Федеральное государственное образовательное бюджетное учреждение высшего образования

**«Финансовый университет при Правительстве Российской Федерации»**

**(Финансовый университет)**

Колледж информатики и программирования

Дипломный проект

Тема «Разработка компьютерной 3D игры на Unity “Chief’s World”»

Ученик Шатилов Арсений Олегович

Учебная группа 4ПКС-120

Специальность 09.02.03 Программирование в компьютерных системах

|  |  |  |
| --- | --- | --- |
| Руководитель  дипломного проекта |  | Сибирев И.В. |
|  | (подпись) | (инициалы, фамилия) |
| Председатель предметной (цикловой) комиссии |  |  |
|  | (подпись) | (инициалы, фамилия) |

Москва – 2024 г

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc168023242)

[Глава 1. Предпроектное исследование 6](#_Toc168023243)

[1.1. Описание предметной области 6](#_Toc168023244)

[1.2. Сравнительный анализ программ-аналогов 7](#_Toc168023245)

[1.3. Постановка задачи 8](#_Toc168023246)

[1.4. Характеристика инструментальных средств разработки 10](#_Toc168023247)

[Глава 2. Проектирование и реализация программы 12](#_Toc168023248)

[2.1. Анализ требований и определение спецификаций программного обеспечения 12](#_Toc168023249)

[2.2. Проектирование программного обеспечения 14](#_Toc168023250)

[2.3. Разработка программного обеспечения 15](#_Toc168023251)

[2.4. Откладка и тестирование программы 22](#_Toc168023252)

[2.5. Руководство по использованию программы 27](#_Toc168023253)

[ЗАКЛЮЧЕНИЕ 32](#_Toc168023254)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ИНТЕРНЕТ-РЕСУРСОВ 34](#_Toc168023255)

[ПРИЛОЖЕНИЕ 37](#_Toc168023256)

# **ВВЕДЕНИЕ**

В ходе выполнения дипломного проекта, представилась уникальная возможность исследовать процесс создания компьютерных игр. Игровая индустрия, столь важная для современного цифрового общества, продолжает расширять границы развлечений, влияя на культуру и предоставляя новые способы взаимодействия и обучения. Игры, воплощающие творческие и инновационные идеи, требуют глубокого понимания как технических, так и дизайнерских аспектов их разработки. В рамках проекта, сосредоточенного на создании 3D игры "Chef's World" в среде Unity, особое внимание уделяется слиянию аспектов геймификации и образования, что предоставляет игрокам уникальный опыт, в рамках которого они могут изучать кулинарное искусство, развиваясь и развлекаясь одновременно. Unity, как мощная и гибкая платформа для разработки игр, предоставляет инструменты для создания многоуровневых и динамичных игровых миров, усиливая эмоциональный отклик и удержание внимания игроков.

Цель проекта: создание компьютерной 3D игры «Chef's World» на платформе Unity. Основной задачей проекта является разработка увлекательного и качественного игрового приложения, в котором игрок мог бы погрузиться в уникальный мир кулинарных приключений. Проект предполагает использование языка программирования C# для реализации функционала.

Для успешной реализации поставленных целей необходимо решить следующие задачи:

1. Проанализировать особенности разработки компьютерных игр в жанре кулинарии и изучение ведущих представителей данного жанра.
2. Выбрать инструментальные средства для разработки игры на базе Unity, включая работу с анимацией, звуковым оформлением и визуальными эффектами.
3. Определить функциональные и технические требования к игре, включая создание атмосферы кулинарного восторга, игровой механики, управления персонажем и врагами.
4. Спроектировать и реализовать сценарную логику игры, включая создание уровней, сценариев и взаимодействия персонажей.
5. Спроектировать и реализовать визуальную составляющую игры, включая модели персонажей, анимации, текстуры и освещение.
6. Разработать звуковое сопровождение и аудиоэффекты, создающие атмосферу кулинарного мира.
7. Протестировать игру на выявление и устранение возможных ошибок и недочетов.
8. Подготовить пользовательскую документацию, включая описание геймплея, управление и рекомендации по оптимальным настройкам.

Объектом дипломного проекта является создание компьютерной 3D игры на Unity под названием «Chef's World».

Исследовательская часть проекта включает в себя следующие аспекты:

1. Эффективность интерактивных методов обучения в контексте кулинарного мира. Анализ применимости визуализации, тестирования и интерактивных форм для создания увлекательного опыта и понимания кулинарных концепций.
2. Влияние игровой системы на вовлеченность игроков. Исследование, как игровой процесс влияет на мотивацию и результаты игроков в изучении и применении кулинарных навыков.
3. Технические аспекты. Исследование производительности игры, оптимизация кода и обеспечение стабильной работы для обеспечения качественного игрового опыта.

Методы разработки включают в себя:

1. Метод проектирования интерфейса. Разработка дизайна игры, учитывающего потребности игроков и создающего привлекательное визуальное воздействие.
2. Метод создания системы управления контентом. Реализация функционала для управления и структурирования уровней, кулинарных задач и интерактивных элементов.
3. Метод создания игровой системы. Разработка механизмов для создания уровней, сценариев и взаимодействия персонажей.
4. Метод тестирования и отладки. Проведение проверок функционала для выявления и устранения возможных ошибок.

В современной индустрии развлечений, где кулинарные игры выделяются своей востребованностью и уникальностью, проект "Chef's World" нацелен на заполнение ниши в этом жанре, предлагая не только развлечение, но и образовательный аспект. Источниками для исследования послужили многочисленные онлайн-платформы, профессиональные форумы и сообщества разработчиков, где обсуждаются передовые практики и инновационные методы создания игрового контента.

Для обеспечения актуальности и востребованности предложенной игры, был проведен тщательный анализ трендов игровой индустрии, исследования предпочтений целевой аудитории и изучены успешные кейсы известных кулинарных игр. В дополнение, важным аспектом является внедрение современных графических и программных решений, для чего Unity как платформа для разработки является идеальным инструментом, позволяющим сочетать высокое качество исполнения с широкими возможностями для творчества.

# **Глава 1. Предпроектное исследование**

## **1.1. Описание предметной области**

Предметной областью дипломного проекта является разработка компьютерной 3D игры на Unity под названием «Chef's World». Проект направлен на создание увлекательного игрового пространства, где игроки могут погрузиться в захватывающий мир кулинарных приключений. В контексте разработки игры, рассмотрим ключевые аспекты данной предметной области:

1. Эволюция игровой индустрии: Анализ современных тенденций в игровой индустрии, включая использование новых технологий, тренды в дизайне игр и популярные жанры.
2. Интерактивные методы развлечения: Исследование педагогических принципов и методов, использующих визуализацию, тестирование и другие интерактивные подходы для улучшения вовлеченности игроков и понимания кулинарных концепций в виртуальном мире Chef's World.
3. Оценка эффективности геймдизайна: Анализ результатов исследований, посвященных влиянию геймдизайна на уровень удовлетворенности игроков и их интерес к игровому процессу.
4. Требования к игровым платформам: Разбор требований и предпочтений игроков от современных игровых платформ, включая функционал для создания уровней, управления персонажем и визуализации кулинарных элементов.
5. Технологии в разработке игр: Ознакомление с современными подходами в разработке игр, такими как использование искусственного интеллекта, виртуальной реальности и других технологий, которые могут быть интегрированы для создания качественного игрового опыта.
6. Вовлечение игроков: Анализ потребностей современных геймеров и поиск инновационных решений для улучшения качества игрового процесса и создания захватывающего виртуального мира.
7. Лучшие практики в разработке игр: Изучение опыта успешных разработчиков игр, выявление лучших практик и преодоление вызовов в процессе создания игрового контента.

Исследование предметной области игровой разработки для "Chef's World" подчеркивает его важность в формировании успешного проекта, обогащая разработку глубоким пониманием трендов и ожиданий аудитории. Анализ современных технологий и игровых тенденций, включая интерактивное обучение и геймдизайн, наряду с оценкой требований к платформам и технологиям, открывает перед "Chef's World" возможности для создания инновационного и вовлекающего игрового опыта. Это исследование не только выявляет ключевые направления для развития игры, но и подчеркивает необходимость интеграции современных решений для достижения высокой удовлетворенности игроков и образовательной ценности.

## **1.2. Сравнительный анализ программ-аналогов**

Прежде чем приступать к разработке, проведем анализ и сравним существующие компьютерные игры в жанре 3D кулинарных приключений в таблице 1. Были рассмотрены следующие игры: "Overcooked" на Unity, "Cooking Simulator" на Unreal Engine 4 и "Battle Chef Brigade" с использованием собственного движка.

Таблица 1. Сравнительный анализ

|  |  |  |  |
| --- | --- | --- | --- |
| Характеристика | Overcooked (Unity) | Cooking Simulator (Unreal Engine 4) | Battle Chef Brigade (Custom Engine) |
| Графический стиль | Стилизованная 3D графика | Реалистичная 3D графика | Рисованная 2D графика, 3D анимации |
| Сложность геймплея | Легкий, ориентированный на широкую аудиторию | Средний, более реалистичный подход | Средний, акцент на геймплейном сборе |
| Технические требования | Менее требовательные | Более высокие требования к оборудованию | Менее требовательные |
| Интерактивные элементы | Динамичные кулинарные задачи, кооператив | Взаимодействие с различными кулинарными инструментами | Пазловые битвы с готовкой и поеданием |

Продолжение таблицы 1

|  |  |  |  |
| --- | --- | --- | --- |
| Характеристика | Overcooked (Unity) | Cooking Simulator (Unreal Engine 4) | Battle Chef Brigade (Custom Engine) |
| Особенные механики игры | Кулинарные задачи с управлением несколькими персонажами | Реалистичная физика приготовления, открытый мир кулинарии | Сочетание экшна и готовки, пазловые элементы |
| Поддерживаемые платформы | PC, консоли, мобильные устройства | PC | PC, консоли |
| Стоимость игры | Платная | Платная | Платная |

Каждая из перечисленных игр имеет свои сильные и слабые стороны, и выбор зависит от конкретных предпочтений и ожиданий игроков. Например, "Overcooked" привлекает своей простотой и возможностью играть в кооперативе, что делает её подходящей для игры в компании друзей или семьи. "Cooking Simulator" предлагает более реалистичный опыт приготовления пищи, что может быть интересным для игроков, увлеченных кулинарией. "Battle Chef Brigade" объединяет элементы кулинарии и экшна, предлагая уникальный игровой процесс, который сочетает в себе готовку и боевые действия.

Таким образом, каждый проект обладает уникальными чертами, что позволяет разнообразить игровой процесс и привлечь различную аудиторию.

## **1.3. Постановка задачи**

Для программной реализации необходимо определить входные и выходные данные 3D игры "Chef's World" на платформе Unity. Входными данными являются действия игрока, взаимодействие с окружением, управление персонажем и использование игровых механик. Выходные данные представлены визуальным и звуковым контентом, создаваемым игрой в ответ на действия игрока.

Программа должна обладать следующими функциями:

1. Управление персонажем: Возможность перемещения персонажа в игровом мире с использованием стандартных управляющих устройств.
2. Взаимодействие с окружением: Возможность взаимодействия с объектами в игровом мире, приготовление блюд, декорирование и украшение кухни.
3. Искусственный интеллект персонажей: Реализация интеллекта для персонажей, с которыми игрок взаимодействует в игровом процессе.
4. Звуковое сопровождение: Использование забавных и поддерживающих звуков для создания позитивной атмосферы игры.
5. Графика и дизайн: Использование яркой и привлекательной графики, соответствующей детскому стилю.
6. Интерфейс и пользовательский опыт: Разработка интуитивно понятного пользовательского интерфейса с отображением важной информации. Добавление элементов управления, таких как кнопки действий, меню паузы и настроек.
7. Тестирование и отладка: Проведение тестирования игры на наличие ошибок и недочетов. Исправление багов и улучшение общей производительности игры.

Входные данные включают действия игрока, взаимодействие с окружением, управление персонажем и использование игровых механик.

Выходные данные представлены визуальным и звуковым контентом, создаваемым игрой в ответ на действия игрока.

Для программы установлены следующие требования к реализации: (см. табл. 2):

Таблица 2. Требования реализации

|  |  |
| --- | --- |
| Функциональные требования | Эксплуатационные (нефункциональные) требования |
| Игра должна позволять пользователю готовить различные блюда по заказам клиентов. | Игра должна обеспечивать высокую производительность и минимальное время загрузки. |
| Пользователь должен иметь возможность взаимодействовать с кухонными предметами и ингредиентами. | Игра должна поддерживать возможность обновления для исправления ошибок и добавления новых функций. |

Продолжение таблицы 2

|  |  |
| --- | --- |
| Функциональные требования | Эксплуатационные (нефункциональные) требования |
| Игра должна включать пять основных рецептов, которые игрок может готовить. | Игра должна иметь удобный и интуитивно понятный интерфейс, обеспечивающий легкость в навигации и использовании. |
| Пользователь должен иметь возможность отслеживать прогресс выполнения заказов и получать обратную связь о качестве приготовленных блюд. | Игра должна иметь минимальные системные требования для работы на большинстве современных компьютеров. |

Требования к надежности программного обеспечения:

1. Стабильность: Игра должна обеспечивать стабильную работу без зависаний и вылетов, чтобы предотвратить потерю данных и создать позитивный опыт игры.
2. Сохранение прогресса: Реализация надежной системы сохранения игрового прогресса для предотвращения потери данных пользователя при выходе из игры или других событиях.
3. Совместимость с аппаратным обеспечением: Игра должна корректно функционировать на различных конфигурациях аппаратного обеспечения, обеспечивая приемлемую производительность и графику.
4. Эффективное управление памятью: Оптимизированное использование оперативной памяти для предотвращения перегрузки системы и обеспечения плавного игрового процесса.

## **1.4. Характеристика инструментальных средств разработки**

Для разработки компьютерной 3D игры на платформе Unity “Chef's World” будут применены разнообразные инструментальные средства, включая языки программирования, графические ресурсы и средства разработки. Рассмотрим характеристику некоторых из них:

1. Язык программирования:

- C#: Основной язык программирования для разработки игр на платформе Unity. C# обеспечивает широкие возможности для создания игровой логики, управления объектами и взаимодействия с пользовательским вводом.

1. Графические ресурсы:

- Blender: Мощный инструмент для моделирования 3D объектов, создания анимаций и текстурирования. Blender предоставляет возможности для разработки разнообразных игровых элементов, от персонажей до окружения.

1. Среда разработки (IDE):

- Unity Editor: Официальная интегрированная среда разработки для создания игр на платформе Unity. Unity Editor обеспечивает удобный интерфейс для компоновки сцен, редактирования кода, настройки физики и тестирования игры.

1. Звуковые ресурсы:

- Audacity: Программа для редактирования и обработки звуковых файлов. Audacity позволяет создавать и редактировать звуковые эффекты, музыку и диалоги, необходимые для обогащения звукового оформления игры.

Использование указанных инструментов обеспечит эффективную разработку 3D игры на Unity “Chef's World”. Использование языка C#, Blender, Unity Editor и Audacity оказалось решающим для создания игры "Chef's World" с качественной графикой, звуком и геймплеем. Каждый инструмент играет свою уникальную роль: C# служит для написания надежного кода, Blender помогает визуализировать задуманный мир, Unity Editor эффективно объединяет все части в единое целое, а Audacity дарит глубину звуковому оформлению. Такая синергия обеспечивает баланс между технической отзывчивостью и артистическим выражением, необходимый для современной 3D игры.

# **Глава 2. Проектирование и реализация программы**

## **2.1. Анализ требований и определение спецификаций программного обеспечения**

Для более ясного представления архитектуры и функционала игры, приведены диаграммы вариантов использования. Эти диаграммы помогают визуализировать взаимодействие пользователя с игровым процессом, а также отображают, как система реагирует на действия пользователя. Схематичное изображение процессов и функций игры выполнено с использованием онлайн-инструментов Draw.io и Plantuml.com, что облегчает понимание структуры проекта и упрощает процесс разработки.

Основные возможности, доступные пользователю в игре, отображены на диаграмме вариантов использования, представленной на рисунке 1.

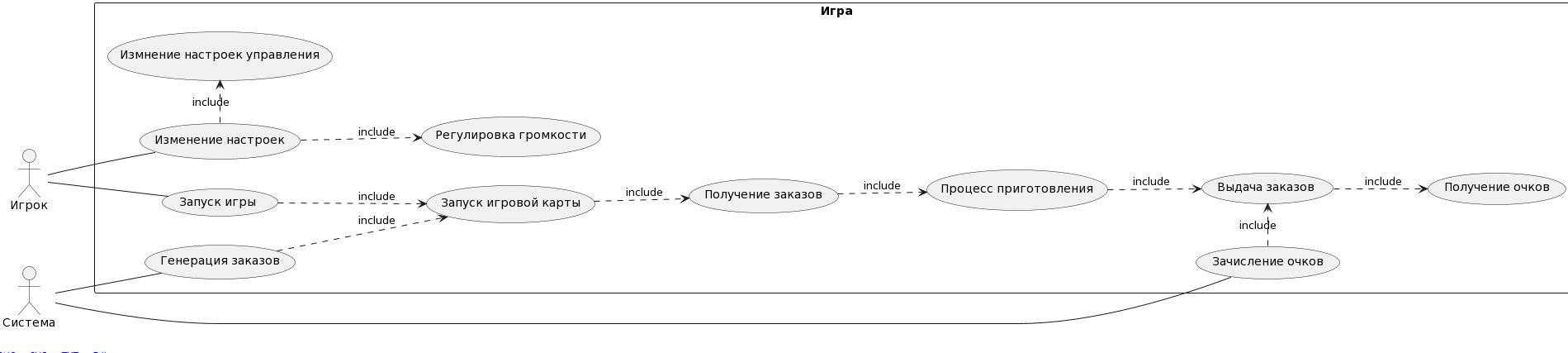


Рисунок 1. Диаграмма вариантов использования

Краткое пояснение диаграммы:

- Игрок может запустить игру или изменить настройки.

- После запуска игры система генерирует заказы, которые игрок выполняет, следуя цепочке действий вплоть до получения очков.

- В изменении настроек доступны регулировка громкости и изменение настроек управления.

Функциональная диаграмма верхнего уровня представлена на рисунке 2.

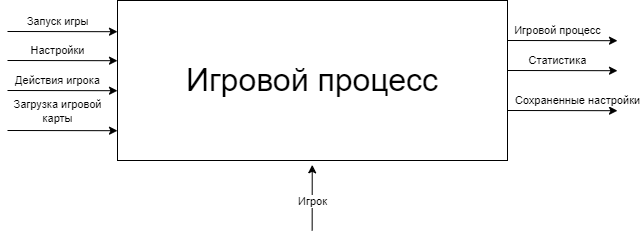


Рисунок 2. Функциональная диаграмма верхнего уровня

Диаграмма последовательности представлена на рисунке 3.

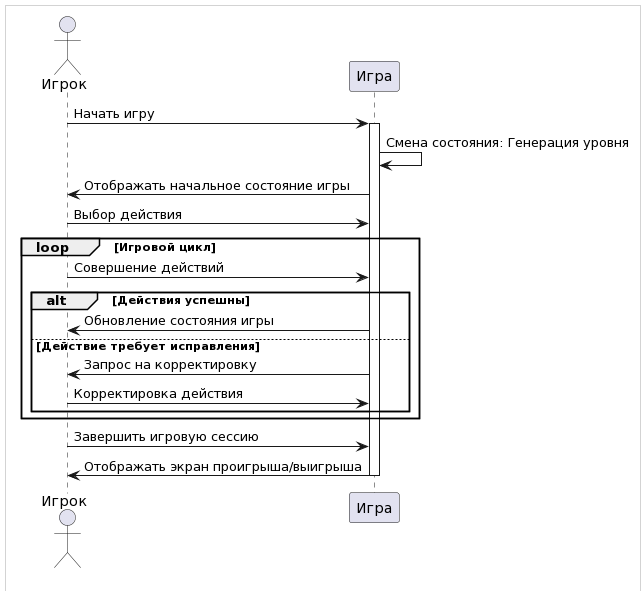


Рисунок 3. Диаграмма последовательностей

Эта диаграмма последовательности представляет собой сценарий игры, где:

- Игрок начинает игру.

- Игра переходит в состояние генерации уровня.

- Игроку отображается начальное состояние игры.

- Игрок выбирает действия в игре, которые затем обрабатываются игрой.

- В цикле игрового процесса игрок выполняет серию действий, на что игра реагирует обновлением состояния или запросом на корректировку.

- По завершении игры игроку отображается экран проигрыша или выигрыша.

## 2.2. Проектирование программного обеспечения

Чтобы обеспечить полное понимание структуры игры, представляется целесообразным разработать две основные схемы: структурную и функциональную.

Структурная схема будет визуализировать архитектуру игры, выявляя взаимосвязи между её компонентами. Это позволит разработчикам и аналитикам легче ориентироваться по коду, понимать зависимости и модульное строение программного продукта.

Функциональная схема, с другой стороны, концентрируется на потоках данных внутри игры и описывает, как функции и подсистемы взаимодействуют между собой при выполнении определённых задач. Она отображает логику обработки данных и реакции системы на входы пользователя, что позволяет более глубоко понять внутреннюю работу программного решения.

Совместное использование структурной и функциональной схем даст нам полную картину, обеспечивая тем самым глубокое понимание как структуры, так и функционала игры. Это облегчит процесс разработки, позволит точно анализировать различные аспекты игры и эффективно оптимизировать её работу. На рисунках 4,5 изображены структурная и функциональная схемы.

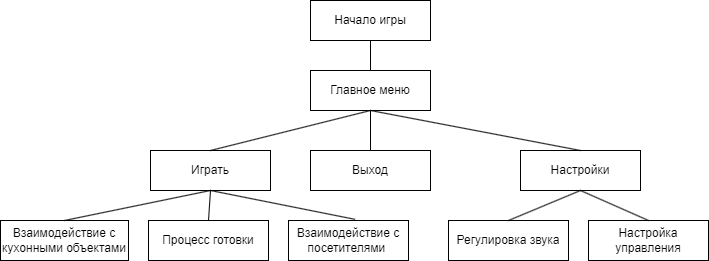


Рисунок 4. Структурная схема игры

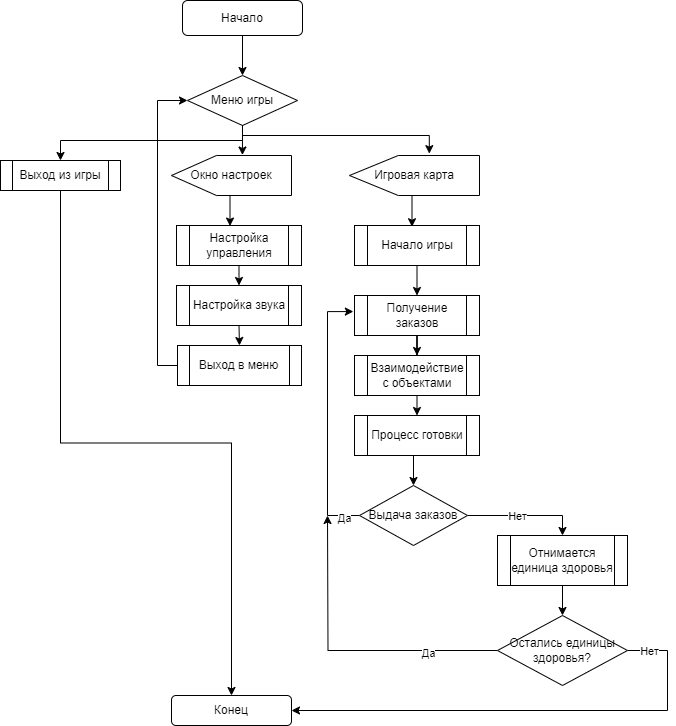


Рисунок 5. Функциональная схема игры

## **2.3. Разработка программного обеспечения**

В процессе разработки игры особое внимание было уделено разработке локации, так как правильно созданная атмосфера играет решающую роль в погружении пользователя в игровой мир. Я стремился к тому, чтобы каждая локация передавала уникальное настроение и стимулировала взаимодействие игрока с игровыми объектами, что важно для кулинарной тематики игры.

В качестве первого шага к этому была разработана стартовая локация, которая устанавливает основной тон игры. Всякий раз, когда игрок взаимодействует с объектами на кухне - будь то ингредиенты для блюд или кухонное оборудование, я хочу, чтобы эти взаимодействия были интуитивно понятными и при этом визуально насыщенными.

Для визуального представления и демонстрации качества работы, локация и её элементы, были изображены на рисунке 6.



Рисунок 6. Игровая локация

Также гарнитура и предметы взаимодействия являются одними из ключевых элементов, который обогащает игровой процесс и добавляет слой реализма. Он состоит из различных кухонных принадлежностей и инструментов, каждый из которых имеет своё функциональное назначение и влияет на игру.

Разработка гарнитуры требовала тщательного подхода к разработке. Инструменты и утварь не просто статичные объекты; они взаимодействуют с игроком, реагируя на его действия.

На рисунке 7 представлена модель кухонной плиты, она выполняет ключевую роль в процессе приготовления блюд. Кухонная плита позволяет игроку жарить мясо, обеспечивая реалистичное и увлекательное взаимодействие с кулинарным процессом. Визуальные и звуковые эффекты, сопровождающие процесс жарки, создают атмосферу настоящей кухни, погружая игрока в мир профессионального повара.



Рисунок 7. Кухонная плита

Код кухонной плиты представлен в Приложении №1.

Стол с разделочной доской в игре является незаменимым инструментом для подготовки ингредиентов. Этот стол представляет собой рабочую поверхность, где игрок может нарезать разнообразные продукты, такие как помидоры, сыр и салат. Процесс нарезки сопровождается анимацией и звуковыми эффектами, что усиливает погружение в игру. На рисунке 8 представлена модель стола с разделочной доской.



Рисунок 8. Стол с разделочной доской

Код разделочной доски представлен в Приложении №1.

Предмет взаимодействия «Контейнер хранения», представленный на рисунке 9, предназначен для удобного хранения ингредиентов, необходимых для приготовления блюд. Всего в игре предусмотрено четыре таких контейнера, каждый из которых выполняет важную функцию в организации кухонного пространства. Игрок может взаимодействовать с контейнерами, извлекая из них необходимые продукты для дальнейшего использования в процессе приготовления блюд.



Рисунок 9. Контейнер хранения

Внимание к деталям персонажа является ключевым элементом для создания уникального и запоминающегося игрового опыта.

Для достижения высокого уровня детализации персонажа, были тщательно проработаны визуальные аспекты, такие как текстуры одежды и анимации движений при выполнении кулинарных операций. Все элементы, от поварской шапки до фартука, разработаны с учетом модных тенденций и функциональности кухонной одежды, что позволяет игрокам не только проявить свою индивидуальность, но и почувствовать себя настоящими мастерами кулинарного искусства. На рисунке 10 изображен персонаж игры.



Рисунок 10. Персонаж игры

Код персонажа игры представлен в Приложении №1.

Ключевую роль в динамике игрового процесса выполняют клиенты, создавая интересные и увлекательные задания для игрока.

Клиенты подходят к ресторану и делают заказы, отображая конкретные блюда, которые они хотят получить. На экране появляется визуальное представление рецепта заказа, включая все необходимые ингредиенты. Игроку необходимо следовать рецепту, подготавливая и собирая ингредиенты, чтобы создать заказанное блюдо.

Клиенты ожидают выполнения заказа в течение ограниченного времени. Если игрок не успевает выполнить заказ в течение 20 секунд, клиент уходит, что влияет на исход игры. Время ожидания добавляет элемент напряжения и стимулирует игрока к более быстрому и точному выполнению задач.

После того как игрок передает готовое блюдо клиенту, клиент оценивает его правильность. Визуальная обратная связь от клиента позволяет игроку понять, насколько точно он следовал рецепту и соответствовал ожиданиям. Это способствует развитию навыков и улучшению результатов в дальнейшем. На рисунке 11 представлен пришедший клиент, ожидающий своего заказа.



Рисунок 11. Клиент

Код клиента представлен в Приложении №1.

В игре присутствует система здоровья, представленная на рисунке 12. Индикатор выполнен в виде трёх сердец, которые отображаются на экране и символизируют жизни игрока.

Каждое сердце представляет одну жизнь. Если игрок неправильно выполняет заказ клиента, одна жизнь из индикатора убавляется. Это может произойти, если ингредиенты не соответствуют заказу, блюдо приготовлено неправильно или клиенту пришлось ждать слишком долго.

Когда игрок теряет все три жизни, игра заканчивается, и игроку необходимо начать заново, чтобы попытаться улучшить свои результаты и обслуживание клиентов.

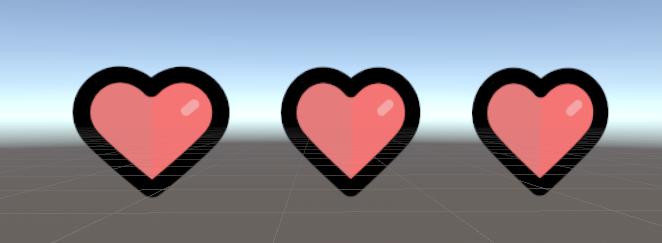


Рисунок 12. Индикатор здоровья

Код системы жизней представлен в Приложении №1.

В игре предусмотрены несколько панелей меню для удобного управления игровым процессом.

Главное меню (представлено на рисунке 13) включает кнопку "Играть", которая начинает новую игру, и кнопку "Выйти", завершающую игру. Когда игрок ставит игру на паузу, появляется панель паузы (рисунок 14), где можно выбрать "Продолжить" для возобновления игры, "Параметры" для открытия панели настроек, или "Главное меню" для возвращения в главное меню.

Панель настроек (рисунок 15) позволяет регулировать громкость звуков и музыки с помощью ползунков, а также настраивать управление. Для закрытия панели настроек и возврата к игре предусмотрена кнопка "Закрыть".



Рисунок 13. Главное меню

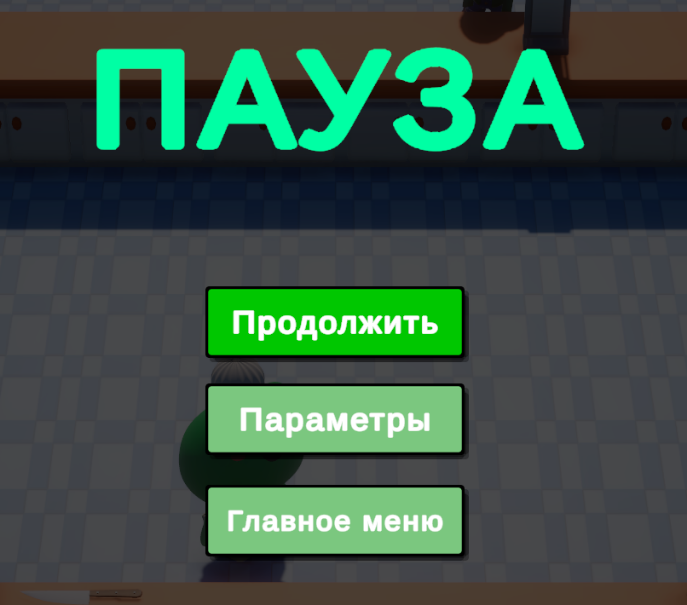


Рисунок 14. Панель паузы

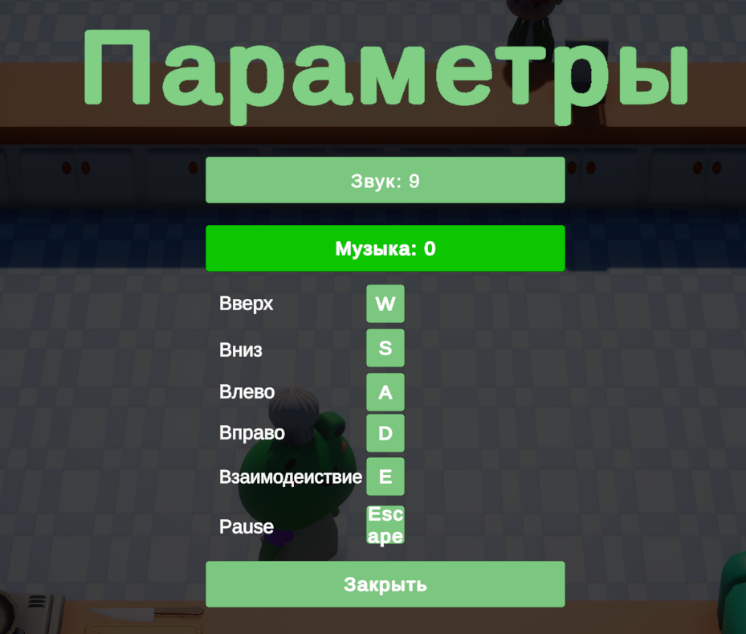


Рисунок 15. Окно настроек

## **2.4. Откладка и тестирование программы**

Тестирование и отладка - важнейшие этапы разработки игры, которые обеспечивают её высокое качество и надежность. В ходе тестирования проводится детальная проверка всех игровых компонентов на наличие ошибок, дефектов и на соответствие установленным стандартам. Отладка позволяет выявить и устранить любые программные недостатки, что гарантирует плавный и стабильный игровой процесс.

В рамках тестирования были тщательно проверены все функциональные модули игры.

Для механики нарезки была проведена проверка точности нарезки различных ингредиентов, таких как помидоры, сыр и салат, с помощью ножа на разделочной доске. Тестирование включало проверку корректности выполнения этой операции и визуального отображения нарезанных продуктов. На изображениях видно, как игрок взаимодействует с капустой (см. рис. 16), и результат нарезки отображается на разделочной доске (см. рис. 17).



Рисунок 16. Отображение анимации нарезания продукта



Рисунок 17. Отображение полученного результата

Для механики жарки была проведена проверка процесса жарки мяса на кухонной плите, где игрок должен правильно жарить мясо, следя за временем. Тестирование включало проверку всех этапов жарки - от начального нагрева до готовности блюда. На изображениях демонстрируется процесс жарки мяса, от начального этапа до готового продукта (см. рис. 18 – 19). Кроме того, проверялась механика пережарки мяса, чтобы при длительной жарке мясо пережаривалось, что добавляет реалистичности и сложности в игру (см. рис. 20).



Рисунок 18. Процесс жарки мяса



Рисунок 19. Готовое мясо



Рисунок 20. Пережаренное мясо

Когда игрок правильно выполняет заказ, следуя рецепту и требованиям клиента, на экране появляется сообщение "Заказ выполнен" с зеленым фоном и галочкой (см. рис. 21). Это подтверждает, что клиент доволен, и заказ был выполнен точно и в срок. Такая реакция клиента положительно влияет на общий игровой прогресс.



Рисунок 21. Правильное выполнение заказа

Если игрок делает ошибки при выполнении заказа, например, использует неправильные ингредиенты или не соблюдает время приготовления, на экране появляется сообщение "Заказ провален" (см. рис. 22). Это означает, что клиент недоволен и заказ не соответствует его ожиданиям. Неверные заказы приводят к потере жизней, влияя на общий успех игрока в игре.



Рисунок 22. Неверное выполнение заказа

Каждый проваленный заказ приводит к потере одного сердца из индикатора жизней, расположенного в верхнем левом углу экрана. Этот индикатор показывает текущее состояние здоровья игрока. На картинке 23 видно, что у игрока есть три полных сердца, что означает полное здоровье. На картинке 24 после проваленного заказа одно сердце исчезает, остаются два, что указывает на потерю здоровья.

Если игрок теряет все сердца, игра заканчивается (см. рис. 25), и игроку приходится начинать заново. Таким образом, система здоровья стимулирует игрока внимательно и точно выполнять заказы, чтобы избежать потери жизней и добиться успеха в игре.



Рисунок 23. Начальное количество здоровья



Рисунок 24. Изменение индикатора здоровья

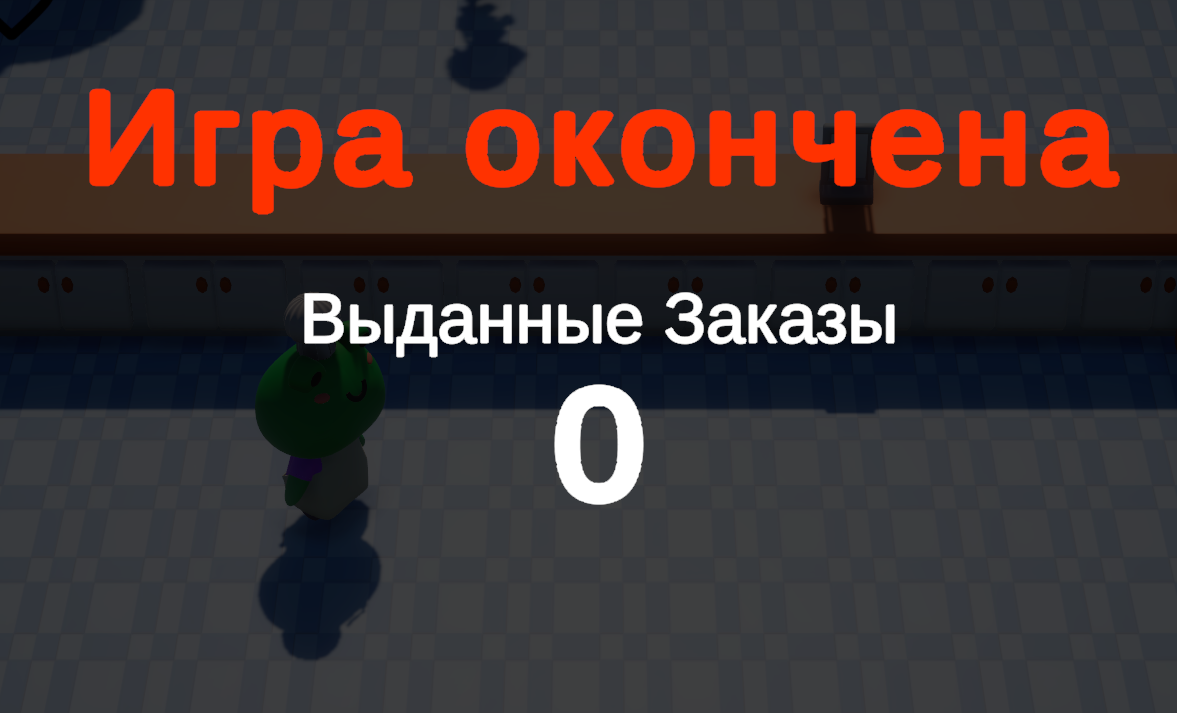


Рисунок 25. Окончание игры

## **2.5. Руководство по использованию программы**

Создание инсталлятора игры осуществляется с помощью среды разработки Unity, которая предлагает обширные возможности для экспорта готового продукта. В Unity имеется множество настроек для конфигурации проекта. На рисунке 26 можно увидеть, что разработчик может задать имя разработчика, название проекта, настроить курсор и логотип игры, а также включить опции для создания специальной версии для разработчиков.

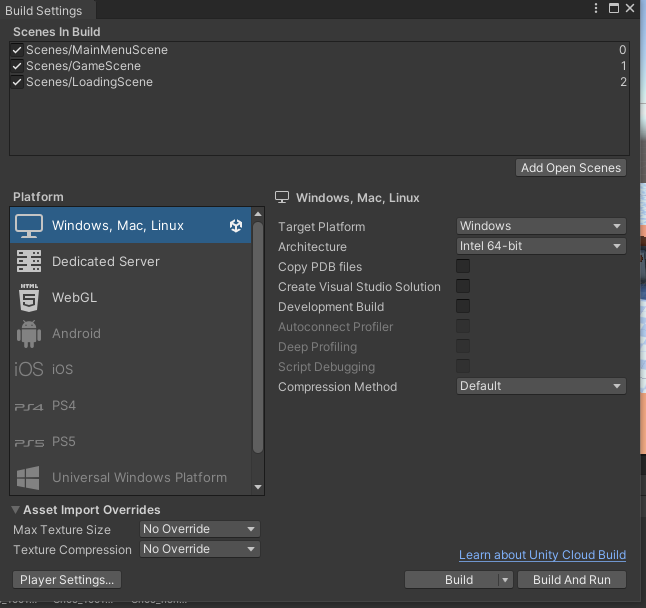


Рисунок 26. Окно «Build Settings» в Unity

Для разработки инсталлятора игры было применено приложение Inno Setup. В процессе установки пользователям предоставляется удобная возможность выбора необходимых параметров (см. рис. 27).

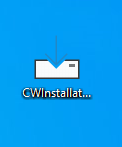


Рисунок 27.Инсталлятор игры

При установке игры пользователь должен прочитать лицензионное соглашение и принять условия пользования. (см. рис. 28)

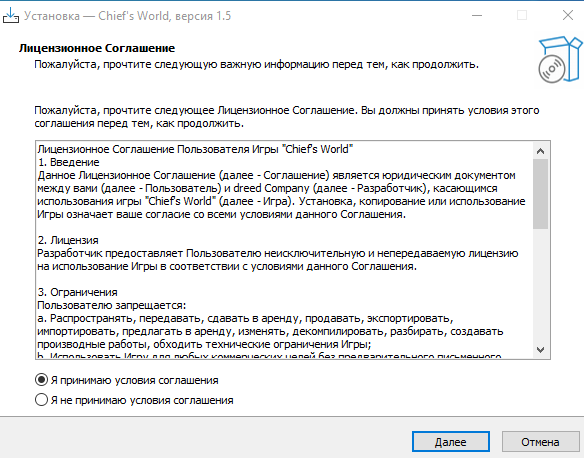


Рисунок 28. Пункт лицензионного соглашения при установке игры

Руководство системного программиста.

1. Общие сведения о программе.

1.1. Программа представляет собой компьютерную игру, в которой игроки управляют процессом приготовления блюд в ресторане. Цель игры - правильно выполнять заказы клиентов, готовя блюда из доступных ингредиентов. Игра разработана на языке программирования C# с использованием Unity.

1.2. Технические и программные средства:

- Язык программирования: C#

- Интегрированная среда разработки: Unity.

- Платформа: Windows.

- Минимальные требования к аппаратному обеспечению: совместимые с Unity.

2. Структура программы.

2.1. Программа состоит из следующих основных компонентов:

Игровая локация: отображает текущее состояние кухни и взаимодействует с пользователем.

Логика игры: управляет правилами игры, обработкой действий пользователя и проверкой условий выполнения заказов.

Графический интерфейс пользователя (GUI): обеспечивает визуализацию игрового процесса.

3. Настройка программы.

3.1. Установка программы:

- Скачайте установочный файл;

- Запустите установочный файл и следуйте инструкциям мастера установки;

- После завершения установки запустите программу;

3.2. Настройка программы:

Программа не требует дополнительной настройки, однако, вы можете настроить параметры звука и управления в меню настроек.

4. Проверка программы.

4.1. Способы проверки:

Проверьте работу программы с использованием контрольных примеров, предоставленных в разделе "Отладка и тестирование программы" в документации.

4.2. Контрольный пример:

Совершайте действия по приготовлению блюд, убедитесь в правильности выполнения заказов и начисления очков, а также в корректности работы системы оповещений.

5. Дополнительные возможности.

Дополнительные разделы функциональных возможностей программы отсутствуют в текущей версии.

6. Сообщения системному программисту.

6.1. Сообщения во время выполнения программы:

- "Заказ выполнен!"

- "Заказ провален."

Руководство программиста.

1. Назначение и условия применения программы.

Программа «Chef's World» предназначена для развлечения пользователей через игровой процесс, основанный на приготовлении блюд по заказам клиентов. Условия применения включают работу на платформах Windows, а также соответствие минимальным системным требованиям Unity.

2. Характеристики программы.

2.1. Интуитивно понятный интерфейс пользователя.

2.2. Анимации и звуковые эффекты для повышения интереса.

3. Обращение к программе.

Программа вызывается открытием исполняемого файла. Пользователь выбирает опции, используя мышь или клавиатуру.

4. Входные и выходные данные.

4.1. Входные данные: Управление осуществляется клавишами «WASD» или настроенными клавишами игрока. Действия игрока, взаимодействие с окружением, управление персонажем и использование игровых механик.

4.2. Выходные данные: Результаты игры сопровождается визуальным и звуковым контентом, создаваемым игрой в ответ на действия игрока, включая текущий счет и состояние заказов. После окончания игры выводится соответствующее сообщение.

5. Сообщения.

Возможны сообщения об ошибках, в этом случае необходимо проверить системные требования и обратиться к разработчику для решения проблемы.

Руководство оператора.

1. Назначение программы.

Программа предназначена для обеспечения пользователей возможностью игры, где они управляют процессом приготовления блюд в виртуальном ресторане. Игра разработана с использованием языка программирования C# и фреймворка Unity, предоставляя увлекательный и простой в использовании игровой процесс.

2. Условия выполнения программы.

2.1. Операционная система: Windows 10.

2.2. Процессор: 3.0 GHz и выше.

2.3. Оперативная память: 8 ГБ и выше.

2.4. Свободное место на жестком диске: 8 ГБ.

2.5. Видеокарта: 2 ГБ и выше.

3. Выполнение программы.

3.1. Загрузка и запуск программы:

Запустите программу, дважды кликнув по ярлыку на рабочем столе или используя меню "Пуск". Подождите, пока игра загрузится.

3.2. Начало игры:

Настройте клавиши управления, все необходимые параметры для вашего удобства и начните выполнение заказов.

3.3. Управление игрой:

Используйте настроенные клавиши (по умолчанию «WASD») для управления персонажем на кухне. Ваша цель - правильно выполнять заказы клиентов и достичь максимального результата.

3.4. Завершение игры:

Игра завершается, когда игрок теряет все жизни. После завершения игры вы увидите ваш счет и предложение начать новую игру.

4. Сообщения оператору

В программе выводятся сообщения о выполнении или провале заказа, звуковые и визуальные уведомления при использовании игровых механик и сведения о набранных очках.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломного проекта был достигнут значительный прогресс в реализации полноценного игрового процесса, объединяющего детализированную графику, продуманную анимацию, вовлекающий звуковой дизайн и многоаспектное взаимодействие объектов для создания богатого и запоминающегося впечатления.

Процесс разработки был направлен на освоение и применение сложных технических навыков, таких как трехмерное моделирование, анимация и программирование на C#. Unity Engine обеспечила гибкую и мощную платформу для создания продукта, который может найти широкое применение в индустрии развлечений. Отличительной чертой разработанной игры является её способность вовлекать игрока в процесс кулинарного творчества, позволяя влиять на события и развитие сюжета через выбор стилей приготовления и решения кулинарных задач.

Работа над проектом позволила не только достичь заранее определённых целей, но и приобрести важные навыки, которые окажутся полезными в дальнейшей профессиональной деятельности. Исследование конкурентов и анализ требований к играм подобного жанра обеспечили необходимые знания для формирования функционала и качества, соответствующего ожиданиям игрового сообщества.

Дальнейшее развитие проекта может включать расширение игровых уровней, сценария, улучшение звукового сопровождения и внедрение новаторских игровых механик, которые принесут дополнительную ценность и углубят геймплей. Также предстоит продолжить тестирование с целью сбора обратной связи от пользователей для дальнейшего улучшения игры. Это позволит не только улучшить качество текущей версии игры, но и заложить фундамент для возможных обновлений и дополнений.

В процессе разработки встречались трудности, связанные с нехваткой опыта в определённых аспектах создания игр и техническими ограничениями, но они были преодолены благодаря упорству и стремлению к обучению. Эти преодолённые препятствия не только способствовали созданию качественного продукта, но и предоставили ценный опыт, который будет способствовать развитию в области игровой разработки.

В итоге проект «Chef's World» представляет собой успешную реализацию кулинарной игры, демонстрирующую возможности современных технологий и творческого подхода. Опыт, полученный в процессе работы над проектом, будет важным этапом в профессиональном росте и дальнейшем развитии в области разработки игр.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ИНТЕРНЕТ-РЕСУРСОВ**

*Законодательные и нормативные акты:*

1. ГОСТ 7.1-2003 Библиографическая запись. Библиографическое описание. Общие требования и правила составления – 92 с.
2. ГОСТ 7.32-2017 Отчет о научно-исследовательской работе. Структура и правила оформления – 47 с.
3. ГОСТ 7.82-2001 Библиографическая запись. Библиографическое описание электронных ресурсов. Общие требования и правила составления – 39 с.
4. ГОСТ Р 7.0.100-2018 Библиографическая запись. Библиографическое описание. Общие требования и правила составления – 122 с.
5. ГОСТ Р 7.0.12-2011 Библиографическая запись. Сокращение слов и словосочетаний на русском языке. Общие требования и правила составления. – М.: ИПК Издательство стандартов, 2004. – 169 с.
6. ГОСТ Р 7.0.5-2008 Библиографическая ссылка. Общие требования и правила составления – 32 с.
7. Единая система программной документации. – М.: Стандартинформ, 2005. – 128 с.

*Учебная и научная литература:*

1. Андрей Кудряшов. Unity в действии: мультиплатформенная разработка на C#: пер. с англ. Москва: ДМК Пресс, 2021.

*Интернет документы:*

1. Аксенова Т.Г. Онлайн-курс по технологии разработки программного обеспечения [Электронный ресурс]. – Режим доступа:<https://classroom.google.com/u/2/w/NTc2MzExNTI0MTY0/t/all> (дата обращения: 09.10.2023).
2. Официальная документация по C# на официальном сайте Microsoft. Доступно на https://docs.microsoft.com/ru-ru/dotnet/csharp/. (дата обращения: 08.12.2023)
3. Ресурс Stack Overflow для поиска ответов на вопросы и помощи в разработке на C# и Unity. Доступно на https://stackoverflow.com/. (дата обращения: 13.12.2023)
4. Сайт игры Battle Chef Brigade для сравнительного анализа. [Электронный ресурс]. – Режим доступа: <https://www.battlechefbrigade.com/> (дата обращения: 12.10.2023)
5. Сайт игры Cooking Simulator для сравнительного анализа. [Электронный ресурс]. – Режим доступа: <https://store.steampowered.com/app/641320/Cooking_Simulator/> (дата обращения: 12.10.2023)
6. Сайт игры Overcooked для сравнительного анализа. [Электронный ресурс]. – Режим доступа: <https://www.overcooked.games/> (дата обращения: 12.10.2023)
7. Сайт по разработке 3D игры Game Kit [Электронный ресурс]. – <https://learn.unity.com/tutorial/3d-game-kit-walkthrough#> (дата обращения: 14.12.2023)
8. Справочник по изучению Unity3D [Электронный ресурс]. – Режим доступа: <https://github.com/deonisiu/Unity3d-Handbook/blob/master/general.md> (дата обращения: 19.02.2024)
9. Статья о логике написания игр [Электронный ресурс]. – Режим доступа: <https://github.com/Kelkhaun/Unity-Roadmap-RU> (дата обращения: 18.12.2023)
10. Статья по разработке 3D игры в Unity [Электронный ресурс]. – Режим доступа: <https://floop.top/ru/unity/> (дата обращения: 15.04.2024)
11. Статья по разработке игры с помощью Unity и C# [Электронный ресурс]. – <https://learn.microsoft.com/ru-ru/archive/msdn-magazine/2014/august/unity-developing-your-first-game-with-unity-and-csharp> (дата обращения: 08.03.2024)
12. Уроки как работать в Blender [Электронный ресурс]. – Режим доступа: <https://medium.com/quick-code/top-tutorials-to-learn-blender-for-3d-modeling-3f4bc3eae1d0> (дата обращения: 20.01.2024)

# **ПРИЛОЖЕНИЕ**

Приложение №1

Скрипты

Листинг 1. Скрипт персонажа

|  |
| --- |
| using System;  using UnityEngine;  public class PlayerBehavior : MonoBehaviour, IItemHandler  {  public static PlayerBehavior SharedInstance { get; private set; }  public event Action OnItemCollected;  public event Action<BaseStation> OnStationChanged;  private bool moving;  private Vector3 movementDirection;  private BaseStation currentStation;  private GameItem currentItem;  private void Awake()  {  if (SharedInstance == null)  {  SharedInstance = this;  }  else  {  Destroy(gameObject);  }  }  private void Start()  {  inputHandler.OnPrimaryAction += ExecutePrimaryAction;  inputHandler.OnSecondaryAction += ExecuteSecondaryAction;  }  private void Update()  {  HandleMovement();  DetectStation();  }  private void ExecutePrimaryAction(object sender, EventArgs e)  {  if (GameManager.Instance.IsGameActive())  {  currentStation?.ExecutePrimaryAction(this);  InteractWithClosestNPC();  }  }  private void ExecuteSecondaryAction(object sender, EventArgs e)  {  if (GameManager.Instance.IsGameActive())  {  currentStation?.ExecuteSecondaryAction(this);  }  }  private void InteractWithClosestNPC()  {  if (currentItem is PlateItem plate)  {  var npc = FindNearestNPC();  if (npc != null && npc.CanAcceptOrder)  {  npc.AcceptOrder(plate);  ClearCurrentItem();  }  }  }  private NPC FindNearestNPC()  {  NPC[] allNPCs = FindObjectsOfType<NPC>();  NPC nearestNPC = null;  float minDistance = float.MaxValue;  foreach (var npc in allNPCs)  {  float distance = Vector3.Distance(transform.position, npc.transform.position);  if (distance < minDistance)  {  minDistance = distance;  nearestNPC = npc;  }  }  return nearestNPC;  }  private void HandleMovement()  {  Vector2 inputVector = inputHandler.GetNormalizedMovementVector();  movementDirection = new Vector3(inputVector.x, 0, inputVector.y).normalized;  Vector3 newPosition = transform.position + movementDirection \* moveSpeed \* Time.deltaTime;  if (!IsPathObstructed(newPosition))  {  transform.position = newPosition;  moving = true;  transform.forward = Vector3.Lerp(transform.forward, movementDirection, Time.deltaTime \* 10);  }  else  {  moving = false;  }  }  private bool IsPathObstructed(Vector3 targetPosition)  {  float characterRadius = 0.5f;  return Physics.CheckCapsule(transform.position, targetPosition, characterRadius, stationDetectionMask);  }  private void DetectStation()  {  if (Physics.Raycast(transform.position, movementDirection, out RaycastHit hit, 2f, stationDetectionMask))  {  if (hit.transform.TryGetComponent(out BaseStation station))  {  ChangeCurrentStation(station);  }  }  else  {  ChangeCurrentStation(null);  }  }  private void ChangeCurrentStation(BaseStation newStation)  {  if (currentStation != newStation)  {  currentStation = newStation;  OnStationChanged?.Invoke(newStation);  }  }  public void AssignCurrentItem(GameItem item)  {  currentItem = item;  OnItemCollected?.Invoke();  }  public GameItem RetrieveCurrentItem()  {  return currentItem;  }  public void ClearCurrentItem()  {  if (currentItem != null)  {  Destroy(currentItem.gameObject);  currentItem = null;  }  }  public Transform GetItemHoldingPosition()  {  return itemHoldingPosition;  }  public bool HasCurrentItem()  {  return currentItem != null;  }  } |

Листинг 2. Скрипт кухонной плиты

|  |
| --- |
| using System;  using UnityEngine;  public class KitchenProcessor : MonoBehaviour  {  public event Action<ProgressEventArgs> OnProgress;  public event Action<StateChangeEventArgs> OnStateChange;  [Serializable]  public class StateChangeEventArgs : EventArgs  {  public ProcessorState NewState;  }  [Serializable]  public class ProgressEventArgs : EventArgs  {  public float Progress;  }  public enum ProcessorState  {  Inactive,  Active,  Completed,  Overprocessed  }  [SerializeField] private DishData[] dishes;  [SerializeField] private DishData[] overprocessDishes;  private ProcessorState currentState;  private float currentTime;  private DishData currentDish;  private float overprocessTime;  private DishData currentOverprocessDish;  private void Awake()  {  SetState(ProcessorState.Inactive);  }  private void Update()  {  if (IsActive())  {  ProcessItem();  }  }  private void SetState(ProcessorState newState)  {  currentState = newState;  OnStateChange?.Invoke(new StateChangeEventArgs { NewState = newState });  }  private bool IsActive()  {  return currentState == ProcessorState.Active || currentState == ProcessorState.Completed;  }  private void ProcessItem()  {  switch (currentState)  {  case ProcessorState.Active:  currentTime += Time.deltaTime;  OnProgress?.Invoke(new ProgressEventArgs { Progress = currentTime / currentDish.ProcessTime });  if (currentTime >= currentDish.ProcessTime)  {  CompleteProcessing();  }  break;  case ProcessorState.Completed:  overprocessTime += Time.deltaTime;  OnProgress?.Invoke(new ProgressEventArgs { Progress = overprocessTime / currentOverprocessDish.ProcessTime });  if (overprocessTime >= currentOverprocessDish.ProcessTime)  {  CompleteOverprocessing();  }  break;  }  }  private void CompleteProcessing()  {  ClearCurrentItem();  SpawnNewItem(currentDish.OutputItem);  SetState(ProcessorState.Completed);  overprocessTime = 0f;  currentOverprocessDish = GetDish(overprocessDishes, GetCurrentItem());  }  private void CompleteOverprocessing()  {  ClearCurrentItem();  SpawnNewItem(currentOverprocessDish.OutputItem);  SetState(ProcessorState.Overprocessed);  OnProgress?.Invoke(new ProgressEventArgs { Progress = 0f });  }  public void Interact(PlayerCharacter player)  {  if (HasNoItem())  {  if (player.HasItem())  {  StartProcessing(player.GetItem());  }  }  else  {  if (player.HasItem())  {  ProcessPlayerInteraction(player);  }  else  {  TransferItemToPlayer(player);  }  }  }  private void StartProcessing(GameItem item)  {  if (IsDishValid(item))  {  PlaceItem(item);  currentDish = GetDish(dishes, item);  SetState(ProcessorState.Active);  currentTime = 0f;  OnProgress?.Invoke(new ProgressEventArgs { Progress = currentTime / currentDish.ProcessTime });  }  }  private void ProcessPlayerInteraction(PlayerCharacter player)  {  if (player.GetItem().TryGetPlate(out Plate plate))  {  if (plate.TryAddIngredient(GetCurrentItem()))  {  ClearCurrentItem();  SetState(ProcessorState.Inactive);  }  }  }  private void TransferItemToPlayer(PlayerCharacter player)  {  GetCurrentItem().SetParent(player);  SetState(ProcessorState.Inactive);  }  private bool IsDishValid(GameItem item)  {  return GetDish(dishes, item) != null;  }  private DishData GetDish(DishData[] dishArray, GameItem item)  {  foreach (var dish in dishArray)  {  if (dish.InputItem == item)  {  return dish;  }  }  return null;  }  private void ClearCurrentItem()  {  GetCurrentItem()?.DestroySelf();  }  private void SpawnNewItem(GameItem newItem)  {  GameItem.Spawn(newItem, this);  }  private GameItem GetCurrentItem()  {  return GetComponentInChildren<GameItem>();  }  private bool HasNoItem()  {  return GetCurrentItem() == null;  }  private void PlaceItem(GameItem item)  {  item.SetParent(this);  }  }  public class PlayerCharacter : MonoBehaviour  {  public bool HasItem() { return GetItem() != null; }  public GameItem GetItem() { return GetComponentInChildren<GameItem>(); }  }  public interface IProgressTracker  {  event Action<KitchenProcessor.ProgressEventArgs> OnProgress;  event Action<KitchenProcessor.StateChangeEventArgs> OnStateChange;  }  [Serializable]  public class DishData  {  public GameItem InputItem;  public GameItem OutputItem;  public float ProcessTime;  }  public class GameItem : MonoBehaviour  {  public void SetParent(MonoBehaviour parent) { /\*...\*/ }  public void DestroySelf() { /\*...\*/ }  public static void Spawn(GameItem item, MonoBehaviour parent) { /\*...\*/ }  public bool TryGetPlate(out Plate plate) { plate = GetComponent<Plate>(); return plate != null; }  }  public class Plate : GameItem  {  public bool TryAddIngredient(GameItem item) { return true; }  } |

Листинг 3. Скрипт разделочной доски

|  |
| --- |
| using System;  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class SlicingStation: BaseStation, IHasProgress {  public static event EventHandler OnAnySlice;  new public static void ResetStaticData() {  OnAnySlice = null;  }  public event EventHandler<IHasProgress.OnProgressChangedEventArgs> OnProgressUpdated;  public event EventHandler OnSlice;  [SerializeField] private SlicingDishSO[] slicingDishArray;  private int slicingProgress;  public override void Interact(Player user) {  if (!HasKitchenItem()) {  if (user.HasKitchenItem()) {  if (IsValidInputForDish(user.GetKitchenItem().GetKitchenItemSO())) {  user.GetKitchenItem().SetKitchenItemParent(this);  slicingProgress = 0;  SlicingDishSO slicingDish = GetSlicingDishForInput(GetKitchenItem().GetKitchenItemSO());  OnProgressUpdated?.Invoke(this, new IHasProgress.OnProgressChangedEventArgs {  progressNormalized = (float)slicingProgress / slicingDish.slicingProgressMax  });  }  }  } else {  if (user.HasKitchenItem()) {  if (user.GetKitchenItem().TryGetPlate(out PlateKitchenItem plate)) {  if (plate.TryAddIngredient(GetKitchenItem().GetKitchenItemSO())) {  GetKitchenItem().DestroySelf();  }  }  } else {  GetKitchenItem().SetKitchenItemParent(user);  }  }  }  public override void InteractAlternate(Player user) {  if (HasKitchenItem() && IsValidInputForDish(GetKitchenItem().GetKitchenItemSO())) {  slicingProgress++;  OnSlice?.Invoke(this, EventArgs.Empty);  OnAnySlice?.Invoke(this, EventArgs.Empty);  SlicingDishSO slicingDish = GetSlicingDishForInput(GetKitchenItem().GetKitchenItemSO());  OnProgressUpdated?.Invoke(this, new IHasProgress.OnProgressChangedEventArgs {  progressNormalized = (float)slicingProgress / slicingDish.slicingProgressMax  });  if (slicingProgress >= slicingDish.slicingProgressMax) {  KitchenItemSO outputKitchenItemSO = GetOutputForInput(GetKitchenItem().GetKitchenItemSO());  GetKitchenItem().DestroySelf();  KitchenItem.SpawnKitchenItem(outputKitchenItemSO, this);  }  }  }  private bool IsValidInputForDish(KitchenItemSO inputKitchenItemSO) {  SlicingDishSO slicingDish = GetSlicingDishForInput(inputKitchenItemSO);  return slicingDish != null;  }  private KitchenItemSO GetOutputForInput(KitchenItemSO inputKitchenItemSO) {  SlicingDishSO slicingDish = GetSlicingDishForInput(inputKitchenItemSO);  if (slicingDish != null) {  return slicingDish.output;  } else {  return null;  }  }  private SlicingDishSO GetSlicingDishForInput(KitchenItemSO inputKitchenItemSO) {  foreach (SlicingDishSO slicingDish in slicingDishArray) {  if (slicingDish.input == inputKitchenItemSO) {  return slicingDish;  }  }  return null;  }  } |

Листинг 4. Скрипт системы жизней

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  public class LivesUIManager : MonoBehaviour  {  [SerializeField] private List<Image> lifeImages;  [SerializeField] private Sprite filledHeartSprite;  [SerializeField] private Sprite emptyHeartSprite;  private void Start()  {  if (GameManager.Instance != null)  {  GameManager.Instance.OnStateChanged += GameManager\_OnStateChanged;  GameManager.Instance.OnLifeChanged += GameManager\_OnLifeChanged;  UpdateLivesUI();  }  }  private void GameManager\_OnStateChanged(object sender, System.EventArgs e)  {  UpdateLivesUI();  }  private void GameManager\_OnLifeChanged(object sender, System.EventArgs e)  {  UpdateLivesUI();  }  private void UpdateLivesUI()  {  if (GameManager.Instance != null)  {  int lives = GameManager.Instance.GetLives();    for (int i = 0; i < lifeImages.Count; i++)  {  if (i < lives)  {  lifeImages[i].sprite = filledHeartSprite;  }  else  {  lifeImages[i].sprite = emptyHeartSprite;  }  }  }  }  } |

Листинг 5. Скрипт клиента

|  |
| --- |
| using System;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.AI;  public class NpcEventArgs : EventArgs  {  public int NpcId { get; set; }  public NpcEventArgs(int id)  {  NpcId = id;  }  }  public class NpcScript : MonoBehaviour  {  public NavMeshAgent agent;  private Animator animator;  public Vector3 spawnPosition;  public Vector3 targetPosition;  private bool atTarget = false;  private bool isOrderPlaced = false;  public event Action OnNpcInactive;  public DishSO currentOrder;  public bool isPlayerInRange { get; private set; }  public bool isReadyToReceiveOrder = true;  public int Id;  private static int nextId = 1;  public int npcToTrackId;  private float idleTime = 0f;  private const float maxIdleTime = 20f;  void Start()  {  Id = GetNextNpcId();  agent = GetComponent<NavMeshAgent>();  animator = GetComponent<Animator>();  DeliveryManager.Instance.OnDishCompleted += OrderCompleted;  SpawnNPC();  }  public static int GetNextNpcId()  {  return nextId++;  }  void Update()  {  bool isWalking = agent.velocity.magnitude > 0.1f;  animator.SetBool("IsWalking", isWalking);  animator.SetBool("IsIdle", !isWalking);  if (!agent.pathPending && agent.remainingDistance <= agent.stoppingDistance && !atTarget)  {  atTarget = true;  animator.SetBool("IsIdle", true);  if (!isOrderPlaced)  {  MakeOrder();  }  }  if (atTarget && isOrderPlaced)  {  idleTime += Time.deltaTime;  if (idleTime >= maxIdleTime)  {  NotifyOrderFailed();  ReturnToSpawn();  }  }  else  {  idleTime = 0f;  }  }  void SpawnNPC()  {  if (PositionTracker.Instance == null)  {  return;  }  var targetPositions = new Vector3[] { new Vector3(-1f, 0f, -7.05f), new Vector3(3.04f, 0f, -7.05f), new Vector3(5.52f, 0f, -7.05f), new Vector3(-4f, 0f, -7.05f) };  foreach (var pos in targetPositions)  {  PositionTracker.Instance.RegisterPosition(pos);  }  Vector3? freePosition = null;  foreach (var pos in targetPositions)  {  if (PositionTracker.Instance.TryOccupyPosition(pos))  {  freePosition = pos;  break;  }  }  if (freePosition == null)  {  return;  }  targetPosition = (Vector3)freePosition;  spawnPosition = new Vector3(-5f, 0f, -20f);  NavMeshHit hit;  if (NavMesh.SamplePosition(spawnPosition, out hit, 1.0f, NavMesh.AllAreas))  {  agent.Warp(hit.position);  GoToTarget();  }  else  {    }  MakeOrder();  }  public void SetOrder(DishSO order)  {  currentOrder = order;  var uiElement = GetComponentInChildren<DeliveryManagerSingleUI>(true);  if (uiElement != null)  {  uiElement.SetDishSO(order);  uiElement.gameObject.SetActive(true);  }  else  {    }  }  void GoToTarget()  {  if (agent.isOnNavMesh)  {  agent.SetDestination(targetPosition);  }  else  {    }  }  void OnTriggerEnter(Collider other)  {  if (other.CompareTag("player"))  {  isPlayerInRange = true;  }  }  void OnTriggerExit(Collider other)  {  if (other.CompareTag("player"))  {  isPlayerInRange = false;  }  }  public bool IsPlayerInRange()  {  return isPlayerInRange;  }  void MakeOrder()  {  isOrderPlaced = true;  DeliveryManager.Instance.SpawnDish(this);  }  public void ReceiveOrder(PlateKitchenObject plate)  {  if (currentOrder != null && DeliveryManager.Instance.IsDishMatch(plate, currentOrder))  {  DeliveryManager.Instance.DeliverDish(plate, this);  ReturnToSpawn();  ClearOrderUI();  }  else  {  ReturnToSpawn();  DeliveryManager.Instance.DeliverDish(plate, this);  ClearOrderUI();  }  }  public void ClearOrderUI()  {  var uiElement = GetComponentInChildren<DeliveryManagerSingleUI>(true);  if (uiElement != null)  {  uiElement.gameObject.SetActive(false);  }  else  {    }  }  void OrderCompleted(object sender, EventArgs e)  {  if (sender == this)  {  ReturnToSpawn();  }  }  public void ReturnToSpawn()  {  PositionTracker.Instance.ReleasePosition(targetPosition);  agent.SetDestination(spawnPosition);  atTarget = false;  isOrderPlaced = false;  animator.SetBool("IsIdle", true);  PositionTracker.Instance.ReleasePosition(targetPosition);  OnNpcInactive?.Invoke();  Invoke("DestroyNpc", 3.0f);  }  void DestroyNpc()  {  Destroy(gameObject);  }  void OnDestroy()  {  DeliveryManager.Instance.OnDishCompleted -= OrderCompleted;  OnNpcInactive?.Invoke();  }  private void NotifyOrderFailed()  {  ClearOrderUI();  DeliveryManager.Instance.NotifyDishFailed(this);  }  } |