Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования
Кафедра инженерной психологии и эргономики
Дисциплина: Компьютерные системы и сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему:

СЕТЕВАЯ КОМПЬЮТЕРНАЯ ИГРА-КВЕСТ «АЛИСА В СТРАНЕ ЧУДЕС»

БГУИР КП 6-05-0612-01-028 ПЗ

Студент Касперец Е.А.

Руководитель Болтак С.В.

Минск 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире компьютерные игры стали неотъемлемой частью индустрии развлечений и образования. Особый интерес представляют многопользовательские игры, которые способствуют развитию навыков командной работы и социального взаимодействия. В рамках данной курсовой работы разработана кооперативная многопользовательская игра на основе классического произведения Льюиса Кэрролла "Алиса в Стране чудес".

Актуальность работы обусловлена растущим спросом на кооперативные игры, которые позволяют игрокам взаимодействовать друг с другом для достижения общих целей. Такие игры не только развлекают, но и способствуют развитию навыков коммуникации и совместного решения задач.

Целью данной курсовой работы является разработка многопользовательской игры с использованием архитектуры *peer-to-peer (P2P)* на языке программирования *Python* с применением библиотеки *Pygame*. *Python* – высокоуровневый язык программирования общего назначения с открытым исходным кодом, отличающийся простым и понятным синтаксисом, что делает его идеальным инструментом для разработки различных приложений, включая игры [2]. *Pygame* – библиотека для создания двумерных игр на языке *Python*, предоставляющая набор инструментов для работы с графикой, звуком и пользовательским вводом, основанная на *SDL (Simple DirectMedia Layer)*[3].

Для достижения поставленной цели были определены следующие задачи:

1 разработка концепции игры на основе сюжета «Алисы в Стране чудес»;

2 реализация базовой механики платформера с физикой движения и коллизиями;

3 создание системы взаимодействия между игроками через сетевое соединение;

4 разработка механик кооперативного прохождения, требующих совместных действий игроков;

5 реализация системы диалогов для повествования сюжета;

6 создание визуального оформления игры с использованием спрайтовой графики.

Практическая значимость работы заключается в создании полноценного игрового продукта, демонстрирующего возможности разработки многопользовательских игр на языке *Python*.

Структура работы включает в себя описание архитектуры приложения, реализацию игровой механики, сетевого взаимодействия, а также тестирование и анализ результатов разработки.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Обзор аналогов

При анализе существующих решений в области кооперативных платформеров прежде всего следует отметить игру *«It Takes Two»*, выпущенную в 2021 году (рисунок 1.1). Данный проект демонстрирует высочайший уровень реализации кооперативных механик и качественную графическую составляющую, однако требует значительных системных ресурсов и обязательного приобретения двух копий игры для совместного прохождения.

Рисунок 1.1 Игра *«It Takes Two»*

Говоря о более камерных проектах, стоит упомянуть *«Unravel Two»*, привлекающий внимание своим визуальным стилем и интуитивно понятным управлением (рисунок 1.2). В то же время отсутствие онлайн-мультиплеера и ограниченный набор игровых механик несколько снижают потенциал данного проекта.

Рисунок 1.2 Игра *«Unravel Two»*

Интересным примером работы с двумя персонажами является *«Brothers: A Tale of Two Sons»*, предлагающий эмоциональное повествование и уникальную систему управления (рисунок 1.3). Однако отсутствие полноценного кооперативного режима и небольшая продолжительность игры существенно ограничивают возможности проекта.

Рисунок 1.3 Игра *«Brothers: A Tale of Two Sons»*

Разработанная игра по мотивам «Алисы в Стране чудес» учитывает как достоинства, так и недостатки рассмотренных аналогов. В частности, проект отличается низкими системными требованиями и простотой установки благодаря использованию *Pygame*. Реализация *P2P*-архитектуры позволила создать стабильное сетевое взаимодействие без необходимости использования специальных сервисов. Безусловно, разработанная игра имеет более скромную графическую составляющую по сравнению с коммерческими аналогами, однако успешно реализует ключевые механики кооперативного взаимодействия.

В заключение стоит отметить, что созданная игра, несмотря на определенные ограничения, предлагает уникальное сочетание доступности, функциональности, что выгодно отличает данный проект от существующих аналогов в контексте поставленных задач.

1.2 Постановка задачи

Основной целью курсовой работы является разработка многопользовательской игры-квеста по мотивам "Алисы в Стране чудес" с использованием языка программирования *Python* и библиотеки *Pygame*. Для достижения поставленной цели необходимо решить следующие задачи:

1 Разработать базовую механику платформера.

В рамках данной задачи требуется создать физический движок, обеспечивающий реалистичное перемещение персонажей в двумерном пространстве, включая механику прыжков с учетом гравитации. Необходимо реализовать систему обнаружения и обработки столкновений между

игровыми объектами, включая взаимодействие с платформами различных типов. Также требуется разработать систему анимации персонажей, включающую смену спрайтов при различных действиях: ходьбе, прыжках и бездействии.

2 Реализовать сетевое взаимодействие.

Необходимо спроектировать и реализовать архитектуру *peer-to-peer* для обеспечения прямого соединения между двумя игроками. Требуется создать механизм синхронизации состояния игрового мира, включая положение персонажей, собранные предметы и состояние игровых объектов. Важной частью является разработка системы обработки сетевых событий с учетом возможных задержек и потерь пакетов.

3 Создать игровые механики.

В игре должна присутствовать интерактивная система диалогов между персонажами, влияющая на развитие сюжета. Требуется разработать набор головоломок, решение которых возможно только при взаимодействии двух игроков. Необходимо реализовать систему собираемых предметов, влияющих на прогресс прохождения игры и открывающих новые возможности для игроков.

4 Разработать пользовательский интерфейс.

Интерфейс должен наглядно отображать текущий прогресс прохождения игры, включая количество собранных предметов и достигнутые цели. Требуется создать систему отображения диалоговых окон с поддержкой выбора вариантов ответа. Необходимо реализовать контекстные подсказки, помогающие игрокам освоить механику игры и понять необходимые действия для продвижения по сюжету.

Технические требования к реализации:

– использование *Python* 3.13 и *Pygame*;

– модульная архитектура приложения;

– оптимизация производительности;

– кроссплатформенность.

Критерии успешности проекта определяются стабильной работой сетевого взаимодействия, корректным функционированием игровых механик и отсутствием критических ошибок в работе приложения.

2 ПРОЕКТИРОВАНИЕ ЗАДАЧИ.

2.1 Структура программы

Программный комплекс разработан на языке *Python* с применением объектно-ориентированного подхода. Архитектура приложения организована по модульному принципу, что обеспечивает масштабируемость и удобство поддержки кода. Основные компоненты системы представлены в таблице 1.

Таблица 1 – Основные модули программного комплекса

Модуль	Функционал	Ключевые классы
game.py	Центральный игровой цикл, управление подсистемами	*Game*
story_screen.py	Управление сюжетными сценами и диалогами	*StoryScreen*
network_manager.py	Сетевое взаимодействие между игроками	*NetworkManager*
physics.py	Физическая модель и обработка коллизий	*PhysicsEngine*
resource_manager.py	Загрузка и управление игровыми ресурсами	*ResourceLoader*

Иерархия классов игровых объектов организована согласно принципам объектно-ориентированного программирования, как показано в таблице 2.

Таблица 2 – Иерархия классов игровых объектов

Базовый класс	Производные классы	Реализуемый функционал
GameObject	*Player, Platform*	Базовые свойства всех игровых объектов
Platform	*MovingPlatform*	Статические и динамические платформы
SpriteSheet	*CharacterSprite*	Управление анимацией персонажей

Физическая модель игры включает следующие параметры, представленные в таблице 3.

Таблица 3 – Параметры физической модели

Параметр	Значение	Назначение
GRAVITY	0.5	Сила гравитационного воздействия
JUMP_FORCE	-12	Импульс при прыжке
MOVE_SPEED	6	Базовая скорость перемещения
FRICTION	0.8	Коэффициент трения

Сетевая подсистема реализует *peer-to-peer* архитектуру с использованием *UDP* протокола. Основные характеристики сетевого взаимодействия представлены в таблице 4.

Таблица 4 – Параметры сетевого взаимодействия

Компонент	Реализация	Особенности
Транспортный протокол	*UDP*	Минимизация задержек
Формат данных	Сериализация *pickle*	Поддержка сложных структур
Порт	5000 - для хоста (первого игрока); 5001 - для клиента (второго игрока)	Стандартный игровой порт
Частота обновлений	30 пакетов/сек	Оптимальный баланс нагрузки

Система управления ресурсами организована согласно структуре каталогов, представленной в таблице 5.

Таблица 5 – Структура каталогов ресурсов

Каталог	Типы файлов	Назначение
assets/characters/	*PNG, JSON*	Спрайты персонажей и их анимации
assets/tiles/	*PNG*	Текстуры окружения
assets/gui/	*SVG, PNG*	Элементы пользовательского интерфейса
assets/sounds/	*OGG, WAV*	Звуковые эффекты и музыка

Для достижения оптимальной производительности в разработанной системе применяется комплексный подход, включающий различные методы оптимизации на всех уровнях приложения от управления ресурсами до рендеринга графики. Особое внимание уделяется минимизации нагрузки на процессор и графический ускоритель, эффективному использованию памяти и оптимизации сетевого взаимодействия. Комплекс организационных мер представлен в таблице 6.

Таблица 6 – Методы оптимизации производительности

Технология	Реализация	Эффект
Пространственное хеширование	Квадродерево для коллизий	Ускорение обработки столкновений
Спрайтовые атласы	Объединение текстур	Снижение числа переключений контекста
Кэширование ресурсов	*LRU*-кэш	Уменьшение времени загрузки
Пакетная отрисовка	Группировка объектов	Снижение нагрузки на *GPU*

Обработка исключений реализована для всех критических подсистем, как показано в таблице 7.

Таблица 7 – Система обработки исключений

Подсистема	Типы исключений	Действия при ошибке
Сетевая	*socket.error*	Повторное подключение
Графическая	*pygame.error*	Переключение на резервный рендерер
Файловая	*IOError*	Использование стандартных ресурсов
Физическая	*ArithmeticError*	Сброс состояния физического движка

Представленная архитектура обеспечивает стабильную работу приложения, позволяя легко расширять функционал за счет модульной структуры и четкого разделения ответственности между компонентами системы.

2.2 Проектирование интерфейса программного средства

Проектирование интерфейса программного средства происходит с помощью библиотеки *Pygame*. *Pygame* – это кроссплатформенная библиотека для разработки компьютерных игр и мультимедийных приложений на языке *Python* [4].

2.2.1 Основной игровой интерфейс

Начальный экран отображается при запуске игры и представляет собой анимированную сцену с историей, вводящей игроков в сюжет. На экране последовательно появляются текстовые сообщения, сопровождаемые визуальными эффектами (рисунок 2.1).

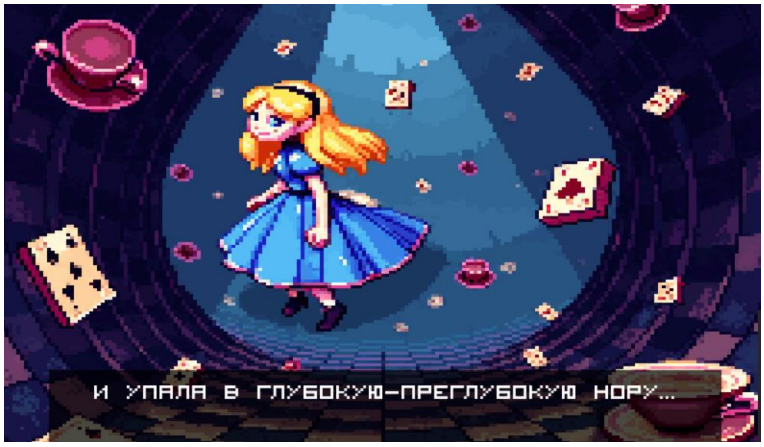


Рисунок 2.1 – Начальный экран игры

Основной игровой интерфейс разработан с учетом необходимости отображения важной игровой информации без перегрузки экрана. В верхней части экрана располагаются индикаторы собранных предметов, отображающие количество найденных ключей и зелий (рисунок 2.3). Для улучшения навигации реализована система подсказок, которая появляется в контекстно-зависимых ситуациях (рисунок 2.4). Все элементы интерфейса оформлены в едином стиле, соответствующем общей тематике игры.

Рисунок 2.3 – Игровой интерфейс

Рисунок 2.4 – Система подсказок

При разработке диалоговой системы основное внимание уделено созданию интуитивно понятного интерфейса взаимодействия между игровыми персонажами. Диалоговое окно активируется при вхождении игрока в зону взаимодействия с персонажем.

Текстовое поле диалога размещается в верхней части экрана на полупрозрачном фоне, что обеспечивает хорошую читаемость текста без потери визуального контакта с игровым миром. Для улучшения восприятия реализована анимация постепенного появления текста.

Варианты ответов располагаются под основным текстом диалога и визуально выделяются при наведении курсора. Для обеспечения комфортного взаимодействия каждый вариант ответа имеет достаточную область выбора и четкое визуальное оформление.

Система адаптирует диалоги в зависимости от роли игрока. Диалоги Алисы (рисунок 2.5) отражают характер исследователя.

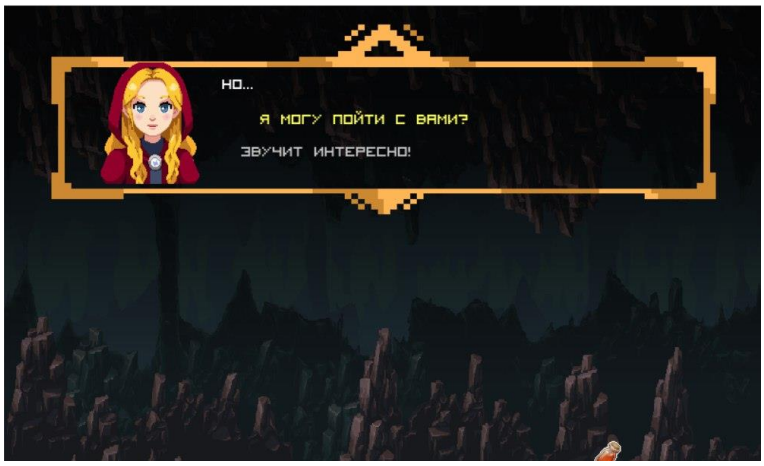


Рисунок 2.5 – Диалог Алисы

Диалоги Кролика (рисунок 2.6) подчеркивают спешку и беспокойство персонажа.

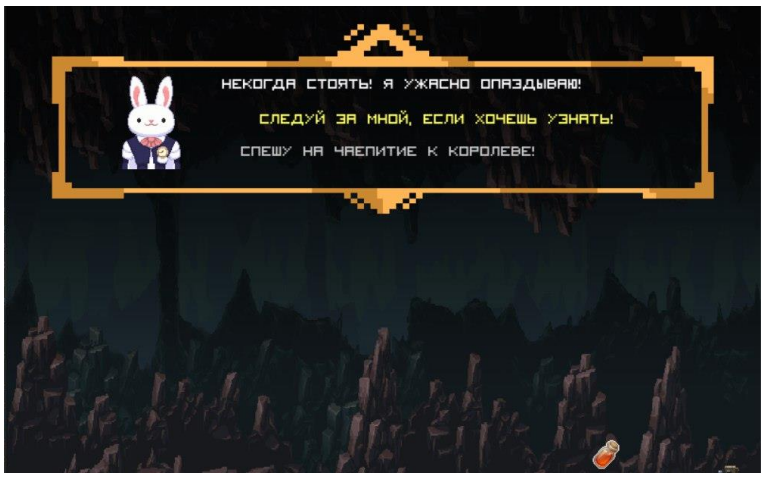


Рисунок 2.6 – Диалог Кролика

2.2.2 Процесс игры

Игровой процесс представляет собой кооперативное прохождение уровня двумя игроками, где каждый управляет уникальным персонажем – Алисой или Кроликом. Взаимодействие между игроками реализовано через сетевой протокол, обеспечивающий синхронизацию всех игровых событий и состояний персонажей в реальном времени.

Каждый персонаж обладает уникальными характеристиками движения. Алиса перемещается со стандартной скоростью *(ALICE_MOVE_SPEED = 6)* и имеет умеренную высоту прыжка *(ALICE_JUMP_FORCE = -15)*. Белый Кролик, в свою очередь, двигается быстрее *(RABBIT_MOVE_SPEED = 8)* и прыгает выше *(RABBIT_JUMP_FORCE = -20)*. Эти различия создают необходимость координации действий между игроками для преодоления препятствий и решения головоломок.

Система сохранения прогресса работает в реальном времени, синхронизируя между игроками следующие параметры: количество собранных предметов (ключей и зелий), состояние диалогов с персонажами, положение игроков на уровне, состояние активированных механизмов и платформ. Это обеспечивает согласованность игрового опыта для обоих участников.

Для успешного завершения уровня игрокам необходимо собрать все ключи и зелья, разбросанные по игровому миру, решить совместные головоломки, требующие координации действий, и достичь финальной точки уровня одновременно. Прогресс прохождения наглядно отображается через изменение игрового окружения и систему визуальных подсказок.

Физическая модель игры учитывает гравитацию *(GRAVITY = 0.5)* и коллизии с платформами, что создает реалистичное ощущение перемещения в игровом мире. Размеры игрового мира *(WORLD_WIDTH = 3000, WORLD_HEIGHT = 1000)* обеспечивают достаточное пространство для исследования и решения головоломок, при этом камера следует за игроками, обеспечивая оптимальный обзор игрового пространства.

Таким образом, процесс игры представляет собой комплексное взаимодействие между игроками, требующее координации действий и совместного решения задач, что создает увлекательный кооперативный игровой опыт в атмосфере сказочного мира «Алисы в Стране чудес».

2.3 Проектирование функционала программного средства

Для корректной работы программы реализован комплекс ключевых функциональных компонентов. Основными функциональными блоками являются:

– система синхронизации состояний игроков;

– обработка физики и коллизий;

– управление игровыми событиями;

– система диалогов и взаимодействий.

Алгоритм синхронизации состояний игроков реализован через сетевой протокол UDP и представлен на рисунке 2.7 [5].

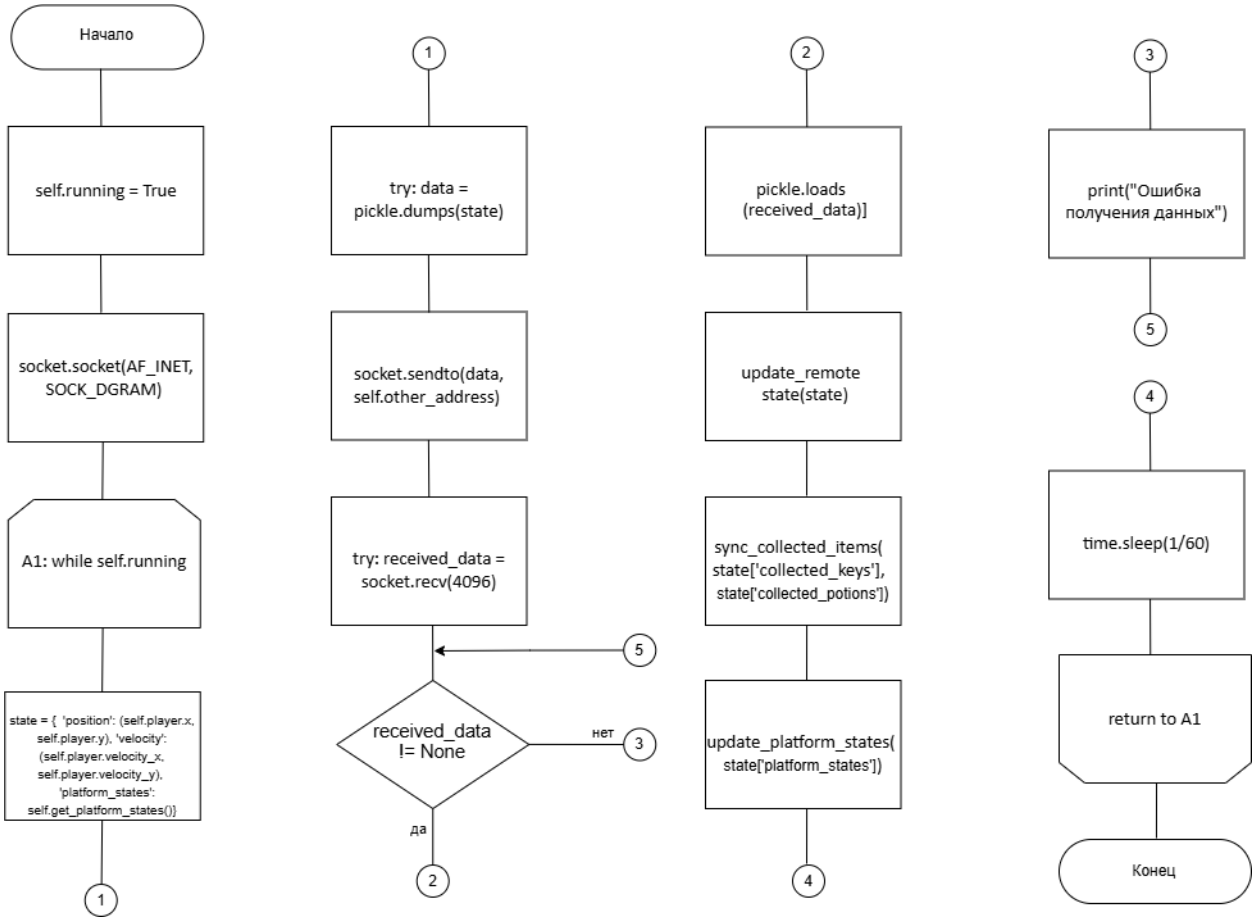


Рисунок 2.7 – Блок-схема алгоритма синхронизации состояний

Синхронизация состояний включает следующие этапы:

1 Формирование пакета данных о состоянии игрока:

```
game_state = {
    "type": "game_state",
    "timestamp": current_time,
    "sequence_number": self.sequence_number,
    "player": {
        "x": self.my_player.x,
        "y": self.my_player.y,
        "state": self.my_player.state,
        "is_jumping": self.my_player.is_jumping
    },
    "collected_items": {
        "keys": [i for i, key in enumerate(self.animated_keys) if key.collected],
        "potions": [i for i, potion in enumerate(self.potions) if potion.collected]
    }
}
```

2 Отправка данных другому игроку.

3 Получение и обработка входящих данных.

4 Применение полученных состояний [6].

Обработка физики включает [7]:

1 Применение гравитации.

```
GRAVITY = 0.5
self.velocity_y += GRAVITY
self.y += self.velocity_y
```

2 Проверка коллизий с платформами:

```
def check_collision(self, platforms):
    for platform in platforms:
        if self.rect.colliderect(platform.rect):
            if self.velocity_y > 0:
                self.rect.bottom = platform.rect.top
                self.velocity_y = 0
                self.is_jumping = False
            elif self.velocity_y < 0:
                self.rect.top = platform.rect.bottom
                self.velocity_y = 0
```

3 Обработка движения персонажа.

4 Обновление состояния анимаций. Система управления игровыми событиями представлена на рисунке 2.8 [8]:

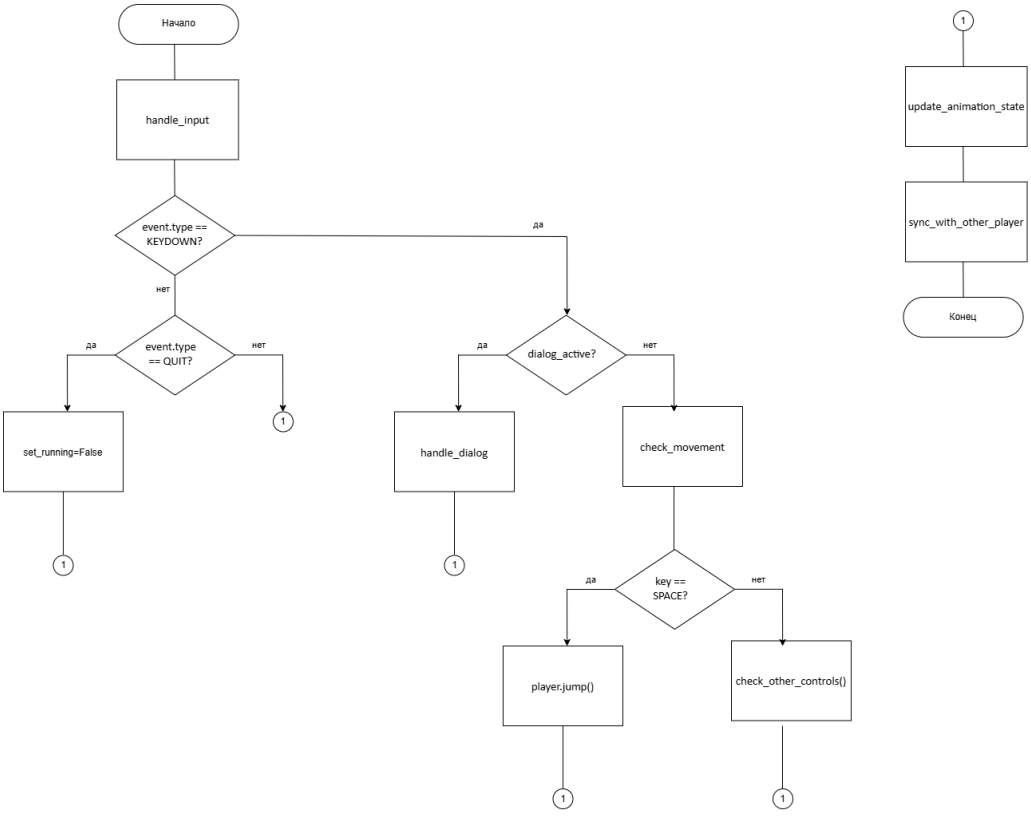


Рисунок 2.8 – Блок-схема системы игровых событий]

Алгоритм проверки условия победы представлен на рисунке 2.9

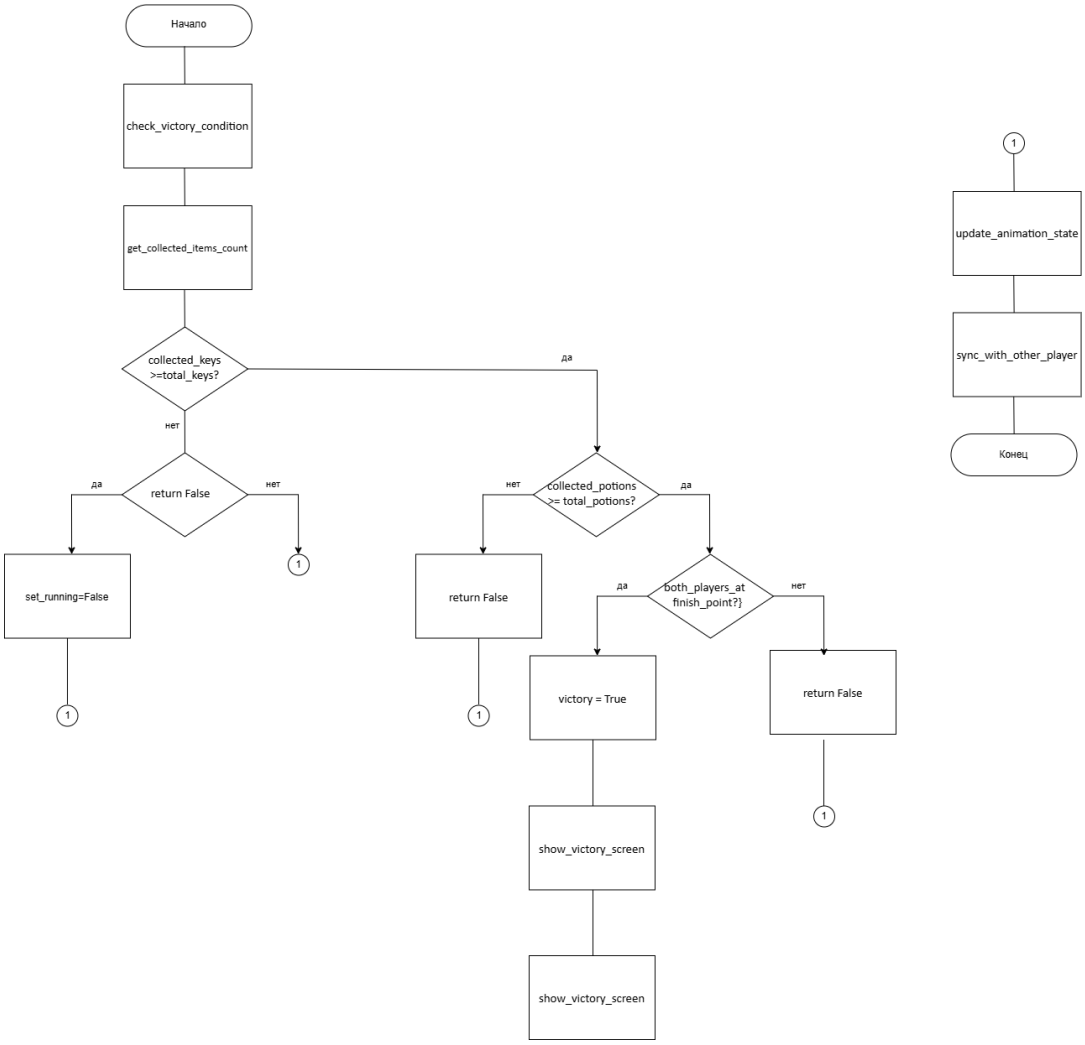


Рисунок 2.9 – Блок-схема проверки условий победы

Диалоговая система активируется при выполнении определенных условий, например, при приближении к игровому персонажу. При активации система загружает соответствующий диалог из *JSON*-файла и отображает его на экране с плавной анимацией текста. Если диалог предусматривает выбор, игрокам предоставляются варианты ответов. Выбор одного игрока автоматически синхронизируется со вторым игроком, обеспечивая целостность повествования для обоих участников.

Таким образом, функционал программного средства представляет собой сложную систему взаимосвязанных компонентов, обеспечивающих многопользовательское взаимодействие в реальном времени. Каждый компонент оптимизирован для обеспечения плавной работы игры и минимизации задержек при сетевом взаимодействии.

3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

3.1 Взаимодействие клиентов в сети

Взаимодействие между игроками реализовано по принципу *peer-to-peer (P2P)* с использованием протокола *UDP*. Данный подход был выбран для обеспечения минимальных задержек при передаче данных, что является критически важным фактором для синхронизации состояний игроков в реальном времени. В отличие от *TCP*, протокол *UDP* не требует установления постоянного соединения и подтверждения доставки пакетов, что значительно снижает сетевую задержку [5].

Система обмена данными построена на использовании сетевых сокетов, где каждый клиент одновременно выполняет функции как отправителя, так и получателя данных. Для обеспечения надежности передачи реализован механизм подтверждения критически важных игровых событий. Структура передаваемых данных представлена в формате *JSON*, что обеспечивает универсальность и легкость обработки информации [9]. Передаваемые данные включают:

```
game_state = {
    "type": "game_state",
    "timestamp": current_time,
    "sequence_number": self.sequence_number,
    "player": {
        "x": self.my_player.x,
        "y": self.my_player.y,
        "state": self.my_player.state
    },
    "collected_items": {
        "keys": [i for i, key in enumerate(self.animated_keys) if key.collected],
        "potions": [i for i, potion in enumerate(self.potions) if potion.collected]
    }
}
```

Для оптимизации сетевого трафика применяется система приоритетов передачи данных:

– высокий приоритет: позиции игроков, состояния прыжков, коллизии;

– средний приоритет: анимации, собранные предметы;

– низкий приоритет: декоративные элементы, фоновые эффекты.

3.2 Система подключения игроков

Процесс инициализации сетевого соединения реализован через создание *UDP*-сокета с оптимизированными параметрами для обеспечения эффективной передачи данных в режиме реального времени [10]. Рассмотрим детально каждый этап инициализации:

1 Создание сокета.

self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

Здесь используется *UDP*-протокол *(SOCK_DGRAM)*, который обеспечивает быструю передачу данных без установления постоянного соединения. Параметр *AF_INET* указывает на использование *IPv4* для сетевого взаимодействия.

2 Настройка параметров сокета:

self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

Опция *SO_REUSEADDR* позволяет повторно использовать адрес сокета, что особенно важно при перезапуске игры или восстановлении соединения. Это предотвращает блокировку порта после некорректного завершения программы.

3 Оптимизация буферов:

self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF, 65536)
self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_SNDBUF, 65536)

Увеличение размеров буферов приема *(SO_RCVBUF)* и отправки *(SO_SNDBUF)* до 64 КБ позволяет:

– минимизировать потери пакетов при интенсивном обмене данными;

– снизить вероятность переполнения буферов;

– оптимизировать производительность сетевого взаимодействия.

4 Распределение ролей и привязка портов:

```
if self.is_host:
    self.socket.bind((host, 5000))
    self.other_address = (host, 5001)
    print(f"Сервер запущен на {host}:5000")
else:
    self.socket.bind((host, 5001))
    self.other_address = (host, 5000)
    print(f"Клиент подключен к {host}:5000")
```

В данном блоке происходит:

– определение роли участника (хост или клиент);

– привязка сокета к конкретному порту (5000 для хоста, 5001 для клиента);

– сохранение адреса второго участника для последующей коммуникации;

– вывод информационного сообщения о статусе подключения.

5 Обработка ошибок:

```
try:
    # Код инициализации
    self.socket_active = True
    return True
except Exception as e:
    print(f"Ошибка при настройке сети: {e}")
    self.socket_active = False
```

```
    return False
```

Система предусматривает обработку возможных ошибок при инициализации:

– занятость порта;

– отсутствие сетевого подключения;

– системные ошибки;

– некорректные параметры сети.

Такая структура инициализации обеспечивает: надежное установление соединения между игроками, оптимальную настройку сетевых параметров, корректную обработку ошибок, возможность восстановления соединения при сбоях.

После успешной инициализации сетевого соединения система готова к обмену игровыми данными и синхронизации состояний между участниками, что является основой для реализации многопользовательского режима игры.

Процесс подключения второго игрока реализован как многоступенчатая система, обеспечивающая надежное установление соединения и синхронизацию игровых состояний.

1 Установление UDP-соединения:

```
def setup_network(self, host):
    try:
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

        # Оптимизация буферов для улучшения производительности
        self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF, 65536)
        self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_SNDBUF, 65536)

        # Настройка неблокирующего режима
        self.socket.setblocking(False)

        if self.is_host:
            self.socket.bind((host, 5000))
            self.other_address = (host, 5001)
        else:
            self.socket.bind((host, 5001))
            self.other_address = (host, 5000)

        self.socket_active = True
        return True
    except Exception as e:
        print(f"Ошибка при настройке сети: {e}")
        return False
```

2 Обмен начальными параметрами:

```
def create_players(self):
    start_x = 100
```

```
    ground_level = WORLD_HEIGHT - PLATFORM_HEIGHT - 200

    if self.is_host:
        self.my_player = Player(start_x, ground_level, "alice")
        self.other_player = Player(start_x + 100, ground_level, "rabbit")
    else:
        self.my_player = Player(start_x + 100, ground_level, "rabbit")
        self.other_player = Player(start_x, ground_level, "alice")
```

3 Синхронизация состояний игрового мира осуществляется через:

```
def send_network_update(self):
    if not self.socket_active:
        return

    try:
        current_time = time.time()
        if current_time - self.last_network_update < self.network_update_rate:
            return

        self.last_network_update = current_time
        game_state = {
            "type": "game_state",
            "timestamp": current_time,
            "sequence_number": self.sequence_number,
            "player": self.create_player_state(),
            "collected_items": self.get_collected_items_state()
        }

        data = pickle.dumps(game_state)
        self.socket.sendto(data, self.other_address)
        self.sequence_number += 1

    except Exception as e:
        print(f"Ошибка при отправке данных: {e}")
        self.socket_active = False
```

Включает передачу:

– параметров игрового мира (размеры, физические константы);

– начальных позиций персонажей;

– состояний игровых объектов;

– системных настроек.

4 Система обработки подключений.

Реализована проверка доступности хоста:

```
def check_server_availability(self):
    try:
        test_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        test_socket.settimeout(2.0)
```

```
            test_socket.sendto(b"ping", self.other_address)
            data, addr = test_socket.recvfrom(1024)
            return True
        except:
            return False
        finally:
            test_socket.close()
```

Также организована валидация параметров подключения:

```
def validate_connection(self):
    if not self.socket_active:
        return False

    try:
        # Проверка корректности адреса
        if not self.other_address[0] or not self.other_address[1]:
            raise ValueError("Некорректный адрес подключения")

        # Проверка доступности порта
        if self.is_host and not self.is_port_available(5000):
            raise ValueError("Порт 5000 занят")

        # Проверка сетевых параметров
        if not self.validate_network_parameters():
            raise ValueError("Некорректные сетевые параметры")

        return True
    except Exception as e:
        print(f"Ошибка валидации: {e}")
        return False
```

Продумана обработка сетевых ошибок:

```
def handle_network_error(self, error):
    try:
        if isinstance(error, socket.timeout):
            print("Превышено время ожидания соединения")
            self.attempt_reconnect()
        elif isinstance(error, ConnectionResetError):
            print("Соединение сброшено")
            self.handle_connection_reset()
        elif isinstance(error, OSError):
            print(f"Системная ошибка: {error}")
            self.handle_system_error()
        else:
            print(f"Неизвестная ошибка: {error}")
            self.emergency_shutdown()
    except Exception as e:
```

```
        print(f"Ошибка при обработке сетевой ошибки: {e}")
```

При разрывах происходит восстановление соединения:

```
def attempt_reconnect(self):
    max_attempts = 5
    attempt = 0

    while attempt < max_attempts and not self.socket_active:
        try:
            print(f"Попытка переподключения {attempt + 1}/{max_attempts}")

            # Закрываем старое соединение
            if hasattr(self, 'socket'):
                self.socket.close()

            # Создаем новое соединение
            success = self.setup_network(self.other_address[0])

            if success:
                print("Переподключение успешно")
                self.socket_active = True
                self.synchronize_game_state()
                break

        except Exception as e:
            print(f"Ошибка при переподключении: {e}")

        attempt += 1
        time.sleep(1)  # Пауза между попытками
```

Данная система обеспечивает надежное сетевое взаимодействие между игроками, способна обрабатывать различные сетевые ошибки и автоматически восстанавливать соединение при разрывах, что критически важно для обеспечения непрерывности игрового процесса.

3.3 Завершение игры и обработка сетевых исключений

Процесс завершения игры и обработка ошибок реализованы в основном игровом модуле и включают несколько ключевых компонентов.

Завершение игрового процесса происходит при выполнении определенных условий, которые проверяются методом *check_victory_condition()*. Основными условиями победы являются:

– сбор всех ключей на уровне;

– получение необходимых зелий;

– достижение обоими игроками конечной точки.

Реализация проверки:

```
def check_victory_condition(self):
    if (len(self.collected_keys) >= len(self.animated_keys) and
        len(self.collected_potions) >= len(self.potions)):
        return True
```

При завершении игры система последовательно выполняет следующие действия:

– останавливает игровой цикл;

– отключает пользовательский ввод;

– отображает экран победы с результатами;

– сохраняет прогресс игроков.

Визуальное оформление завершения реализовано через специальный метод:

```
def draw_victory_screen(self, screen):
    victory_text = self.font.render("Добро пожаловать в страну чудес!", True, (255, 255, 255))
```

В игре реализована многоуровневая система обработки ошибок:

1 Ошибки загрузки ресурсов:

```
def load_texture(self):
    try:
        self.texture = pygame.image.load(self.texture_path).convert_alpha()
    except pygame.error as e:
        print(f"Ошибка загрузки текстуры: {e}")
```

2 Сетевые ошибки

– обработка разрыва соединения;

– восстановление синхронизации;

– сохранение состояния игры.

3 Игровые ошибки:

– проверка корректности позиций объектов;

– валидация игровых действий;

– предотвращение невалидных состояний.

При возникновении критических ошибок система:

– сохраняет текущее состояние игры;

– логирует информацию об ошибке;

– пытается восстановить работоспособность;

– при необходимости корректно завершает игровую сессию.

Такая комплексная система обработки завершения игры и ошибок обеспечивает стабильную работу приложения и комфортный пользовательский опыт даже в нестандартных ситуациях.

4 ТЕСТИРОВАНИЕ

В процессе работы кооперативной игры могут возникать различные исключительные ситуации, требующие корректной обработки для обеспечения стабильной работы. Основной фокус системы обработки ошибок направлен на сетевое взаимодействие между двумя игроками (Алисой и Кроликом) и синхронизацию их состояний в игровом мире.

Ключевой исключительной ситуацией является отключение одного из игроков во время игровой сессии. При потере соединения система определяет текущее состояние игры и предпринимает соответствующие действия. Если отключение происходит во время активной игровой сессии, второй игрок получает уведомление о разрыве соединения, и игра приостанавливается до восстановления связи:

```
def receive_data(self):
    try:
        data = self.socket.recv(4096)
        return pickle.loads(data)
    except (socket.error, pickle.UnpicklingError) as e:
```

Система обрабатывает ситуации с загрузкой игровых ресурсов. При отсутствии или повреждении файлов спрайтов, текстур или звуковых эффектов, программа создает временные заглушки для отсутствующих ресурсов, что позволяет игре продолжать работу:

```
def load_texture(self):
    try:
        self.texture = pygame.image.load(self.texture_path).convert_alpha()
    except pygame.error as e:
        print(f"Ошибка загрузки текстуры: {e}")
```

На рисунке 4.1 представлена корректная загрузка текстур, на рисунке 4.2 показана временная заглушка, если файл по какой-либо причине не загружен или отсутствует, чтобы обеспечить запуск игры.

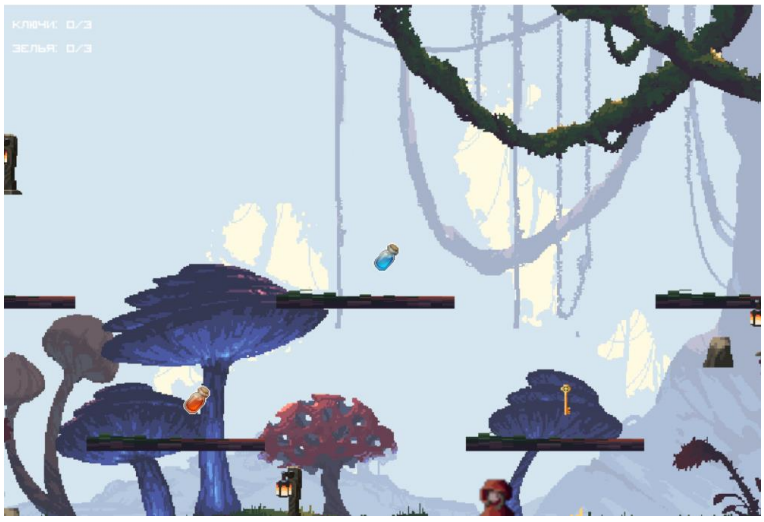


Рисунок 4.1 – Корректное отображение текстуры фона

Рисунок 4.2 – Использование заглушки

Особое внимание уделяется синхронизации состояний игроков при взаимодействии с игровыми объектами. При сборе ключей, активации платформ или взаимодействии с механизмами система проверяет корректность передачи данных между игроками. В случае рассинхронизации происходит автоматическое восстановление согласованного состояния игрового мира.

При проблемах с сетевым подключением или внезапном разрыве соединения система пытается восстановить связь, сохраняя текущее состояние игры.

Система также обрабатывает коллизии и физические взаимодействия. При возникновении ошибок в расчетах позиций персонажей или объектов происходит корректировка их положения для предотвращения застревания или прохождения сквозь твердые поверхности. Это особенно важно при взаимодействии с движущимися платформами и механическими элементами уровнях [11].

В диалоговой системе предусмотрена обработка ошибок при загрузке текстовых данных и синхронизации диалогов между игроками. При возникновении проблем с загрузкой диалогов система предоставляет базовый набор реплик, позволяющий продолжить игровой процесс.

Все исключительные ситуации сопровождаются информативными сообщениями для игроков, объясняющими возникшую проблему и предлагающими возможные действия для её решения.

Таким образом, система обработки исключительных ситуаций обеспечивает стабильную работу игры и комфортный пользовательский опыт даже при возникновении технических проблем. Каждая исключительная ситуация имеет четкий сценарий обработки и сопровождается понятной обратной связью для пользователей.

5 ПРИМЕНЕНИЕ ПРОГРАММЫ

Руководство пользователя предназначено для помощи пользователям в освоении интерфейса игры «Мафия», понимании элементов управления и выполнении основных игровых функций, таких как подключение к лобби, голосование, использование способностей ролей, и других действий.

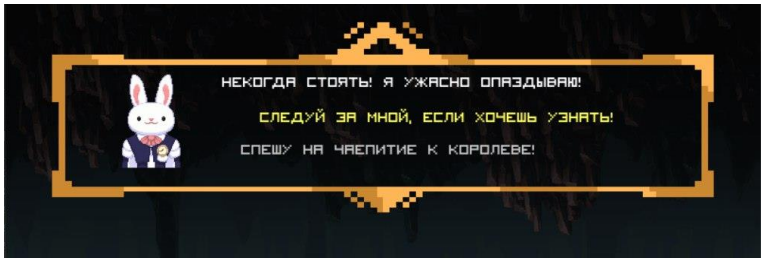
5.1 Руководство пользователя

Разработанное программное средство представляет собой кооперативную игру по мотивам «Алисы в Стране чудес», позволяющую двум игрокам совместно проходить уровни, управляя персонажами Алисы и Кролика. При запуске игры пользователи попадают на начальный экран, где отображается анимированная сцена с историей, погружающей в атмосферу сказочного мира. Текстовые сообщения появляются последовательно, сопровождаясь визуальными эффектами, для продолжения необходимо нажать клавишу *ENTER*.

Система управления персонажами реализована через стандартные клавиши: передвижение осуществляется с помощью *A/D* или стрелок *ВЛЕВО/ВПРАВО*, для прыжка используется клавиша *ПРОБЕЛ*, взаимодействие с объектами и диалоги активируются клавишей *E*, а доступ к меню паузы осуществляется через *ESC* [12]. Персонажи обладают различными характеристиками движения – Алиса перемещается более плавно и имеет умеренную высоту прыжка, в то время как Кролик отличается повышенной скоростью передвижения и способностью совершать высокие прыжки [13].

Сетевое взаимодействие построено по принципу *peer-to-peer*, где один из игроков создает игровую сессию, выбирая роль Алисы, после чего второй игрок может подключиться, указав *IP*-адрес хоста. После успешного соединения оба участника видят экран готовности к началу игры. В процессе прохождения уровня игровой интерфейс отображает важную информацию: количество собранных предметов, индикаторы прогресса, подсказки по управлению и статус сетевого подключения.

При встрече с игровыми персонажами активируется система диалогов (рисунок 5.1).



Игрокам предоставляется выбор вариантов ответа, влияющих на развитие сюжета. Выбор осуществляется с помощью клавиш *ВВЕРХ/ВНИЗ* и подтверждается нажатием *ENTER*.

Игровой процесс включает в себя сбор ключей и зелий, необходимых для продвижения по уровню, активацию различных механизмов и платформ для открытия новых путей, а также взаимодействие между игроками для решения головоломок.

Для успешного прохождения уровня игрокам необходимо координировать свои действия, используя уникальные способности каждого персонажа. Алиса может активировать определенные платформы и механизмы, недоступные Кролику, в то время как Кролик благодаря высоким прыжкам имеет доступ к зонам, которые не может достичь Алиса. Некоторые игровые головоломки специально спроектированы таким образом, что требуют одновременных действий обоих игроков для их решения [16].

При возникновении сетевых проблем система автоматически пытается восстановить соединение. Уровень считается пройденным при выполнении условий победы: сбор всех необходимых предметов и достижение конечной точки обоими игроками.

Таким образом, разработанное программное средство предоставляет пользователям интуитивно понятный интерфейс и увлекательные игровые механики, позволяющие двум игрокам совместно исследовать красочный мир игры и решать интересные головоломки в атмосфере сказочного приключения.

ЗАКЛЮЧЕНИЕ

В процессе выполнения курсовой работы была разработана кооперативная игра по мотивам «Алисы в Стране чудес» с использованием современных технологий разработки игр. Основной технологической базой выступила библиотека *Pygame* для создания игровой графики и физики, а также реализовано *peer-to-peer* сетевое взаимодействие для обеспечения многопользовательского режима.

В рамках разработанного программного средства игроки могут совместно проходить уровни, управляя уникальными персонажами - Алисой и Кроликом, каждый из которых обладает собственными характеристиками и способностями. Разработанный пользовательский интерфейс обеспечивает интуитивное управление и погружение в атмосферу сказочного мира.

К основным преимуществам разработанной игры можно отнести реализацию прямого сетевого соединения между игроками, что обеспечивает минимальные задержки при передаче данных, а также продуманную систему физики и коллизий, позволяющую создавать интересные игровые ситуации. Особого внимания заслуживает реализация кооперативных механик, требующих слаженного взаимодействия игроков для решения головоломок и преодоления препятствий.

Среди технических достоинств стоит отметить модульную архитектуру кода, которая разделяет игровую логику, сетевое взаимодействие и пользовательский интерфейс. Это обеспечивает удобство поддержки и возможность дальнейшего расширения функционала. Система обработки исключительных ситуаций гарантирует стабильную работу даже при возникновении сетевых проблем или ошибок.

В качестве направлений дальнейшего развития проекта можно выделить:

– реализацию системы сохранения прогресса;

– добавление новых уровней и игровых механик;

– расширение возможностей взаимодействия между игроками;

– улучшение визуальных эффектов и анимаций.

Анализ существующих аналогов показал, что разработанная игра предлагает уникальный подход к кооперативному прохождению, сочетая классические платформерные механики с необходимостью постоянного взаимодействия между игроками. Реализованное сетевое взаимодействие и система синхронизации состояний обеспечивают комфортный игровой процесс даже при нестабильном соединении.

Таким образом, в результате выполнения курсовой работы создано полноценное программное средство, реализующее все заявленные функциональные требования и предоставляющее пользователям увлекательный игровой опыт. Модульная архитектура и продуманная система обработки ошибок создают надежную основу для дальнейшего развития проекта.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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

[2] Документация языка программирования *Python* [Электронный ресурс]. – Режим доступа: https://www.python.org

[3] Документация библиотеки *Pygame* [Электронный ресурс]. – Режим доступа: https://www.pygame.org/docs/

[4] Руководство по программированию игр на *Python* с *Pygame* [Электронный ресурс]. – Режим доступа: https://realpython.com/pygame-a-primer/

[5] Документация библиотеки *socket* [Электронный ресурс]. – Режим доступа: https://docs.python.org/3/library/socket.html

[6] Документация библиотеки *pickle* для сериализации данных [Электронный ресурс]. – Режим доступа: https://docs.python.org/3/library/pickle.html

[7] Руководство по разработке *2D*-игр на *Python* [Электронный ресурс]. – Режим доступа: https://www.gamedev.net/tutorials/_/technical/game-programming/2d-game-development-with-python-r3021/

[8] Документация по работе с анимацией спрайтов в *Pygame* [Электронный ресурс]. – Режим доступа: https://www.pygame.org/docs/ref/sprite.html

[9] Основы программирования сетевых игр на *Python* / Свейгарт Эл. – М.: Вильямс, 2020. – 400 с.

[10] Разработка игр на *Python*: практическое руководство / Филлипс Д. – СПб.: Питер, 2021. – 512 с.

[11] Документация по обработке коллизий в *Pygame* [Электронный ресурс]. – Режим доступа: https://www.pygame.org/docs/ref/rect.html

[12] Паттерны проектирования в разработке игр / Норг Р. – М.: ДМК Пресс, 2019. – 384 с.

[13] Документация библиотеки *threading* для многопоточности [Электронный ресурс]. – Режим доступа: https://docs.python.org/3/library/threading.html

[14] Руководство по реализации физики в *2D* играх [Электронный ресурс]. – Режим доступа: https://gamedevelopment.tutsplus.com/tutorials/basic-2d-platformer-physics-part-1--cms-25799

[15] Документация по работе с изображениями в *Pygame* [Электронный ресурс]. – Режим доступа: https://www.pygame.org/docs/ref/image.html

[16] Разработка пользовательского интерфейса в играх / Фокс Б. – СПб.: БХВ-Петербург, 2021. – 336 с.

[17] Документация по работе со звуком в *Pygame* [Электронный ресурс]. – Режим доступа: https://www.pygame.org/docs/ref/mixer.html

ПРИЛОЖЕНИЕ А
(обязательное)

Листинг кода

```
import pygame
import socket
import pickle
import threading
import sys
import os
import time
import random
import json
import pygame.font

# Инициализация Pygame
pygame.init()

# Константы
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 600
WORLD_WIDTH = 3000
WORLD_HEIGHT = 1000

# Размеры персонажей
ALICE_WIDTH = 192
ALICE_HEIGHT = 192
RABBIT_WIDTH = 80
RABBIT_HEIGHT = 80
ALICE_OFFSET = 70
RABBIT_OFFSET = 10

# Физика
GRAVITY = 0.5
ALICE_JUMP_FORCE = -12  # Увеличиваем силу прыжка Алисы
RABBIT_JUMP_FORCE = -20
ALICE_MOVE_SPEED = 6
RABBIT_MOVE_SPEED = 8

# Скорости анимации
ALICE_ANIMATION_SPEED = 0.04
RABBIT_IDLE_ANIMATION_SPEED = 0.15
RABBIT_WALK_ANIMATION_SPEED = 0.06

DIRT_BROWN = (65, 50, 35)
DIRT_DARK = (55, 40, 25)
DIRT_LIGHT = (75, 60, 45)
PLATFORM_HEIGHT = 50
SHADOW_HEIGHT = 400
SHADOW_ALPHA = 200

# Константы для диалогов
DIALOG_PADDING = 40
DIALOG_WIDTH = 700
DIALOG_HEIGHT = 200
CHOICE_HEIGHT = 40
CHOICE_PADDING = 10
FONT_SIZE = 24
TEXT_COLOR = (255, 255, 255)
CHOICE_COLOR = (200, 200, 200)
```

```
CHOICE_HOVER_COLOR = (255, 255, 100)
DIALOG_TRIGGER_DISTANCE = 150
DIALOG_PROMPT_COLOR = (255, 255, 100)
CHOICE_INDENT = 20
EXIT_HINT_DURATION = 3.0

# Константы для платформ и препятствий
PLATFORM_COLOR = (139, 69, 19)
ALICE_PLATFORM_COLOR = (100, 149, 237)
RABBIT_PLATFORM_COLOR = (255, 165, 0)
SWITCH_COLOR = (255, 215, 0)
DOOR_COLOR = (160, 82, 45)
COLLECTIBLE_COLOR = (255, 20, 147)

class SpriteSheet:
    def __init__(self, frames, animation_speed, width, height):
        self.frames = frames
        self.frames_count = len(frames)
        self.target_width = width
        self.target_height = height
        self.current_frame = 0
        self.frame_progress = 0.0
        self.last_update = time.time()
        self.animation_speed = animation_speed

    def get_current_frame(self):
        current_time = time.time()
        delta_time = current_time - self.last_update
        self.last_update = current_time

        # Обновляем прогресс анимации
        self.frame_progress += delta_time / self.animation_speed

        # Если достигли следующего кадра
        if self.frame_progress >= 1.0:
            self.current_frame = (self.current_frame + 1) % self.frames_count
            self.frame_progress = 0.0

        return self.frames[self.current_frame]

    def reset_animation(self):
        self.current_frame = 0
        self.frame_progress = 0.0
        self.last_update = time.time()

def load_sprite(character_name, state="idle"):
    # Настройки количества кадров для каждого спрайта
    sprite_config = {
        "alice": {
            "idle": {"frames": 18, "speed": ALICE_ANIMATION_SPEED, "size": (ALICE_WIDTH,
ALICE_HEIGHT)},
            "walk": {"frames": 24, "speed": ALICE_ANIMATION_SPEED, "size": (ALICE_WIDTH,
ALICE_HEIGHT)}
        },
        "rabbit": {
            "idle": {"frames": 2, "speed": RABBIT_IDLE_ANIMATION_SPEED, "size": (RABBIT_WIDTH,
RABBIT_HEIGHT)},
            "walk": {"frames": 4, "speed": RABBIT_WALK_ANIMATION_SPEED, "size": (RABBIT_WIDTH,
RABBIT_HEIGHT)}  # Обновлено до 4 кадров
        }
    }

    base_path = os.path.join("assets", "characters", character_name)
```

```
try:
    config = sprite_config[character_name][state]
    frames = []

    # Загружаем каждый кадр отдельно
    for i in range(1, config["frames"] + 1):
        # Пробуем разные форматы файлов
        possible_filenames = [
            f"{character_name}_{state} ({i}).png",
            f"{character_name}_{state} ({i}).jpg"
        ]

        frame_path = None
        for filename in possible_filenames:
            temp_path = os.path.join(base_path, filename)
            if os.path.exists(temp_path):
                frame_path = temp_path
                break

        if not frame_path:
            print(f"Файл не найден для кадра {i} в {character_name}/{state}")
            continue

        # Загружаем и масштабируем кадр
        try:
            frame = pygame.image.load(frame_path).convert_alpha()
        except pygame.error as e:
            print(f"Ошибка загрузки {frame_path}: {e}")
            continue

        # Определяем размеры для масштабирования с сохранением пропорций
        source_width = frame.get_width()
        source_height = frame.get_height()
        source_ratio = source_width / source_height
        target_ratio = config["size"][0] / config["size"][1]

        if source_ratio > target_ratio:
            new_width = config["size"][0]
            new_height = int(new_width / source_ratio)
        else:
            new_height = config["size"][1]
            new_width = int(new_height * source_ratio)

        # Масштабируем спрайт
        scaled = pygame.transform.smoothscale(frame, (new_width, new_height))

        # Создаем финальную поверхность
        final = pygame.Surface(config["size"], pygame.SRCALPHA)

        # Центрируем спрайт
        x = (config["size"][0] - new_width) // 2
        y = (config["size"][1] - new_height) // 2

        final.blit(scaled, (x, y))
        frames.append(final)

    if not frames:
        raise FileNotFoundError(f"Не найдено кадров для {character_name}/{state}")

    return SpriteSheet(frames, config["speed"], config["size"][0], config["size"][1])
except Exception as e:
    print(f"Ошибка при загрузке спрайтов {character_name}/{state}: {str(e)}")
    # Создаем заглушку при ошибке
```

```
        surface = pygame.Surface(sprite_config[character_name][state]["size"], pygame.SRCALPHA)
        color = (255, 0, 0) if character_name == "alice" else (0, 0, 255)
        pygame.draw.rect(surface, color, surface.get_rect())
        return SpriteSheet([surface], 0.1, surface.get_width(), surface.get_height())

class Camera:
    def __init__(self, width, height):
        self.width = width
        self.height = height
        self.scroll_x = 0
        self.scroll_y = 0
        self.target_scroll_x = 0
        self.target_scroll_y = 0
        self.lerp_speed = 0.08
        self.deadzone_x = 100
        self.deadzone_y = 80

    def update(self, target_x, target_y):
        # Вычисляем центр экрана
        screen_center_x = self.width // 2
        screen_center_y = self.height // 2

        # Определяем целевую позицию с учетом мертвой зоны
        current_view_x = target_x - self.scroll_x
        current_view_y = target_y - self.scroll_y

        # Проверяем, вышел ли персонаж за пределы мертвой зоны
        if abs(current_view_x - screen_center_x) > self.deadzone_x:
            if current_view_x > screen_center_x:
                self.target_scroll_x = target_x - (screen_center_x + self.deadzone_x)
            else:
                self.target_scroll_x = target_x - (screen_center_x - self.deadzone_x)

        if abs(current_view_y - screen_center_y) > self.deadzone_y:
            if current_view_y > screen_center_y:
                self.target_scroll_y = target_y - (screen_center_y + self.deadzone_y)
            else:
                self.target_scroll_y = target_y - (screen_center_y - self.deadzone_y)

        # Плавная интерполяция к целевой позиции
        self.scroll_x += (self.target_scroll_x - self.scroll_x) * self.lerp_speed
        self.scroll_y += (self.target_scroll_y - self.scroll_y) * self.lerp_speed

        # Ограничение камеры границами мира
        self.scroll_x = max(0, min(self.scroll_x, WORLD_WIDTH - self.width))
        self.scroll_y = max(0, min(self.scroll_y, WORLD_HEIGHT - self.height))

    def apply(self, x, y):
        # Округляем значения для избежания дрожания спрайтов
        return round(x - self.scroll_x), round(y - self.scroll_y)

class Player:
    def __init__(self, x, y, character_name):
        self.x = x
        self.y = y
        self.character_name = character_name
        self.vel_y = 0
        self.is_jumping = False
        self.facing_right = True
        self.moving = False
        self.width = ALICE_WIDTH if character_name == "alice" else RABBIT_WIDTH
        self.height = ALICE_HEIGHT if character_name == "alice" else RABBIT_HEIGHT
        self.offset = ALICE_OFFSET if character_name == "alice" else RABBIT_OFFSET
```

```
        self.sprites = {
            "idle": load_sprite(character_name, "idle"),
            "walk": load_sprite(character_name, "walk")
        }
        self.current_state = "idle"
        self.last_state = "idle"
        self.last_update_time = time.time()
        self.state_changed = False

        # Настройки в зависимости от персонажа
        if character_name == "alice":
            self.move_speed = ALICE_MOVE_SPEED
            self.jump_force = ALICE_JUMP_FORCE
        else:
            self.move_speed = RABBIT_MOVE_SPEED
            self.jump_force = RABBIT_JUMP_FORCE

    def move(self, direction):
        current_time = time.time()
        if current_time - self.last_update_time > 1/60:
            old_x = self.x
            new_x = self.x + direction * self.move_speed


            if hasattr(self, '_platforms'):
                new_x = self.check_horizontal_collisions(self._platforms, new_x)

            self.x = new_x
            self.last_update_time = current_time


        if direction != 0:
            self.facing_right = direction > 0
            self.moving = True
            self.current_state = "walk"
        else:
            self.moving = False
            self.current_state = "idle"


        if self.current_state != self.last_state:
            self.last_state = self.current_state
            self.sprites[self.current_state].reset_animation()

    def jump(self):
        if not self.is_jumping:
            self.vel_y = self.jump_force
            self.is_jumping = True

    def update(self, platforms=None):
        self.vel_y += GRAVITY
        old_y = self.y
        old_x = self.x


        self.y += self.vel_y


        player_rect = pygame.Rect(self.x, self.y, self.width - self.offset, self.height - self.offset)


        platform_y = WORLD_HEIGHT - PLATFORM_HEIGHT
        if self.y > platform_y - self.height + self.offset:
```

```
            self.y = platform_y - self.height + self.offset
            if self.vel_y > 0:
                self.vel_y = 0
                self.is_jumping = False


        if platforms:
            for platform in platforms:
                platform_rect = platform.rect
                player_rect = pygame.Rect(self.x, self.y, self.width - self.offset, self.height - self.offset)


                if player_rect.colliderect(platform_rect):

                    if (old_y + self.height - self.offset <= platform.y + 5 and
                        self.vel_y > 0 and
                        self.y + self.height - self.offset > platform.y):
                        self.y = platform.y - self.height + self.offset
                        self.vel_y = 0
                        self.is_jumping = False
                        break

                    elif (old_y >= platform.y + platform.height - 5 and
                          self.vel_y < 0 and
                          self.y < platform.y + platform.height):
                        self.y = platform.y + platform.height
                        self.vel_y = 0
                        break

    def check_horizontal_collisions(self, platforms, new_x):

        if not platforms:
            return new_x

        player_rect = pygame.Rect(new_x, self.y, self.width - self.offset, self.height - self.offset)

        for platform in platforms:
            if player_rect.colliderect(platform.rect):

                if self.x < platform.x:
                    return platform.x - (self.width - self.offset)

                else:
                    return platform.x + platform.width

        return new_x

    def check_collectibles(self, collectibles):

        player_rect = pygame.Rect(self.x, self.y, self.width - self.offset, self.height - self.offset)
        collected_items = []

        for collectible in collectibles:
            if not collectible.collected and player_rect.colliderect(collectible.rect):
                if collectible.collect(self.character_name):
                    collected_items.append(collectible)

        return collected_items

    def check_platform_triggers(self, platforms):

        player_rect = pygame.Rect(self.x, self.y, self.width - self.offset, self.height - self.offset)
        triggered_platforms = []
```

```
        for platform in platforms:
            if (platform.platform_type == "alice_trigger" and
                self.character_name == "alice" and
                player_rect.colliderect(platform.rect)):
                triggered_platforms.append(platform)

        return triggered_platforms

    def draw(self, screen, camera):
        screen_x, screen_y = camera.apply(self.x, self.y)
        if -self.width <= screen_x <= SCREEN_WIDTH and -self.height <= screen_y <= SCREEN_HEIGHT:
            sprite = self.sprites[self.current_state].get_current_frame()
            if not self.facing_right:
                sprite = pygame.transform.flip(sprite, True, False)
            screen.blit(sprite, (screen_x, screen_y))

class DialogSystem:
    def __init__(self):
        pygame.font.init()
        self.font = pygame.font.Font(os.path.join("assets", "fonts", "visitor2.otf"), FONT_SIZE)


        self.dialog_bg = pygame.image.load(os.path.join("assets", "gui", "dialog_box.png"))
        self.dialog_bg = pygame.transform.scale(self.dialog_bg, (DIALOG_WIDTH, DIALOG_HEIGHT))



        try:
            self.alice_portrait = pygame.image.load(os.path.join("assets", "characters", "alice",
"alice_dialog.png")).convert_alpha()
            self.alice_portrait = pygame.transform.scale(self.alice_portrait, (128, 128))
        except:
            self.alice_portrait = None

        try:
            self.rabbit_portrait = pygame.image.load(os.path.join("assets", "characters", "rabbit",
"rabbit_dialog.png")).convert_alpha()
            self.rabbit_portrait = pygame.transform.scale(self.rabbit_portrait, (128, 128))
        except:
            self.rabbit_portrait = None


        with open(os.path.join("assets", "dialogs", "rabbit_dialog.json"), 'r', encoding='utf-8') as f:
            self.dialogs = json.load(f)

        self.reset_state()

    def reset_state(self):

        self.current_dialog = None
        self.current_choices = []
        self.selected_choice = 0
        self.is_active = False
        self.text_alpha = 0
        self.current_text = ""
        self.target_text = ""
        self.text_timer = 0
        self.my_turn = False
        self.current_speaker = None
        self.dialog_ended = False
        self.dialog_completed = False
        self.exit_hint_timer = 0
        self.show_exit_hint = False
```

```
def start_dialog(self, dialog_id, character_name):

    if self.dialog_completed:
        return

    if dialog_id in self.dialogs:
        print(f"Запуск диалога {dialog_id} для {character_name}")
        self.current_dialog = self.dialogs[dialog_id]
        self.current_choices = self.current_dialog.get("choices", [])
        self.selected_choice = 0
        self.is_active = True
        self.text_alpha = 0
        self.current_text = ""
        self.target_text = self.current_dialog["text"]
        self.text_timer = 0
        self.current_speaker = self.current_dialog.get("speaker")
        self.my_turn = (self.current_speaker == character_name)
        self.dialog_ended = False
        print(f"Диалог запущен. Говорящий: {self.current_speaker}, Мой ход: {self.my_turn}")
    else:
        print(f"Ошибка: диалог {dialog_id} не найден")

def complete_dialog(self):

    self.dialog_completed = True
    self.dialog_ended = True
    self.is_active = False
    self.show_exit_hint = True
    self.exit_hint_timer = EXIT_HINT_DURATION


def update(self, dt):
    if self.is_active:

        if len(self.current_text) < len(self.target_text):
            self.text_timer += dt
            if self.text_timer >= 0.02:
                self.text_timer = 0
                self.current_text = self.target_text[:len(self.current_text) + 1]


        if self.text_alpha < 255:
            self.text_alpha = min(255, self.text_alpha + 510 * dt)


    if self.show_exit_hint:
        self.exit_hint_timer -= dt
        if self.exit_hint_timer <= 0:
            self.show_exit_hint = False

def wrap_text(self, text, max_width):

    words = text.split()
    lines = []
    current_line = []

    for word in words:
        test_line = ' '.join(current_line + [word])
        test_surface = self.font.render(test_line, True, TEXT_COLOR)
        if test_surface.get_width() <= max_width:
            current_line.append(word)
        else:
```

```
            if current_line:
                lines.append(' '.join(current_line))
                current_line = [word]
            else:

                lines.append(word)

        if current_line:
            lines.append(' '.join(current_line))

        return lines

    def draw(self, screen, character_name):


        if self.show_exit_hint:
            hint_text = "Теперь нужно найти выход из кроличьей норы!"
            hint_surface = self.font.render(hint_text, True, DIALOG_PROMPT_COLOR)
            hint_x = (SCREEN_WIDTH - hint_surface.get_width()) // 2
            hint_y = SCREEN_HEIGHT // 2


            bg_rect = pygame.Rect(hint_x - 10, hint_y - 5, hint_surface.get_width() + 20, hint_surface.get_height() +
10)
            bg_surface = pygame.Surface((bg_rect.width, bg_rect.height), pygame.SRCALPHA)
            bg_surface.fill((0, 0, 0, 128))
            screen.blit(bg_surface, bg_rect)
            screen.blit(hint_surface, (hint_x, hint_y))
            return

        if not self.is_active or not self.current_dialog or self.dialog_completed:
            return


        dialog_x = (SCREEN_WIDTH - DIALOG_WIDTH) // 2
        dialog_y = 20


        screen.blit(self.dialog_bg, (dialog_x, dialog_y))


        portrait = None
        if self.current_speaker == "alice" and self.alice_portrait:
            portrait = self.alice_portrait
        elif self.current_speaker == "rabbit" and self.rabbit_portrait:
            portrait = self.rabbit_portrait

        if portrait:
            portrait_x = dialog_x + DIALOG_PADDING
            portrait_y = dialog_y + DIALOG_PADDING
            screen.blit(portrait, (portrait_x, portrait_y))
            text_start_x = portrait_x + 140
        else:

            speaker_text = "Алиса" if self.current_speaker == "alice" else "Кролик"
            speaker_surface = self.font.render(speaker_text, True, (255, 255, 100))
            screen.blit(speaker_surface, (dialog_x + DIALOG_PADDING, dialog_y + DIALOG_PADDING))
            text_start_x = dialog_x + DIALOG_PADDING


        text_area_width = DIALOG_WIDTH - (text_start_x - dialog_x) - DIALOG_PADDING
        text_start_y = dialog_y + DIALOG_PADDING + 10
```

```
    lines = self.wrap_text(self.current_text, text_area_width)


    text_y = text_start_y
    max_text_height = DIALOG_HEIGHT - (text_start_y - dialog_y) - DIALOG_PADDING

    for line in lines:
        if text_y + FONT_SIZE > dialog_y + DIALOG_HEIGHT - DIALOG_PADDING:
            break

        text_surface = self.font.render(line, True, TEXT_COLOR)
        text_surface.set_alpha(self.text_alpha)
        screen.blit(text_surface, (text_start_x, text_y))
        text_y += FONT_SIZE + 3


    if (self.current_text == self.target_text and
        self.current_speaker == character_name and
        self.current_choices):

        choices_start_y = text_y + 10

        for i, choice in enumerate(self.current_choices):
            choice_y = choices_start_y + i * (FONT_SIZE + CHOICE_PADDING)


            if choice_y + FONT_SIZE > dialog_y + DIALOG_HEIGHT - DIALOG_PADDING:
                break

            color = CHOICE_HOVER_COLOR if i == self.selected_choice else CHOICE_COLOR
            indent = CHOICE_INDENT * 2 if i == self.selected_choice else CHOICE_INDENT


            choice_text = choice['text']
            max_choice_width = text_area_width - indent
            choice_lines = self.wrap_text(choice_text, max_choice_width)

            if choice_lines:
                choice_surface = self.font.render(choice_lines[0], True, color)
                screen.blit(choice_surface, (text_start_x + indent, choice_y))

def handle_input(self, event, character_name):

    if not self.is_active or not self.my_turn or not self.current_choices or self.dialog_completed:
        return None

    if event.type == pygame.KEYDOWN:

        if self.current_text != self.target_text:
            self.current_text = self.target_text
            return None

        if event.key == pygame.K_UP:
            self.selected_choice = (self.selected_choice - 1) % len(self.current_choices)
            return None
        elif event.key == pygame.K_DOWN:
            self.selected_choice = (self.selected_choice + 1) % len(self.current_choices)
            return None
        elif event.key == pygame.K_RETURN:
            if self.selected_choice < len(self.current_choices):
                return self.selected_choice
    return None
```

```
class Platform:
    def __init__(self, x, y, width, height, platform_type="normal", character_access=None):
        self.x = x
        self.y = y
        self.width = width
        self.height = height
        self.platform_type = platform_type
        self.character_access = character_access
        self.is_active = True
        self.rect = pygame.Rect(x, y, width, height)


        self.load_textures()

    def load_textures(self):

        try:

            self.block_texture = pygame.image.load(os.path.join("assets", "tiles", "mainlev_build.png")).convert_alpha()
            self.terrain_texture = pygame.image.load(os.path.join("assets", "tiles", "terrain.png")).convert_alpha()

            # Создаем поверхность для платформы с текстурой
            self.surface = self.create_textured_platform()
        except:
            # Если не удалось загрузить текстуры, используем цветные прямоугольники
            self.block_texture = None
            self.terrain_texture = None
            self.surface = None

    def create_textured_platform(self):

        if not self.block_texture:
            return None

        surface = pygame.Surface((self.width, self.height + 20), pygame.SRCALPHA)


        block_size = 16
        tile_size = 32


        for x in range(0, self.width, tile_size):
            for y in range(0, self.height + 20, tile_size):

                if y < self.height:

                    block_rect = pygame.Rect(0, 0, block_size, block_size)
                else:

                    block_rect = pygame.Rect(0, block_size, block_size, block_size)

                block_sprite = self.block_texture.subsurface(block_rect)
                scaled_block = pygame.transform.scale(block_sprite, (tile_size, tile_size))
                surface.blit(scaled_block, (x, y))


        if self.platform_type == "alice_only":
            overlay = pygame.Surface((self.width, self.height), pygame.SRCALPHA)
            overlay.fill((*ALICE_PLATFORM_COLOR, 80))
            surface.blit(overlay, (0, 0))
        elif self.platform_type == "rabbit_only":
            overlay = pygame.Surface((self.width, self.height), pygame.SRCALPHA)
```

```
            overlay.fill((*RABBIT_PLATFORM_COLOR, 80))
            surface.blit(overlay, (0, 0))
        elif self.platform_type == "switch":
            overlay = pygame.Surface((self.width, self.height), pygame.SRCALPHA)
            overlay.fill((*SWITCH_COLOR, 120))
            surface.blit(overlay, (0, 0))

        return surface

    def can_stand_on(self, character_name):

        if not self.is_active and self.platform_type == "door":
            return False

        return True

    def get_color(self):

        if self.platform_type == "alice_only":
            return ALICE_PLATFORM_COLOR
        elif self.platform_type == "rabbit_only":
            return RABBIT_PLATFORM_COLOR
        elif self.platform_type == "switch":
            return SWITCH_COLOR
        elif self.platform_type == "door":
            return DOOR_COLOR if self.is_active else (80, 40, 20)
        elif self.platform_type == "moving":
            return (255, 100, 255)
        else:
            return PLATFORM_COLOR

    def draw(self, screen, camera):

        screen_x, screen_y = camera.apply(self.x, self.y)
        if -self.width <= screen_x <= SCREEN_WIDTH and -self.height <= screen_y <= SCREEN_HEIGHT:


            if self.surface:

                screen.blit(self.surface, (screen_x, screen_y - 20))
            else:

                color = self.get_color()
                pygame.draw.rect(screen, color, (screen_x, screen_y, self.width, self.height))


                pygame.draw.rect(screen, (0, 0, 0), (screen_x, screen_y, self.width, self.height), 2)


            if self.platform_type == "alice_only":
                try:
                    font = pygame.font.Font(os.path.join("assets", "fonts", "visitor2.otf"), 18)
                except:
                    font = pygame.font.Font(None, 20)
                text = font.render("A", True, (255, 255, 255))
                text_rect = text.get_rect(center=(screen_x + self.width//2, screen_y + self.height//2))


                bg_rect = text_rect.inflate(6, 6)
                pygame.draw.rect(screen, (0, 0, 0, 128), bg_rect)
                screen.blit(text, text_rect)
            elif self.platform_type == "rabbit_only":
                try:
```

```
                font = pygame.font.Font(os.path.join("assets", "fonts", "visitor2.otf"), 18)
            except:
                font = pygame.font.Font(None, 20)
            text = font.render("R", True, (255, 255, 255))
            text_rect = text.get_rect(center=(screen_x + self.width//2, screen_y + self.height//2))


            bg_rect = text_rect.inflate(6, 6)
            pygame.draw.rect(screen, (0, 0, 0, 128), bg_rect)
            screen.blit(text, text_rect)
        elif self.platform_type == "switch":
            try:
                font = pygame.font.Font(os.path.join("assets", "fonts", "visitor2.otf"), 14)
            except:
                font = pygame.font.Font(None, 16)
            text = font.render("SW", True, (0, 0, 0))
            text_rect = text.get_rect(center=(screen_x + self.width//2, screen_y + self.height//2))
            screen.blit(text, text_rect)
        elif self.platform_type == "moving":

            pass

class Collectible:
    def __init__(self, x, y, collectible_type="key"):
        self.x = x
        self.y = y
        self.collectible_type = collectible_type
        self.collected = False
        self.rect = pygame.Rect(x, y, 20, 20)

    def collect(self, character_name):

        if not self.collected:
            self.collected = True
            return True
        return False

    def draw(self, screen, camera):

        if self.collected:
            return

        screen_x, screen_y = camera.apply(self.x, self.y)
        if -20 <= screen_x <= SCREEN_WIDTH and -20 <= screen_y <= SCREEN_HEIGHT:
            pygame.draw.circle(screen, COLLECTIBLE_COLOR, (screen_x + 10, screen_y + 10), 10)
            pygame.draw.circle(screen, (255, 255, 255), (screen_x + 10, screen_y + 10), 10, 2)


            try:
                font = pygame.font.Font(os.path.join("assets", "fonts", "visitor2.otf"), 14)
            except:
                font = pygame.font.Font(None, 16)
            text = font.render("K", True, (255, 255, 255))
            screen.blit(text, (screen_x + 6, screen_y + 4))

class AnimatedKey:
    def __init__(self, x, y, key_type="key2"):
        self.x = x
        self.y = y
        self.key_type = key_type
        self.collected = False
        self.rect = pygame.Rect(x, y, 32, 32)
        self.animation_frames = []
```

```
    self.current_frame = 0
    self.animation_timer = 0
    self.animation_speed = 0.1
    self.load_animation()

def load_animation(self):

    try:
        if self.key_type == "key2":

            for i in range(12):
                frame_path = os.path.join("assets", "items", f"Key 2 - GOLD - {i:04d}.png")
                frame = pygame.image.load(frame_path).convert_alpha()

                original_size = frame.get_size()
                aspect_ratio = original_size[0] / original_size[1]
                new_height = 32
                new_width = int(new_height * aspect_ratio)
                frame = pygame.transform.scale(frame, (new_width, new_height))
                self.animation_frames.append(frame)
        elif self.key_type == "key5":

            for i in range(18):
                frame_path = os.path.join("assets", "items", f"Key 5 - GOLD - frame{i:04d}.png")
                frame = pygame.image.load(frame_path).convert_alpha()
                # Сохраняем пропорции при масштабировании
                original_size = frame.get_size()
                aspect_ratio = original_size[0] / original_size[1]
                new_height = 32
                new_width = int(new_height * aspect_ratio)
                frame = pygame.transform.scale(frame, (new_width, new_height))
                self.animation_frames.append(frame)
        elif self.key_type == "key15":

            for i in range(48):
                frame_path = os.path.join("assets", "items", f"Key 15 - GOLD - frame{i:04d}.png")
                frame = pygame.image.load(frame_path).convert_alpha()

                original_size = frame.get_size()
                aspect_ratio = original_size[0] / original_size[1]
                new_height = 32
                new_width = int(new_height * aspect_ratio)
                frame = pygame.transform.scale(frame, (new_width, new_height))
                self.animation_frames.append(frame)
    except:

        fallback = pygame.Surface((32, 32), pygame.SRCALPHA)
        pygame.draw.circle(fallback, (255, 215, 0), (16, 16), 12)
        pygame.draw.circle(fallback, (255, 255, 255), (16, 16), 8)
        self.animation_frames = [fallback]

def update(self, dt):

    if not self.collected and self.animation_frames:
        self.animation_timer += dt
        if self.animation_timer >= self.animation_speed:
            self.animation_timer = 0
            self.current_frame = (self.current_frame + 1) % len(self.animation_frames)

def collect(self, character_name):

    if not self.collected:
        self.collected = True
```

```
            return True
        return False

    def draw(self, screen, camera):

        if self.collected or not self.animation_frames:
            return

        screen_x, screen_y = camera.apply(self.x, self.y)
        current_sprite = self.animation_frames[self.current_frame]
        sprite_width = current_sprite.get_width()
        sprite_height = current_sprite.get_height()

        if -sprite_width <= screen_x <= SCREEN_WIDTH and -sprite_height <= screen_y <= SCREEN_HEIGHT:
            screen.blit(current_sprite, (screen_x, screen_y))

class Potion:
    def __init__(self, x, y, potion_type="red"):
        self.x = x
        self.y = y
        self.potion_type = potion_type
        self.collected = False
        self.rect = pygame.Rect(x, y, 24, 32)
        self.load_texture()

    def load_texture(self):

        try:
            if self.potion_type == "red":
                self.texture = pygame.image.load(os.path.join("assets", "items", "Red Potion.png")).convert_alpha()
            elif self.potion_type == "green":
                self.texture = pygame.image.load(os.path.join("assets", "items", "Green Potion.png")).convert_alpha()
            elif self.potion_type == "blue":
                self.texture = pygame.image.load(os.path.join("assets", "items", "Blue Potion.png")).convert_alpha()


            original_size = self.texture.get_size()
            aspect_ratio = original_size[0] / original_size[1]
            new_height = 32
            new_width = int(new_height * aspect_ratio)
            self.texture = pygame.transform.scale(self.texture, (new_width, new_height))


            self.rect = pygame.Rect(self.x, self.y, new_width, new_height)
        except:
            self.texture = None

    def collect(self, character_name):

        if not self.collected:
            self.collected = True
            return True
        return False

    def draw(self, screen, camera):

        if self.collected:
            return

        screen_x, screen_y = camera.apply(self.x, self.y)
        if self.texture:
            texture_width = self.texture.get_width()
            texture_height = self.texture.get_height()
```

```
        if -texture_width <= screen_x <= SCREEN_WIDTH and -texture_height <= screen_y <=
SCREEN_HEIGHT:
            screen.blit(self.texture, (screen_x, screen_y))
        else:

            color = (255, 0, 0) if self.potion_type == "red" else (0, 255, 0) if self.potion_type == "green" else (0, 0, 255)
            pygame.draw.rect(screen, color, (screen_x, screen_y, 24, 32))

class Lamp:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.width = 32
        self.height = 64
        self.rect = pygame.Rect(x, y, self.width, self.height)
        self.load_texture()

    def load_texture(self):

        try:
            self.texture = pygame.image.load(os.path.join("assets", "tiles", "oak_woods_v1.0", "decorations",
"lamp.png")).convert_alpha()
            self.texture = pygame.transform.scale(self.texture, (self.width, self.height))
        except:
            self.texture = None

    def draw(self, screen, camera):

        screen_x, screen_y = camera.apply(self.x, self.y)
        if -self.width <= screen_x <= SCREEN_WIDTH and -self.height <= screen_y <= SCREEN_HEIGHT:
            if self.texture:
                screen.blit(self.texture, (screen_x, screen_y))
            else:

                pygame.draw.rect(screen, (139, 69, 19), (screen_x, screen_y + 40, 8, 24))  # Столб
                pygame.draw.circle(screen, (255, 255, 0), (screen_x + 4, screen_y + 20), 12)  # Свет
                pygame.draw.circle(screen, (255, 255, 255), (screen_x + 4, screen_y + 20), 8)  # Лампа

class Decoration:
    def __init__(self, x, y, decoration_type="grass"):
        self.x = x
        self.y = y
        self.decoration_type = decoration_type
        self.width = 32
        self.height = 32
        self.rect = pygame.Rect(x, y, self.width, self.height)
        self.load_texture()

    def load_texture(self):

        try:
            if self.decoration_type == "grass1":
                self.texture = pygame.image.load(os.path.join("assets", "tiles", "oak_woods_v1.0", "decorations",
"grass_1.png")).convert_alpha()
            elif self.decoration_type == "grass2":
                self.texture = pygame.image.load(os.path.join("assets", "tiles", "oak_woods_v1.0", "decorations",
"grass_2.png")).convert_alpha()
            elif self.decoration_type == "grass3":
                self.texture = pygame.image.load(os.path.join("assets", "tiles", "oak_woods_v1.0", "decorations",
"grass_3.png")).convert_alpha()
            elif self.decoration_type == "rock1":
                self.texture = pygame.image.load(os.path.join("assets", "tiles", "oak_woods_v1.0", "decorations",
"rock_1.png")).convert_alpha()
```

```
            elif self.decoration_type == "rock2":
                self.texture = pygame.image.load(os.path.join("assets", "tiles", "oak_woods_v1.0", "decorations",
"rock_2.png")).convert_alpha()
            elif self.decoration_type == "rock3":
                self.texture = pygame.image.load(os.path.join("assets", "tiles", "oak_woods_v1.0", "decorations",
"rock_3.png")).convert_alpha()
            elif self.decoration_type == "fence":
                self.texture = pygame.image.load(os.path.join("assets", "tiles", "oak_woods_v1.0", "decorations",
"fence_1.png")).convert_alpha()
            elif self.decoration_type == "fence2":
                self.texture = pygame.image.load(os.path.join("assets", "tiles", "oak_woods_v1.0", "decorations",
"fence_2.png")).convert_alpha()


            self.texture = pygame.transform.scale(self.texture, (self.width, self.height))
        except:
            self.texture = None

    def draw(self, screen, camera):

        screen_x, screen_y = camera.apply(self.x, self.y)
        if -self.width <= screen_x <= SCREEN_WIDTH and -self.height <= screen_y <= SCREEN_HEIGHT:
            if self.texture:
                screen.blit(self.texture, (screen_x, screen_y))
            else:

                color = (34, 139, 34) if "grass" in self.decoration_type else (139, 69, 19)
                pygame.draw.rect(screen, color, (screen_x, screen_y, self.width, self.height))

class MovingPlatform(Platform):
    def __init__(self, x, y, width, height, target_x, target_y, platform_type="moving", movement_type="linear"):
        super().__init__(x, y, width, height, platform_type)
        self.start_x = x
        self.start_y = y
        self.target_x = target_x
        self.target_y = target_y
        self.is_moving = False
        self.move_speed = 2
        self.activated = False
        self.movement_type = movement_type
        self.direction = 1
        self.move_distance = 100

    def activate(self):

        if not self.activated:
            self.activated = True
            self.is_moving = True

    def update(self):
        """Обновляет позицию платформы"""
        if not self.activated:
            return

        if self.movement_type == "linear":

            if self.is_moving:
                dx = self.target_x - self.x
                dy = self.target_y - self.y
                distance = (dx * dx + dy * dy) ** 0.5

                if distance > self.move_speed:
                    self.x += (dx / distance) * self.move_speed
```

```
                self.y += (dy / distance) * self.move_speed
                self.rect.x = self.x
                self.rect.y = self.y
            else:
                self.x = self.target_x
                self.y = self.target_y
                self.rect.x = self.x
                self.rect.y = self.y
                self.is_moving = False

        elif self.movement_type == "horizontal":

            self.x += self.direction * self.move_speed


            if self.x >= self.start_x + self.move_distance:
                self.x = self.start_x + self.move_distance
                self.direction = -1
            elif self.x <= self.start_x - self.move_distance:
                self.x = self.start_x - self.move_distance
                self.direction = 1

            self.rect.x = self.x

        elif self.movement_type == "vertical":

            self.y += self.direction * self.move_speed


            if self.y >= self.start_y + self.move_distance:
                self.y = self.start_y + self.move_distance
                self.direction = -1
            elif self.y <= self.start_y - self.move_distance:
                self.y = self.start_y - self.move_distance
                self.direction = 1

            self.rect.y = self.y

class Game:
    def __init__(self, host, is_host):
        self.screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
        pygame.display.set_caption("P2P Game")
        self.clock = pygame.time.Clock()
        self.camera = Camera(SCREEN_WIDTH, SCREEN_HEIGHT)

        # Флаг для контроля состояния сокета
        self.socket_active = True

        # Создаем фон и платформу
        self.background = self.create_background()
        self.platform = self.create_platform()
        self.shadow = self.create_shadow()

        # Проверяем и создаем файл диалогов, если он отсутствует
        self.ensure_dialog_file_exists()

        # Загружаем финальные картинки
        try:
            self.final_image = pygame.image.load(os.path.join("assets", "tiles", "Final level.png")).convert_alpha()
            self.final_image = pygame.transform.scale(self.final_image, (SCREEN_WIDTH, SCREEN_HEIGHT))
            print("Final level.png успешно загружен")

            smile_path = os.path.join("assets", "tiles", "Smile.png")
```

```
    if not os.path.exists(smile_path):
        print(f"Ошибка: файл {smile_path} не найден")
        self.smile_image = None
    else:
        self.smile_image = pygame.image.load(smile_path).convert_alpha()
        self.smile_image = pygame.transform.scale(self.smile_image, (SCREEN_WIDTH, SCREEN_HEIGHT))
        print("Smile.png успешно загружен")
except Exception as e:
    print(f"Ошибка при загрузке картинок: {e}")
    print(f"Проверьте наличие файлов в папке assets/tiles/")
    self.final_image = None
    self.smile_image = None

# Создание игроков с соответствующими спрайтами
platform_y = WORLD_HEIGHT - PLATFORM_HEIGHT
if is_host:
    self.my_player = Player(100, platform_y - ALICE_HEIGHT + ALICE_OFFSET, "alice")
    self.other_player = Player(250, platform_y - RABBIT_HEIGHT + RABBIT_OFFSET, "rabbit")
else:
    self.my_player = Player(250, platform_y - RABBIT_HEIGHT + RABBIT_OFFSET, "rabbit")
    self.other_player = Player(100, platform_y - ALICE_HEIGHT + ALICE_OFFSET, "alice")

# Добавляем флаг для контроля завершения работы
self.is_shutting_down = False

# Настройка сети
self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
if is_host:
    self.socket.bind((host, 5000))
    self.other_address = (host, 5001)
else:
    self.socket.bind((host, 5001))
    self.other_address = (host, 5000)

# Устанавливаем таймаут для сокета
self.socket.settimeout(0.1)

# Запуск потока для приема данных
self.receive_thread = threading.Thread(target=self.receive_data, daemon=False)  # Делаем поток не демоном
self.receive_thread.start()

# Добавляем систему диалогов
self.dialog_system = DialogSystem()
self.is_host = is_host

# Инициализируем состояние диалога
self.dialog_state = {
    "is_active": False,
    "current_dialog_id": None,
    "current_speaker": None,
    "dialog_completed": False
}

# Создаем платформы и препятствия
self.platforms = self.create_platforms()
self.animated_keys = self.create_animated_keys()
self.potions = self.create_potions()
self.lamps = self.create_lamps()
self.decorations = self.create_decorations()
self.signs = self.create_signs()
self.collected_keys = 0
self.collected_potions = 0
self.moving_platforms = self.create_moving_platforms()
```

```
        # Состояние финала
        self.victory_achieved = False
        self.victory_timer = 0
        self.victory_duration = 5.0  # 5 секунд показа финального экрана

        # Если мы хост, инициируем первый диалог после небольшой задержки
        if is_host:
            self.initial_dialog_timer = 2.0  # 2 секунды задержки
        else:
            self.initial_dialog_timer = None

    def ensure_dialog_file_exists(self):
        """Проверяет наличие файла диалогов и создает его, если отсутствует"""
        dialog_path = os.path.join("assets", "dialogs")
        dialog_file = os.path.join(dialog_path, "rabbit_dialog.json")

        if not os.path.exists(dialog_path):
            os.makedirs(dialog_path)
            print(f"Создана директория диалогов: {dialog_path}")

        if not os.path.exists(dialog_file):
            default_dialogs = {
                "start": {
                    "text": "Ой-ой! Я опаздываю! Я опаздываю!",
                    "speaker": "rabbit",
                    "choices": [
                        {
                            "text": "Постойте! Куда вы так спешите?",
                            "next": "dialog2"
                        }
                    ]
                },
                "dialog2": {
                    "text": "На важное-преважное чаепитие! Нет времени объяснять!",
                    "speaker": "rabbit",
                    "choices": [
                        {
                            "text": "Можно мне с вами?",
                            "next": "dialog3"
                        }
                    ]
                },
                "dialog3": {
                    "text": "Следуйте за мной, если осмелитесь! Только поторопитесь!",
                    "speaker": "rabbit",
                    "choices": [
                        {
                            "text": "Я иду за вами!",
                            "next": "end"
                        }
                    ]
                }
            }

            try:
                with open(dialog_file, 'w', encoding='utf-8') as f:
                    json.dump(default_dialogs, f, ensure_ascii=False, indent=4)
                print(f"Создан файл диалогов: {dialog_file}")
            except Exception as e:
                print(f"Ошибка при создании файла диалогов: {e}")

    def create_shadow(self):
```

```
        """Создает градиент затемнения сверху"""
        shadow = pygame.Surface((SCREEN_WIDTH, SCREEN_HEIGHT), pygame.SRCALPHA)

        # Создаем градиент от темного к прозрачному
        for y in range(SHADOW_HEIGHT):
            # Нелинейный градиент для более плавного перехода
            progress = y / SHADOW_HEIGHT
            alpha = int(SHADOW_ALPHA * (1 - progress * progress))
            pygame.draw.line(shadow, (0, 0, 0, alpha), (0, y), (SCREEN_WIDTH, y))

        return shadow

    def create_platform(self):
        """Создает платформу для ходьбы"""
        platform = pygame.Surface((SCREEN_WIDTH, PLATFORM_HEIGHT))
        platform.fill(DIRT_BROWN)

        # Добавляем текстуру на платформу
        for _ in range(500):  # Меньше вариаций для платформы
            x = random.randint(0, SCREEN_WIDTH)
            y = random.randint(0, PLATFORM_HEIGHT)
            size = random.randint(2, 4)
            color = random.choice([DIRT_DARK, DIRT_LIGHT])
            pygame.draw.ellipse(platform, color, (x, y, size, size * 0.7))

        # Добавляем верхнюю границу платформы
        pygame.draw.line(platform, DIRT_LIGHT, (0, 0), (SCREEN_WIDTH, 0), 2)

        return platform

    def create_background(self):
        """Создает многослойный фон пещеры с параллаксом"""
        backgrounds = {}

        try:
            # Загружаем 4 слоя фона пещеры с правильными именами
            bg1 = pygame.image.load(os.path.join("assets", "tiles", "background_caves  1.png")).convert_alpha()
            bg2 = pygame.image.load(os.path.join("assets", "tiles", "background_caves  2.png")).convert_alpha()
            bg3 = pygame.image.load(os.path.join("assets", "tiles", "background_caves  3.png")).convert_alpha()
            bg4 = pygame.image.load(os.path.join("assets", "tiles", "background_caves  4.png")).convert_alpha()

            # Масштабируем фоны под размер экрана
            backgrounds['layer1'] = pygame.transform.scale(bg1, (SCREEN_WIDTH, SCREEN_HEIGHT))
            backgrounds['layer2'] = pygame.transform.scale(bg2, (SCREEN_WIDTH, SCREEN_HEIGHT))
            backgrounds['layer3'] = pygame.transform.scale(bg3, (SCREEN_WIDTH, SCREEN_HEIGHT))
            backgrounds['layer4'] = pygame.transform.scale(bg4, (SCREEN_WIDTH, SCREEN_HEIGHT))

        except Exception as e:
            print(f"Ошибка загрузки фона: {e}")
            # Fallback - создаем простой градиентный фон пещеры
            background = pygame.Surface((SCREEN_WIDTH, SCREEN_HEIGHT))
            for y in range(SCREEN_HEIGHT):
                # Градиент от темно-серого к черному
                color_value = max(20, 80 - (y * 60 // SCREEN_HEIGHT))
                color = (color_value, color_value - 10, color_value - 5)
                pygame.draw.line(background, color, (0, y), (SCREEN_WIDTH, y))

            backgrounds['layer1'] = background
            backgrounds['layer2'] = background.copy()
            backgrounds['layer3'] = background.copy()
            backgrounds['layer4'] = background.copy()

        return backgrounds
```

```
    def create_platforms(self):
        """Создает платформы и препятствия для кооперативного прохождения"""
        platforms = []

        # Обычные платформы для прыжков (разные высоты для разнообразия)

        platforms.append(Platform(1000, 650, 100, 20))  # Средняя высота
        platforms.append(Platform(1300, 600, 100, 20))  # Выше

        # Высокие платформы для зайца (немного изменены)
        platforms.append(Platform(1600, 500, 120, 20))  # Немного ниже
        platforms.append(Platform(1900, 450, 100, 20))  # Выше
        platforms.append(Platform(2200, 400, 100, 20))  # Самая высокая

        # Низкие платформы для Алисы (триггеры для подвижных платформ)
        platforms.append(Platform(1800, 820, 100, 20, "alice_trigger"))  # Ниже
        platforms.append(Platform(2000, 800, 100, 20, "alice_trigger"))  # Ниже

        # Специальные низкие платформы для Алисы (поднимаем выше, чтобы не мешали проходу)
        ground_level = WORLD_HEIGHT - PLATFORM_HEIGHT

        platforms.append(Platform(1100, ground_level - 80, 100, 20))  # Поднимаем выше
        platforms.append(Platform(1400, ground_level - 100, 120, 20))  # Поднимаем выше



        # Дополнительные платформы для разнообразия

        platforms.append(Platform(1750, ground_level - 100, 90, 20)) # Средняя высота
        platforms.append(Platform(2450, ground_level - 120, 100, 20)) # Высокая

        # Финальные платформы для совместного прохождения
        platforms.append(Platform(2800, 600, 150, 20))
        platforms.append(Platform(3000, 550, 150, 20))

        # Выход - убираем низкую платформу под высокой
        platforms.append(Platform(3200, 500, 200, 20))

        # Дополнительная платформа для доступа к предметам
        platforms.append(Platform(850, ground_level - 120, 80, 20))  # Для доступа к ключу

        return platforms

    def create_animated_keys(self):
        """Создает анимированные ключи"""
        keys = []
        ground_level = WORLD_HEIGHT - PLATFORM_HEIGHT  # Добавляем определение ground_level

        # Ключи на разных уровнях для интересного геймплея
        keys.append(AnimatedKey(1120, ground_level - 112, "key2"))   # На платформе 1100 (первый ключ)
        keys.append(AnimatedKey(1650, 470, "key5"))  # На высокой платформе зайца
        keys.append(AnimatedKey(770, WORLD_HEIGHT - PLATFORM_HEIGHT - 32, "key15")) # На земле для
Алисы

        return keys

    def create_potions(self):
        """Создает зелья"""
        potions = []

        # Зелья на разных уровнях для всех персонажей
```

```
        potions.append(Potion(620, WORLD_HEIGHT - PLATFORM_HEIGHT - 92, "red"))    # На платформе 600 (доступно)
        potions.append(Potion(1750, 420, "green")) # На высокой платформе зайца (выше)
        potions.append(Potion(1770, WORLD_HEIGHT - PLATFORM_HEIGHT - 132, "blue"))  # На средней платформе

        return potions

    def create_lamps(self):
        """Создает фонари"""
        lamps = []

        # Фонари на разных уровнях для лучшего освещения
        ground_level = WORLD_HEIGHT - PLATFORM_HEIGHT - 64  # На уровне земли
        lamps.append(Lamp(750, ground_level))   # У старта
        lamps.append(Lamp(1250, ground_level))  # На пути
        lamps.append(Lamp(1750, ground_level))  # У платформ Алисы
        lamps.append(Lamp(2150, ground_level))  # У высоких платформ зайца
        lamps.append(Lamp(2750, ground_level))  # У финала

        # Дополнительные фонари на платформах

        lamps.append(Lamp(1620, 470))  # На высокой платформе
        lamps.append(Lamp(2820, 570))  # На финальной платформе

        return lamps

    def create_decorations(self):
        """Создает декорации"""
        decorations = []

        # Уменьшенное количество декораций на земле
        ground_level = WORLD_HEIGHT - PLATFORM_HEIGHT - 32
        decorations.append(Decoration(700, ground_level, "rock1"))
        decorations.append(Decoration(1500, ground_level, "grass2"))
        decorations.append(Decoration(2100, ground_level, "fence"))
        decorations.append(Decoration(2700, ground_level, "rock2"))

        # Декорации на платформах (обновленные позиции)
        decorations.append(Decoration(1020, 620, "grass2"))     # На второй платформе
        decorations.append(Decoration(1320, 570, "fence"))      # На третьей платформе
        decorations.append(Decoration(1620, 470, "rock1"))      # На высокой платформе
        decorations.append(Decoration(1920, 420, "grass3"))     # На платформе зайца
        decorations.append(Decoration(2220, 370, "rock2"))      # На самой высокой платформе

        # Декорации на новых платформах
        decorations.append(Decoration(1770, ground_level - 100, "rock3"))  # На средней
        decorations.append(Decoration(2470, ground_level - 120, "fence"))  # На высокой

        return decorations

    def create_signs(self):
        """Создает знаки"""
        signs = []

        # Только два знака в самом конце карты
        ground_level = WORLD_HEIGHT - PLATFORM_HEIGHT - 48

        # Знак на высокой платформе (для зайца)
        signs.append(Sign(2850, 550))  # На платформе 2800, 600

        # Знак на земле (для Алисы)
        signs.append(Sign(2850, ground_level))  # На земле
```

```
        return signs

    def create_moving_platforms(self):
        """Создает подвижные платформы для зайца и Алисы"""
        moving_platforms = []

        # Платформы для кролика - горизонтальное движение
        moving_platforms.append(MovingPlatform(1500, 600, 100, 20, 0, 0, "moving", "horizontal"))
        moving_platforms.append(MovingPlatform(2100, 550, 100, 20, 0, 0, "moving", "horizontal"))

        # Платформы для Алисы - вертикальное движение
        moving_platforms.append(MovingPlatform(1700, 700, 100, 20, 0, 0, "moving", "vertical"))
        moving_platforms.append(MovingPlatform(2300, 650, 100, 20, 0, 0, "moving", "vertical"))

        # Активируем их сразу для демонстрации
        for platform in moving_platforms:
            platform.activate()

        return moving_platforms

    def check_victory_condition(self):
        """Проверяет условие победы - оба персонажа рядом с знаками и все предметы собраны"""
        if self.victory_achieved:
            return

        # Проверяем, собраны ли все предметы
        if self.collected_keys < 3 or self.collected_potions < 3:
            return  # Если не все предметы собраны, победа невозможна

        # Находим позиции обоих игроков
        alice = self.my_player if self.my_player.character_name == "alice" else self.other_player
        rabbit = self.other_player if self.my_player.character_name == "alice" else self.my_player

        # Проверяем расстояние до знаков (первый на платформе, второй на земле)
        if len(self.signs) >= 2:
            platform_sign = self.signs[0]  # Знак на платформе (для зайца)
            ground_sign = self.signs[1]    # Знак на земле (для Алисы)

            # Расстояние для активации знака
            sign_distance = 80

            # Проверяем, находится ли зайец рядом со знаком на платформе, а Алиса рядом со знаком на земле
            rabbit_near_platform = ((rabbit.x - platform_sign.x) ** 2 + (rabbit.y - platform_sign.y) ** 2) ** 0.5 <
sign_distance
            alice_near_ground = ((alice.x - ground_sign.x) ** 2 + (alice.y - ground_sign.y) ** 2) ** 0.5 < sign_distance

            # Условие победы: все предметы собраны И зайец у знака на платформе И Алиса у знака на земле
            if rabbit_near_platform and alice_near_ground:
                self.victory_achieved = True
                self.victory_timer = 0
                print("Победа! Добро пожаловать в страну чудес!")

    def draw_victory_screen(self, screen):
        """Отрисовывает экран победы"""
        if not self.victory_achieved:
            return

        # Белый фон с плавным появлением
        white_surface = pygame.Surface((SCREEN_WIDTH, SCREEN_HEIGHT))
        white_surface.fill((255, 255, 255))
        white_alpha = min(255, int(self.victory_timer * 255))  # Плавное появление белого
        white_surface.set_alpha(white_alpha)
```

```
        screen.blit(white_surface, (0, 0))

        # Текст победы (черный, с анимацией печатания)
        if self.victory_timer > 1:  # Текст появляется через секунду
            try:
                font = pygame.font.Font(os.path.join("assets", "fonts", "visitor2.otf"), 48)
            except:
                font = pygame.font.Font(None, 48)

            full_text = "Добро пожаловать в страну чудес!"
            # Анимация печатания текста
            text_progress = min(1.0, (self.victory_timer - 1) * 0.5)  # Полное появление за 2 секунды
            visible_chars = int(len(full_text) * text_progress)
            current_text = full_text[:visible_chars]

            if current_text:  # Проверяем, что текст не пустой
                text = font.render(current_text, True, (0, 0, 0))  # Черный текст
                text_rect = text.get_rect(center=(SCREEN_WIDTH//2, SCREEN_HEIGHT//2))
                screen.blit(text, text_rect)

        # Улыбка появляется после завершения анимации текста
        if self.victory_timer > 4 and self.smile_image:  # Улыбка появляется через 4 секунды
            smile_alpha = min(255, int((self.victory_timer - 4) * 255))  # Плавное появление

            # Картинка на весь экран
            scaled_image = pygame.transform.scale(self.smile_image, (SCREEN_WIDTH, SCREEN_HEIGHT))

            # Создаем временную поверхность для правильного наложения прозрачности
            temp_surface = pygame.Surface((SCREEN_WIDTH, SCREEN_HEIGHT), pygame.SRCALPHA)
            temp_surface.blit(scaled_image, (0, 0))
            temp_surface.set_alpha(smile_alpha)
            screen.blit(temp_surface, (0, 0))

    def check_platform_activation(self):
        """Проверяет активацию подвижных платформ"""
        triggered_platforms = self.my_player.check_platform_triggers(self.platforms)

        for platform in triggered_platforms:
            # Когда Алиса становится на платформу, активируем соответствующую подвижную платформу
            platform_id = f"{platform.x}_{platform.y}"

            # Активируем подвижные платформы в зависимости от триггера
            if platform.x == 1800:  # Первая платформа Алисы
                if len(self.moving_platforms) > 0 and not self.moving_platforms[0].activated:
                    self.moving_platforms[0].activate()
                    print("Активирована первая подвижная платформа для зайца!")
            elif platform.x == 2000:  # Вторая платформа Алисы
                if len(self.moving_platforms) > 1 and not self.moving_platforms[1].activated:
                    self.moving_platforms[1].activate()
                    print("Активирована вторая подвижная платформа для зайца!")

    def make_choice(self, choice_index):
        """Обработка выбора варианта ответа"""
        if not self.dialog_system.my_turn or not self.dialog_system.current_dialog:
            return

        if choice_index >= len(self.dialog_system.current_dialog.get("choices", [])):
            return

        choice = self.dialog_system.current_dialog["choices"][choice_index]
        next_dialog = choice.get("next")

        # Если это конец диалога
```

```
        if next_dialog == "end":
            self.end_dialog()
            # Отправляем сообщение о завершении диалога
            try:
                data = pickle.dumps({
                    "type": "dialog_end",
                    "dialog_state": self.dialog_state
                })
                self.socket.sendto(data, self.other_address)
            except Exception as e:
                print(f"Ошибка при отправке завершения диалога: {e}")
            return

        # Переход к следующему диалогу
        if next_dialog and next_dialog in self.dialog_system.dialogs:
            next_dialog_data = self.dialog_system.dialogs[next_dialog]
            next_speaker = next_dialog_data.get("speaker")

            self.dialog_state["current_dialog_id"] = next_dialog
            self.dialog_state["current_speaker"] = next_speaker
            self.dialog_state["is_active"] = True

            # Отправляем обновление состояния диалога
            try:
                data = pickle.dumps({
                    "type": "dialog_update",
                    "dialog_state": self.dialog_state
                })
                self.socket.sendto(data, self.other_address)
            except Exception as e:
                print(f"Ошибка при отправке обновления диалога: {e}")

            # Запускаем новый диалог
            self.dialog_system.reset_state()
            self.dialog_system.start_dialog(next_dialog, self.my_player.character_name)
        else:
            # Если следующего диалога нет, завершаем
            self.end_dialog()

    def receive_data(self):
        """Получение данных от другого игрока"""
        while self.socket_active and not self.is_shutting_down:
            try:
                if not self.socket_active or self.is_shutting_down:
                    break

                data, addr = self.socket.recvfrom(1024)
                if not data:
                    continue

                received_data = pickle.loads(data)

                # Обработка сообщений о подключении
                if received_data.get("type") == "dialog_update":
                    dialog_state = received_data.get("dialog_state")
                    if dialog_state:
                        self.dialog_state.update(dialog_state)
                        if dialog_state.get("current_dialog_id"):
                            self.dialog_system.start_dialog(
                                dialog_state["current_dialog_id"],
                                self.my_player.character_name
                            )
```

```
            elif received_data.get("type") == "dialog_end":
                self.dialog_system.complete_dialog()
                self.dialog_state["is_active"] = False
                self.dialog_state["current_dialog_id"] = None
                self.dialog_state["current_speaker"] = None
                self.dialog_state["dialog_completed"] = True

            else:
                # Обработка обычных игровых данных
                if "player" in received_data:
                    player_data = received_data["player"]
                    self.other_player.x = player_data["x"]
                    self.other_player.y = player_data["y"]
                    self.other_player.facing_right = player_data["facing_right"]
                    self.other_player.moving = player_data["moving"]

                if "collected_keys" in received_data:
                    for key_index in received_data["collected_keys"]:
                        if key_index < len(self.animated_keys):
                            self.animated_keys[key_index].collected = True

                if "collected_potions" in received_data:
                    for potion_index in received_data["collected_potions"]:
                        if potion_index < len(self.potions):
                            self.potions[potion_index].collected = True

                if "counters" in received_data:
                    counters = received_data["counters"]
                    if counters["keys"] > self.collected_keys:
                        self.collected_keys = counters["keys"]
                    if counters["potions"] > self.collected_potions:
                        self.collected_potions = counters["potions"]

        except:
            if not self.socket_active or self.is_shutting_down:
                break
            continue

def send_data(self):
    """Отправка данных другому игроку"""
    if not self.socket_active:  # Проверяем флаг перед отправкой
        return

    try:
        # Собираем данные о собранных предметах
        collected_keys_data = []
        for i, key in enumerate(self.animated_keys):
            if key.collected:
                collected_keys_data.append(i)

        collected_potions_data = []
        for i, potion in enumerate(self.potions):
            if potion.collected:
                collected_potions_data.append(i)

        data = pickle.dumps({
            "type": "game_state",
            "player": {
                "x": self.my_player.x,
                "y": self.my_player.y,
                "facing_right": self.my_player.facing_right,
                "moving": self.my_player.moving
            },
```

```
            "collected_keys": collected_keys_data,
            "collected_potions": collected_potions_data,
            "counters": {
                "keys": self.collected_keys,
                "potions": self.collected_potions
            }
        })
        if self.socket_active:  # Дополнительная проверка перед отправкой
            self.socket.sendto(data, self.other_address)
    except socket.error:
        self.socket_active = False
    except Exception:
        pass

def close(self):
    """Корректно закрываем игру и сетевое соединение"""
    # Устанавливаем флаг завершения работы
    self.is_shutting_down = True
    self.socket_active = False

    # Закрываем сокет
    try:
        if hasattr(self, 'socket'):
            try:
                self.socket.shutdown(socket.SHUT_RDWR)
            except:
                pass
            try:
                self.socket.close()
            except:
                pass
    except:
        pass

    # Ждем завершения потока receive_thread
    if hasattr(self, 'receive_thread'):
        try:
            self.receive_thread.join(timeout=2.0)  # Увеличиваем таймаут до 2 секунд
        except:
            pass

def run(self):
    running = True
    last_time = time.time()
    try:
        while running:
            current_time = time.time()
            dt = current_time - last_time
            last_time = current_time

            # Обработка начального диалога для хоста
            if self.is_host and self.initial_dialog_timer is not None:
                self.initial_dialog_timer -= dt
                if self.initial_dialog_timer <= 0:
                    self.initial_dialog_timer = None
                    self.start_dialog("start")

            # Получаем состояние клавиш
            keys = pygame.key.get_pressed()

            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    running = False
```

```
                    self.is_shutting_down = True
                    break
                elif event.type == pygame.KEYDOWN:
                    if (event.key == pygame.K_SPACE or event.key == pygame.K_w) and not
self.dialog_system.is_active:
                        self.my_player.jump()
                    else:
                        self.handle_input(event)

            if not running:
                break

            # Проверяем, есть ли активный диалог, блокирующий движение
            allow_movement = not self.dialog_system.is_active

            # Обрабатываем нажатия клавиш
            if allow_movement:
                if keys[pygame.K_LEFT] or keys[pygame.K_a]:
                    self.my_player.move(-1)
                elif keys[pygame.K_RIGHT] or keys[pygame.K_d]:
                    self.my_player.move(1)
                else:
                    self.my_player.move(0)
            else:
                self.my_player.move(0)

            # Обновления игры...
            self.my_player._platforms = self.platforms + self.moving_platforms
            self.other_player._platforms = self.platforms + self.moving_platforms

            self.my_player.update(self.platforms + self.moving_platforms)
            self.other_player.update(self.platforms + self.moving_platforms)

            for key in self.animated_keys:
                key.update(dt)

            for platform in self.moving_platforms:
                platform.update()

            for sign in self.signs:
                sign.update([self.my_player, self.other_player])

            collected = self.my_player.check_collectibles(self.animated_keys)
            if collected:
                self.collected_keys += len(collected)

            collected = self.my_player.check_collectibles(self.potions)
            if collected:
                self.collected_potions += len(collected)

            self.check_platform_activation()
            self.check_victory_condition()

            if self.initial_dialog_timer is None:
                self.check_dialog_trigger()

            next_x = self.my_player.x
            next_y = self.my_player.y
            if keys[pygame.K_LEFT]:
                next_x -= self.my_player.move_speed * 5
            elif keys[pygame.K_RIGHT]:
                next_x += self.my_player.move_speed * 5
            self.camera.update(next_x, next_y)
```

```
            self.dialog_system.update(dt)

            if self.victory_achieved:
                self.victory_timer += dt

            if self.socket_active:
                self.send_data()

            # Отрисовка
            self.draw_background_with_parallax(self.screen)
            platform_y = SCREEN_HEIGHT - PLATFORM_HEIGHT
            self.screen.blit(self.platform, (0, platform_y))

            for platform in self.platforms:
                platform.draw(self.screen, self.camera)
            for platform in self.moving_platforms:
                platform.draw(self.screen, self.camera)
            for collectible in self.animated_keys:
                collectible.draw(self.screen, self.camera)
            for collectible in self.potions:
                collectible.draw(self.screen, self.camera)
            for lamp in self.lamps:
                lamp.draw(self.screen, self.camera)
            for decoration in self.decorations:
                decoration.draw(self.screen, self.camera)
            for sign in self.signs:
                sign.draw(self.screen, self.camera)

            alice = self.my_player if self.my_player.character_name == "alice" else self.other_player
            rabbit = self.other_player if self.my_player.character_name == "alice" else self.my_player

            rabbit.draw(self.screen, self.camera)
            alice.draw(self.screen, self.camera)

            self.screen.blit(self.shadow, (0, 0))
            self.dialog_system.draw(self.screen, self.my_player.character_name)
            self.draw_ui(self.screen)
            self.draw_victory_screen(self.screen)

            pygame.display.flip()
            self.clock.tick(60)

        except:
            self.is_shutting_down = True
        finally:
            self.close()
            pygame.quit()


    def handle_input(self, event):
        """Обработка ввода для диалогов"""
        if not self.dialog_system.is_active:
            return

        choice = self.dialog_system.handle_input(event, self.my_player.character_name)
        if choice is not None and self.dialog_system.my_turn:
            print(f"Выбран вариант {choice + 1}")
            self.make_choice(choice)

    def check_dialog_trigger(self):
        """Проверка возможности начать диалог"""
        # Вычисляем расстояние между персонажами
```

```
    distance = ((self.my_player.x - self.other_player.x) ** 2 +
            (self.my_player.y - self.other_player.y) ** 2) ** 0.5

    # Если игроки достаточно близко и диалог не активен и не завершен
    if (distance < DIALOG_TRIGGER_DISTANCE and
        not self.dialog_system.is_active and
        not self.dialog_system.dialog_completed and
        self.my_player.character_name == "alice"):

            self.start_dialog("start")

def start_dialog(self, dialog_id):
    """Начало диалога"""
    if not self.dialog_system.dialog_completed:
        print(f"Начинаем диалог: {dialog_id}")
        self.dialog_system.start_dialog(dialog_id, self.my_player.character_name)
        self.dialog_state["is_active"] = True
        self.dialog_state["current_dialog_id"] = dialog_id
        self.dialog_state["current_speaker"] = "alice"

        # Отправляем сообщение о начале диалога
        try:
            data = pickle.dumps({
                "type": "dialog_update",
                "dialog_state": self.dialog_state
            })
            self.socket.sendto(data, self.other_address)
        except Exception as e:
            print(f"Ошибка при отправке начала диалога: {e}")

def end_dialog(self):
    """Завершение диалога"""
    print("Завершаем диалог")
    self.dialog_system.complete_dialog()
    self.dialog_state["is_active"] = False
    self.dialog_state["current_dialog_id"] = None
    self.dialog_state["current_speaker"] = None
    self.dialog_state["dialog_completed"] = True

    # Отправляем сообщение о завершении диалога
    try:
        data = pickle.dumps({
            "type": "dialog_end",
            "dialog_state": self.dialog_state
        })
        self.socket.sendto(data, self.other_address)
    except Exception as e:
        print(f"Ошибка при отправке завершения диалога: {e}")

def draw_ui(self, screen):
    """Отрисовка пользовательского интерфейса"""
    # Показываем UI только после завершения диалога
    if not self.dialog_system.dialog_completed:
        return

    # Используем пиксельный шрифт
    try:
        pixel_font = pygame.font.Font(os.path.join("assets", "fonts", "visitor2.otf"), 20)
    except:
        # Fallback на системный шрифт
        pixel_font = pygame.font.Font(None, 20)

    # Показываем количество собранных ключей
```

```
        keys_text = pixel_font.render(f"Ключи: {self.collected_keys}/3", True, (255, 255, 255))
        screen.blit(keys_text, (10, 10))

        # Показываем количество собранных зелий
        potions_text = pixel_font.render(f"Зелья: {self.collected_potions}/3", True, (255, 255, 255))
        screen.blit(potions_text, (10, 35))

        # Показываем прогресс и подсказки
        if self.collected_keys == 3 and self.collected_potions == 3:
            victory_text = pixel_font.render("Все предметы собраны! Найдите выход из норы!", True, (0, 255, 0))
            screen.blit(victory_text, (10, SCREEN_HEIGHT - 30))
        else:
            hint_text = pixel_font.render("Соберите все предметы, чтобы активировать выход!", True, (255, 200, 0))
            screen.blit(hint_text, (10, SCREEN_HEIGHT - 30))

    def draw_background_with_parallax(self, screen):
        """Отрисовывает многослойный фон с эффектом параллакса"""
        # Вычисляем смещение для параллакса
        parallax_factor_1 = 0.1  # Самый дальний слой
        parallax_factor_2 = 0.3
        parallax_factor_3 = 0.5
        parallax_factor_4 = 0.7  # Ближайший слой

        # Вычисляем смещения
        offset_1 = int(self.camera.scroll_x * parallax_factor_1)
        offset_2 = int(self.camera.scroll_x * parallax_factor_2)
        offset_3 = int(self.camera.scroll_x * parallax_factor_3)
        offset_4 = int(self.camera.scroll_x * parallax_factor_4)

        # Заполняем экран черным цветом как базовый фон
        screen.fill((0, 0, 0))

        # Отрисовываем слои с параллаксом, если они доступны
        if 'layer1' in self.background:
            screen.blit(self.background['layer1'], (-offset_1, 0))
            if offset_1 > 0:
                screen.blit(self.background['layer1'], (SCREEN_WIDTH - offset_1, 0))

        if 'layer2' in self.background:
            screen.blit(self.background['layer2'], (-offset_2, 0))
            if offset_2 > 0:
                screen.blit(self.background['layer2'], (SCREEN_WIDTH - offset_2, 0))

        if 'layer3' in self.background:
            screen.blit(self.background['layer3'], (-offset_3, 0))
            if offset_3 > 0:
                screen.blit(self.background['layer3'], (SCREEN_WIDTH - offset_3, 0))

        if 'layer4' in self.background:
            screen.blit(self.background['layer4'], (-offset_4, 0))
            if offset_4 > 0:
                screen.blit(self.background['layer4'], (SCREEN_WIDTH - offset_4, 0))

class Flag:
    def __init__(self, x, y, flag_type="top"):
        self.x = x
        self.y = y
        self.flag_type = flag_type  # "top" или "bottom"
        self.width = 32
        self.height = 64
        self.rect = pygame.Rect(x, y, self.width, self.height)
        self.load_texture()
```

```
    def load_texture(self):
        """Загружает текстуру флага"""
        try:
            # Используем текстуру лампы как основу для флага
            self.texture = pygame.image.load(os.path.join("assets", "tiles", "oak_woods_v1.0", "decorations",
"lamp.png")).convert_alpha()
            self.texture = pygame.transform.scale(self.texture, (self.width, self.height))

            # Создаем флаг поверх лампы
            flag_surface = pygame.Surface((self.width, self.height), pygame.SRCALPHA)
            flag_surface.blit(self.texture, (0, 0))

            # Добавляем флаг
            flag_color = (255, 0, 0) if self.flag_type == "top" else (0, 0, 255)
            pygame.draw.rect(flag_surface, flag_color, (self.width//2, 5, 20, 15))
            pygame.draw.polygon(flag_surface, flag_color, [(self.width//2 + 20, 5), (self.width//2 + 20, 12),
(self.width//2 + 25, 10)])

            self.texture = flag_surface
        except:
            # Fallback - простой флаг
            self.texture = pygame.Surface((self.width, self.height), pygame.SRCALPHA)
            pygame.draw.rect(self.texture, (139, 69, 19), (self.width//2 - 2, 20, 4, 40))  # Столб
            flag_color = (255, 0, 0) if self.flag_type == "top" else (0, 0, 255)
            pygame.draw.rect(self.texture, flag_color, (self.width//2, 5, 20, 15))

    def draw(self, screen, camera):
        """Отрисовка флага"""
        screen_x, screen_y = camera.apply(self.x, self.y)
        if -self.width <= screen_x <= SCREEN_WIDTH and -self.height <= screen_y <= SCREEN_HEIGHT:
            screen.blit(self.texture, (screen_x, screen_y))

class Decoration:
    def __init__(self, x, y, decoration_type="grass"):
        self.x = x
        self.y = y
        self.decoration_type = decoration_type
        self.width = 32
        self.height = 32
        self.rect = pygame.Rect(x, y, self.width, self.height)
        self.load_texture()

    def load_texture(self):
        """Загружает текстуру декорации"""
        try:
            if self.decoration_type == "grass1":
                self.texture = pygame.image.load(os.path.join("assets", "tiles", "oak_woods_v1.0", "decorations",
"grass_1.png")).convert_alpha()
            elif self.decoration_type == "grass2":
                self.texture = pygame.image.load(os.path.join("assets", "tiles", "oak_woods_v1.0", "decorations",
"grass_2.png")).convert_alpha()
            elif self.decoration_type == "grass3":
                self.texture = pygame.image.load(os.path.join("assets", "tiles", "oak_woods_v1.0", "decorations",
"grass_3.png")).convert_alpha()
            elif self.decoration_type == "rock1":
                self.texture = pygame.image.load(os.path.join("assets", "tiles", "oak_woods_v1.0", "decorations",
"rock_1.png")).convert_alpha()
            elif self.decoration_type == "rock2":
                self.texture = pygame.image.load(os.path.join("assets", "tiles", "oak_woods_v1.0", "decorations",
"rock_2.png")).convert_alpha()
            elif self.decoration_type == "rock3":
                self.texture = pygame.image.load(os.path.join("assets", "tiles", "oak_woods_v1.0", "decorations",
"rock_3.png")).convert_alpha()
```

```
            elif self.decoration_type == "fence":
                self.texture = pygame.image.load(os.path.join("assets", "tiles", "oak_woods_v1.0", "decorations",
"fence_1.png")).convert_alpha()
            elif self.decoration_type == "fence2":
                self.texture = pygame.image.load(os.path.join("assets", "tiles", "oak_woods_v1.0", "decorations",
"fence_2.png")).convert_alpha()

            # Масштабируем декорацию
            self.texture = pygame.transform.scale(self.texture, (self.width, self.height))
        except:
            self.texture = None

    def draw(self, screen, camera):
        """Отрисовка декорации"""
        screen_x, screen_y = camera.apply(self.x, self.y)
        if -self.width <= screen_x <= SCREEN_WIDTH and -self.height <= screen_y <= SCREEN_HEIGHT:
            if self.texture:
                screen.blit(self.texture, (screen_x, screen_y))
            else:
                # Fallback - простая декорация
                color = (34, 139, 34) if "grass" in self.decoration_type else (139, 69, 19)
                pygame.draw.rect(screen, color, (screen_x, screen_y, self.width, self.height))

    def create_flags(self):
        """Создает флаги для финала"""
        flags = []

        # Верхний флаг на высокой платформе зайца (2200, 450)
        flags.append(Flag(2220, 386, "top"))  # На платформе 2200, 450 - высота флага 64

        # Нижний флаг под платформой
        ground_level = WORLD_HEIGHT - PLATFORM_HEIGHT - 64
        flags.append(Flag(2220, ground_level, "bottom"))

        return flags

    def create_signs(self):
        """Создает знаки"""
        signs = []

        # Только два знака в самом конце карты
        ground_level = WORLD_HEIGHT - PLATFORM_HEIGHT - 48

        # Знак на высокой платформе (для зайца)
        signs.append(Sign(2850, 550))  # На платформе 2800, 600

        # Знак на земле (для Алисы)
        signs.append(Sign(2850, ground_level))  # На земле

        return signs

class Sign:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.width = 32
        self.height = 48
        self.rect = pygame.Rect(x, y, self.width, self.height)
        self.detection_radius = 80  # Радиус обнаружения персонажей
        self.is_player_near = False
        self.load_texture()

    def load_texture(self):
```

```
        """Загружает текстуру знака"""
        try:
            self.texture = pygame.image.load(os.path.join("assets", "tiles", "oak_woods_v1.0", "decorations",
"sign.png")).convert_alpha()
            self.texture = pygame.transform.scale(self.texture, (self.width, self.height))
        except:
            # Fallback - простой знак
            self.texture = pygame.Surface((self.width, self.height), pygame.SRCALPHA)
            # Столб
            pygame.draw.rect(self.texture, (139, 69, 19), (self.width//2 - 2, 30, 4, 18))
            # Табличка
            pygame.draw.rect(self.texture, (160, 82, 45), (4, 8, 24, 16))
            pygame.draw.rect(self.texture, (0, 0, 0), (4, 8, 24, 16), 2)

    def update(self, players):
        """Обновляет состояние знака в зависимости от близости игроков"""
        self.is_player_near = False

        for player in players:
            distance = ((player.x - self.x) ** 2 + (player.y - self.y) ** 2) ** 0.5
            if distance <= self.detection_radius:
                self.is_player_near = True
                break

    def draw(self, screen, camera):
        """Отрисовка знака с размытой подсветкой"""
        screen_x, screen_y = camera.apply(self.x, self.y)
        if -self.width <= screen_x <= SCREEN_WIDTH and -self.height <= screen_y <= SCREEN_HEIGHT:
            # Создаем размытое свечение
            glow_color = (0, 255, 0) if self.is_player_near else (255, 0, 0)

            # Создаем несколько слоев свечения для эффекта размытия
            glow_sizes = [40, 30, 20, 10]
            glow_alphas = [30, 50, 80, 120]

            for size, alpha in zip(glow_sizes, glow_alphas):
                glow_surface = pygame.Surface((self.width + size, self.height + size), pygame.SRCALPHA)
                glow_rect = pygame.Rect(0, 0, self.width + size, self.height + size)

                # Создаем градиентное свечение
                for i in range(size // 2):
                    current_alpha = max(0, alpha - (i * alpha // (size // 2)))
                    current_color = (*glow_color, current_alpha)

                    inner_rect = pygame.Rect(i, i, self.width + size - 2*i, self.height + size - 2*i)
                    if inner_rect.width > 0 and inner_rect.height > 0:
                        pygame.draw.ellipse(glow_surface, current_color, inner_rect)

                screen.blit(glow_surface, (screen_x - size//2, screen_y - size//2))

            # Отрисовываем сам знак
            if self.texture:
                screen.blit(self.texture, (screen_x, screen_y))
            else:
                # Fallback
                pygame.draw.rect(screen, (139, 69, 19), (screen_x + self.width//2 - 2, screen_y + 30, 4, 18))
                pygame.draw.rect(screen, (160, 82, 45), (screen_x + 4, screen_y + 8, 24, 16))
                pygame.draw.rect(screen, (0, 0, 0), (screen_x + 4, screen_y + 8, 24, 16), 2)
```