: none;
            user-select: none;
            font-weight: normal;
            transition: background-color 0.3s;
        }
        .form_radio_group .form_radio_group-item:first-child label {
            border-radius: 20px 0 0 20px;
        }
        .form_radio_group .form_radio_group-item:last-child label {
            border-radius: 0 20px 20px 0;
            border-right: 1px solid #ddd;
        }
        .form_radio_group input[type=radio]:checked + label {
            background: #26A69A;
            color: white;
        }
        .form_radio_group label:hover {
            background-color: #f0f0f0;
        }
        .form_radio_group input[type=radio]:disabled + label {
            background: #efefef;
            color: #999;
            cursor: not-allowed;
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="game-header">
            <h1 class="text-center">Codenames</h1>
            <p class="text-center">Добро пожаловать в игру! <br> Настройте параметры и
нажмите "Начать игру".</p>
```

```
        <p class="text-center"><a href="/rules" style="color: #26A69A; text-decoration: none;
font-weight: bold;">Правила игры</a></p>
      </div>
      <form action="/create-game" method="get">
        <div class="form-group">
          <label>Первый ход:</label>
          <div class="form_radio_group">
            <div class="form_radio_group-item">
              <input id="first-team-random" type="radio" name="first_team" value="random"
checked>
              <label for="first-team-random">Случайно</label>
            </div>
            <div class="form_radio_group-item">
              <input id="first-team-red" type="radio" name="first_team" value="red">
              <label for="first-team-red">Красные</label>
            </div>
            <div class="form_radio_group-item">
              <input id="first-team-blue" type="radio" name="first_team" value="blue">
              <label for="first-team-blue">Синие</label>
            </div>
          </div>
        </div>

        <div class="form-group">
          <label>Язык слов:</label>
          <div class="form_radio_group">
            <div class="form_radio_group-item">
              <input id="language-russian" type="radio" name="language" value="russian"
checked>
              <label for="language-russian">Русский</label>
            </div>
            <div class="form_radio_group-item">
              <input id="language-english" type="radio" name="language" value="english">
              <label for="language-english">Английский</label>
            </div>
          </div>
        </div>

        <div class="text-center" style="margin-top: 30px;">
          <button type="submit" class="btn btn-large">Начать игру</button>
        </div>
      </form>
    </div>
</body>
</html>
```