

Реферат

ИГРОВОЕ 2D ПРИЛОЖЕНИЕ «*FARMER'S VALLEY*» В ЖАНРЕ АРКАДНОГО ФЕРМЕРСКОГО СИМУЛЯТОРА В *TOP-DOWN* ПРОЕКЦИИ НА ПЛАТФОРМЕ *UNITY*: дипломная работа / И.Д. Дубовцов – Гомель: ГГТУ им. П. О. Сухого, 2024 – Дипломная работа: 234 страниц, 26 рисунков, 26 таблиц, 10 источников, семь приложений.

Ключевые слова: игра, приложение, компьютерная, ферма, игровой процесс, предмет, инвентарь, чат, *NPC*, графика, пользователь, разработка, токен, *API*, игровой персонаж, игровой движок, *Unity*, функционал, архитектура, структура, класс, структура данных, параметр, взаимодействие, графический интерфейс, интерфейс, искусственный интеллект.

Характеристика проделанной работы: при выполнении работы рассмотрены и проанализированы схожие игровые приложения. Проведен обзор подобных приложений, после чего сформирован список требований, которым должен соответствовать разрабатываемый программный продукт. При разработке использованы объектно-ориентированный и структурный подходы программирования. В результате разработки спроектировано и реализовано компьютерное игровое приложение в жанре аркадного фермерского симулятора.

Дипломная работа выполнена самостоятельно, приведенный в дипломной работе материал объективно отражает состояние разрабатываемого объекта, пояснительная записка проверена в системе «Антиплагиат.ру» (ссылка на систему: <https://www.antiplagiat.ru/>). Процент оригинальности составляет 86. Все заимствованные из литературных и других источников теоретические и методологические положения и концепции сопровождаются ссылками на источники, которые указаны в «Списке использованных источников».

Резюме

Темой работы является «игровое 2D приложение в жанре аркадного фермерского симулятора в *top-down* проекции на платформе *Unity*».

Объектом исследования является игровое приложение для компьютеров.

Целью работы является разработка игрового приложения для компьютеров в жанре аркадного фермерского симулятора.

Основным результатом работы является, компьютерное игровое приложение.

Резюмэ

Тэмай працы з'яўляецца гульнявое 2D дадатак ў жанры аркаднага фермерскага сімулятара ў *top-down* праекцыі на платформе *Unity*.

Аб'ектам даследавання з'яўляецца гульнявое прыкладанне для кампутараў.

Мэтай працы з'яўляецца распрацоўка гульнявога дадатку для кампутараў у жанры аркаднага фермерскага сімулятара.

Асноўным вынікам працы з'яўляецца, кампутарнае гульнявое дадатак.

Summary

The subject of the work is a 2D game application in the genre of arcade farming simulator in top-down projection on the Unity platform.

The object of the study is a game application for computers.

The purpose of the work is to develop a game application for computers in the genre of arcade farming simulator.

The main result of the work is a computer game application.

СОДЕРЖАНИЕ

Перечень условных обозначений и сокращений	7
Введение	8
1 Существующие методы реализации игрового приложения в жанре аркадного симулятора жизни фермера	10
1.1 Описание предметной области	10
1.2 Жанр аркадного фермерского симулятора	11
1.3 Аналитический обзор представителей игр жанра аркадного фермерского симулятора	13
1.4 Игровой движок <i>Unity</i>	16
1.5 Игровой движок <i>Unreal Engine</i>	19
1.6 Сравнение <i>Unity</i> и <i>Unreal Engine</i>	21
1.7 Требования к проектируемому программному обеспечению	23
2 Архитектура игрового приложения « <i>Farmer's Valley</i> »	26
2.1 Определение основных функций	27
2.2 Структурные элементы	29
2.3 Устройство внутренней архитектуры и схема разделения данных	30
2.4 Архитектура пользовательского интерфейса	35
3 Программная реализация игрового приложения « <i>Farmer's Valley</i> »	38
3.1 Взаимодействие игровых элементов	38
3.2 Принцип работы модуля управления игровым миром и управления фермой	39
3.3 Принцип работы программных модулей управления инвентарем	42
3.3 Принцип работы модуля экономической и социальной системы	49
3.4 Паттерны в приложении « <i>Farmer's Valley</i> »	51
4 Валидация и верификация результатов работы игрового приложения « <i>Farmer's Valley</i> »	53
4.1 Принципы работы пользовательского графического интерфейса и основные механики игрового приложения « <i>Farmer's Valley</i> »	53
4.2 Виды тестирования игр	61
4.3 Функциональное тестирование	61
4.3 Юзабилити-тестирование	65
5 Экономическое обоснование игрового приложения « <i>Farmer's Valley</i> »	67
5.1 Техничко-экономическое обоснование целесообразности разработки программного продукта и оценка его конкурентоспособности	67
5.2 Оценка трудоемкости работ по созданию программного обеспечения	69
5.3 Расчёт затрат на разработку программного продукта	73
5.4 Расчёт договорной цены и частных экономических эффектов от производства и использования программного продукта	77

6.1 Требования к производственному освещению	80
7 Энергосбережение и ресурсосбережение при эксплуатации программного обеспечения.....	83
7.1 Энергосбережение и ресурсосбережение при внедрении и эксплуатации программного обеспечения	83
Заключение	86
Список использованных источников	87
Приложение А Программный код приложения	88
Приложение Б Руководство системного программиста.....	222
Приложение В Руководство программиста	223
Приложение Г Руководство пользователя.....	224
Приложение Д Формулы расчета экономической эффективности	233
Приложение Ё Результат опытной эксплуатации.....	225
Приложение Ж Список опубликованных работ	233

Перечень условных обозначений и сокращений

В настоящей пояснительной записке применяются следующие термины, обозначения и сокращения.

Геймплей – компонент игры, отвечающий за взаимодействие игры и игрока.

Рендеринг – процесс получения изображения из *2D* или *3D* модели, компьютерный расчет и визуализация объектов и сцен со всей информацией о материалах, текстурах и освещении.

ОС – операционные системы.

ПК – персональный компьютер.

ПО – программное обеспечение.

2D – двухмерное пространство.

3D – трёхмерное пространство.

F2P – бесплатное игровое приложение.

NPC – неигровой персонаж.

API – программный интерфейс, то есть описание способов взаимодействия одной компьютерной программы с другими.

Токены – это блоки, на которые система разбивает текст.

ChatGPT – чат-бот с генеративным искусственным интеллектом.

QA (Quality Assurance) специалист – это профессионал, отвечающий за обеспечение качества программного обеспечения в *IT*-компаниях.

ММО – массовая многопользовательская онлайн-игра.

ИИ – искусственный интеллект.

ВВЕДЕНИЕ

Индустрия видеоигр непрерывно развивается, принося новые технологии и улучшая пользовательский опыт. Современные игры сочетают в себе высококачественную графику, сложные игровые механики и интерактивные элементы, что позволяет привлечь и удержать внимание широкой аудитории. Одним из популярных жанров являются аркадные фермерские симуляторы, которые предлагают игрокам возможность управлять виртуальными фермами, развивая свои навыки стратегического планирования и управления ресурсами.

Целью данной дипломной работы является разработка игрового 2D приложения в жанре аркадного фермерского симулятора под названием «*Farmer's Valley*», выполненного в *top-down* проекции на платформе *Unity*. Уникальность проекта заключается в использовании 2.5D графики, которая сочетает элементы двухмерной и трехмерной графики, а также в реализации социального взаимодействия с неигровыми персонажами (*NPC*) при помощи *ChatGPT API*. Эта интеграция позволит создать более реалистичные и динамичные диалоги, значительно улучшая пользовательский опыт.

Актуальность разработки обусловлена растущим интересом к играм, сочетающим простоту управления с богатым игровым контентом и возможностями для социального взаимодействия. Введение 2.5D графики и интеграция современных технологий искусственного интеллекта, таких как *ChatGPT*, открывают новые горизонты для создания инновационных игровых продуктов.

Новизна проекта заключается в уникальной комбинации визуальных и технологических решений. Применение 2.5D графики позволяет создать визуально привлекательную и разнообразную игровую среду, а использование *ChatGPT API* для общения с *NPC* обеспечивает динамичное и реалистичное взаимодействие, что является редкостью в аркадных фермерских симуляторах.

Проект «*Farmer's Valley*» не только соответствует современным тенденциям в индустрии видеоигр, но и вносит вклад в развитие технологий взаимодействия человека и компьютера. Использование искусственного интеллекта для управления диалогами с *NPC* может быть применено в других областях, таких как образовательные программы и интерактивные обучающие системы. Это подчеркивает значимость и практическую применимость данной работы как для игровой индустрии, так и для научно-исследовательских задач.

Основной целью дипломной работы является создание игрового приложения, которое будет сочетать в себе инновационные графические и интерактивные решения. Для достижения данной цели необходимо решить следующие задачи:

– провести анализ существующих методов и технологий создания аркадных фермерских симуляторов.

– определить основные функциональные компоненты приложения и разработать его архитектуру.

– реализовать основные механики игрового процесса, такие как управление фермой, экономическая система и смена дня и ночи, сохранение и загрузка игрового состояния.

– провести тестирование и верификацию приложения для обеспечения его стабильной работы и соответствия поставленным требованиям.

Таким образом, разработка игрового приложения «*Farmer's Valley*» представляет собой актуальную и перспективную задачу, соответствующую современным тенденциям в индустрии видеоигр и научных исследованиях.

Проект не только демонстрирует возможности использования новых технологий в разработке игр, но и открывает пути для дальнейших исследований и применения полученных знаний в других областях.

1 СУЩЕСТВУЮЩИЕ МЕТОДЫ РЕАЛИЗАЦИИ ИГРОВЫХ ПРИЛОЖЕНИЙ В ЖАНРЕ АРКАДНОГО СИМУЛЯТОРА ЖИЗНИ ФЕРМЕРА

1.1 Описание предметной области

По степени влияния на потребителей и вовлеченности их в интерактивное окружение, предлагаемое видеоиграми, этот сегмент уже давно выделяется среди других видов развлечений.

Разработку игр невозможно рассматривать обособленно от индустрии компьютерных игр в целом. Непосредственно создание игр – это только часть комплексной «экосистемы», обеспечивающей полный жизненный цикл производства, распространения и потребления таких сложных продуктов, как компьютерные игры.

В структуре современной игровой индустрии можно выделить следующие уровни:

- платформы;
- игровые движки;
- разработка видеоигр;
- издание и оперирование;
- популяризация и потребление.

Платформы – аппаратно-программные системы, позволяющие запускать интерактивные игровые приложения. Среди основных видов можно выделить следующие:

- персональные компьютеры на базе *Windows*, *Mac/OS X* или *Linux*;
- игровые консоли (специализированные устройства для игр, *Xbox One*, *PlayStation 4*, *Nintendo Wii U*);
- мобильные устройства (*iOS*, *Android*, *Windows*).

Игровые движки – программная прослойка между платформой и собственно кодом игры. Использование готового игрового движка позволяет существенно упростить разработку новых игр, удешевить их производство и существенно сократить время до запуска. Кроме того, современные игровые движки обеспечивают кроссплатформенность создаваемых продуктов. Из наиболее продвинутых движков можно выделить: *Unity 3D*, *Unreal Development Kit*, *CryENGINE 3 Free SDK*.

Разработка игр. Большое количество компаний и независимых команд занимаются созданием компьютерных игр. В разработке участвуют специалисты разных профессий: программисты, гейм-дизайнеры, художники, QA специалисты и др. К разработке крупных коммерческих игровых продуктов привлекаются большие профессиональные команды. Стоимость разработки

подобных проектов может составлять десятки миллионов долларов. Однако вполне успешные игровые проекты могут воплощаться и небольшими командами энтузиастов. Этому способствует присутствие на рынке большого количества открытых и распространенных платформ, качественных и практически бесплатных движков, площадок по привлечению «народных» инвестиций (краудфандинг) и доступных каналов распространения.

Издание и оперирование игр. Распространением игр или оперированием (в случае с ММО) занимаются, как правило, не сами разработчики, а издатели. При этом издатели (или операторы) локализуют игры, взаимодействуют с владельцами платформ, проводят маркетинговые компании, разворачивают инфраструктуру, обеспечивают техническую и информационную поддержку выпускаемым играм. Для средних и небольших игровых продуктов данный уровень практически не доступен. Такие продукты, как правило, сами разработчики выводят на рынок, напрямую взаимодействуя с платформами.

Популяризация. Специализированные средства массовой информации всегда являлись мощным каналом донесения информации до пользователей. Сейчас наиболее эффективным и широко представленным направлением СМИ являются информационные сайты, посвященные игровой тематике. Игровые журналы, долгое время выступавшие главным источником информации об играх, в настоящее время уступили свое место интернет-ресурсам. Специализированные выставки все еще остаются важным информационными площадками для игровой индустрии (*E3, GDC, Gamescom, ИзроМир, DevGamm*). Прямое общение прессы и игроков с разработчиками, обмен опытом между участниками рынка, новые контакты – вот то, что предлагают конференции и выставки в концентрированной форме. Еще один важный канал донесения полезной информации до игроков – это ТВ-передачи, идущие как в формате классического телевидения, так и на множестве каналов видео-контента.

Игроки – это основной источник прибыли для игровых продуктов. Но в современном мире наиболее активные игроки стали существенной движущей силой в популяризации игр и отчасти в расширении контента.

1.2 Жанр аркадного фермерского симулятора

Жанр игр про ферму включает в себя уникальные особенности и подходы к разработке. Создание игры в этом жанре требует не только технических навыков, но и понимания аспектов, характерных для фермерской тематики. Некоторые ключевые этапы разработки фермерской игры включают в себя концепцию, дизайн, программирование, тестирование и выпуск игры.

Разработчики фермерских игр должны учитывать разнообразные аспекты, включая управление фермой, выращивание культур, разведение скота,

улучшение оборудования и многое другое. Важно создать баланс между реализмом и увлекательным игровым процессом, чтобы игроки могли насладиться аутентичным опытом фермерской деятельности.

Фермерские игры обычно имеют вид сверху вниз и позволяют игрокам управлять всеми аспектами фермерского хозяйства. Игроки могут посеять и ухаживать за растениями, строить и улучшать фермерские постройки, обрабатывать поля, управлять скотом и многое другое. Цель игры может варьироваться от создания прибыльного фермерского хозяйства до выживания в форс-мажорных обстоятельствах.

Разработка игр в жанре фермерских симуляторов требует специфических навыков и инструментов. Этот жанр игр подразумевает создание виртуальных ферм, где игроки могут управлять аспектами фермерского хозяйства, от посевов до ухода за животными.

Создание фермерской игры начинается с концепции, где разработчики определяют основные механики и особенности игрового процесса. Затем идет проектирование элементов игры, включая графику, интерфейс, уровни сложности и систему управления. Программирование игры включает в себя создание логики игровых механик, взаимодействия объектов на ферме и реализацию анимаций.

Для разработки фермерских игр часто используются специализированные игровые движки, такие как *Unity* или *Godot*. Эти инструменты предоставляют разработчикам возможность создавать детальные 2D и 3D модели ферм, животных, растений, а также реализовывать сложные игровые системы, например, учет погодных условий или сезонности.

Фермерские симуляторы предлагают игрокам широкий спектр деятельности, включая посев и уход за культурами, разведение и уход за животными, строительство и расширение фермерских построек, покупку нового оборудования и техники. Важным аспектом игры является реализм фермерской деятельности, отраженный в визуальных и звуковых эффектах, а также в поведении растений и животных.

Игры в жанре фермерских симуляторов часто представлены в виде сверху вниз, что позволяет игрокам получить обзор на всю ферму и ее окрестности. Важным аспектом таких игр является создание уникальной атмосферы и стиля фермы, чтобы игроки могли погрузиться в виртуальный мир сельского хозяйства.

Фермерские игры предлагают игрокам разнообразие сценариев и возможностей развития. Игроки могут выбирать, какие культуры выращивать, какими методами ухаживать за животными, как распределять ресурсы и многое другое. Это позволяет каждой игре быть уникальной и дать разработчикам возможность создать идеальную ферму по своему вкусу.

1.3 Аналитический обзор представителей игр жанра аркадного фермерского симулятора

В этом подразделе представлен аналитический обзор наиболее значимых и популярных представителей жанра аркадного фермерского симулятора. Цель этого обзора – выявить ключевые черты и механики, которые способствовали успеху данных игр, а также понять, какие аспекты их дизайна и игрового процесса привлекают игроков. Рассмотрение успешных примеров позволит выделить лучшие практики и тенденции в жанре, что станет основой для дальнейшего развития собственной игры.

Серия игр *Farming Simulator* представляет собой значительное явление в жанре аркадных фермерских симуляторов. Начав своё существование в 2008 году, эта серия продолжает выпускаться и по сей день, демонстрируя высокий уровень популярности среди игроков. Основная концепция игр этой серии заключается в симуляции сельскохозяйственного труда, где игроку предоставляется возможность заниматься разнообразными видами деятельности, включая земледелие, животноводство, лесоводство и другие аспекты фермерской жизни.

Farming Simulator выделяется своей реалистичностью и широким спектром возможностей. Игроки могут выбирать разные направления в сельском хозяйстве, будь то выращивание сельскохозяйственных культур или разведение животных, а также приобретать и использовать разнообразное оборудование. Игра предлагает детализированную симуляцию, в которой каждое действие требует тщательного планирования и выполнения.

Обучение и интерфейс. Для новичков в игре предусмотрены обучающие задания, которые помогают освоить основные механики игры и понять, как эффективно управлять фермой. Ветераны серии, в свою очередь, могут использовать свой накопленный опыт для оптимизации процессов и максимизации урожайности.

Разнообразие деятельности. Игрокам предоставляется широкий спектр занятий, начиная от вспашки и посева полей и заканчивая уходом за животными и лесозаготовками. В игре представлено множество культур для выращивания, таких как пшеница, ячмень, кукуруза, картофель и другие. Каждая культура требует особого подхода к уходу и удобрению.

Экономические аспекты. Помимо земледелия, игроки также могут заниматься торговлей, покупая и продавая оборудование, а также продукцию своей фермы. Важным элементом игрового процесса является логистика, включающая перевозку урожая и управление запасами. В последних версиях игры были добавлены поезда, которые значительно облегчают транспортировку, но требуют дополнительных затрат и навыков управления.

Лесное хозяйство. В версии *Farming Simulator* была введена возможность заниматься лесоводством, что добавляет ещё один интересный элемент в игровой процесс. Этот аспект игры также требует специфических знаний и навыков для успешного ведения бизнеса.

Игры серии *Farming Simulator* учат игроков основам фермерства, предлагая погружение в процесс и понимание всех тонкостей сельскохозяйственного труда. Эти игры не только развлекают, но и предоставляют возможность получить полезные навыки, которые могут быть применимы и в реальной жизни.

Таким образом, серия *Farming Simulator* является не просто развлекательным продуктом, но и обучающим инструментом, который помогает игрокам лучше понять и оценить труд фермеров, предоставляя глубокий и увлекательный игровой опыт.

На рисунке 1.1 представлен кадр из игры *Farming Simulator*.



Рисунок 1.1 – Кадр игры из *Farming Simulator*

Игра *Stardew Valley*, созданная и разработанная всего одним человеком, Эриком Бароном, является значимым представителем жанра аркадных фермерских симуляторов. Вышедшая в 2016 году, она быстро завоевала популярность благодаря своему уникальному сочетанию различных игровых механик и богатого контента. Игра сочетает элементы фермерства, ролевых игр и симуляторов жизни, создавая уникальный и увлекательный игровой опыт.

Сюжет и персонажи. В основе сюжета лежит история персонажа, который получает в наследство ферму от своего дедушки. Игрок может настроить

внешность и пол персонажа, однако эти параметры не влияют на игровой процесс. Главный герой свободен в своих действиях: можно заниматься фермерством, исследовать пещеры, общаться с местными жителями или выполнять квесты.

Игровая свобода. *Stardew Valley* предоставляет игрокам значительную свободу выбора. Игроки могут решать, каким образом управлять фермой, какие культуры выращивать, как ухаживать за животными и какие задания выполнять. В игре присутствуют элементы дэйтинг-симулятора, позволяющие строить отношения с другими персонажами и даже вступать в брак.

На рисунке 1.2 представлен кадр из игры *Stardew Valley*.



Рисунок 1.2 – Кадр из игры *Stardew Valley*

Ролевая система и развитие навыков. В отличие от многих других фермерских симуляторов, *Stardew Valley* включает ролевую систему, в которой игроки могут прокачивать навыки своего персонажа. Каждый вид деятельности, будь то использование мотыг или лейки, приносит очки опыта, которые улучшают соответствующие навыки и открывают новые возможности.

Сезонность и планирование. Игра симулирует смену времён года, что влияет на доступность различных культур и видов деятельности. Летом можно выращивать определенные растения и собирать мед, а зимой заниматься консервацией продуктов. Важно учитывать прогноз погоды и планировать свои действия, чтобы максимально эффективно использовать ресурсы и время.

Стратегическое планирование. Успешное ведение фермы требует тщательного планирования. Игрокам нужно учитывать бюджет, выбирать и подготавливать участки для посадки, а также защищать урожай от вредителей с помощью пугал. В игре существует ограничение по времени и энергии персонажа, что добавляет элемент стратегического планирования в повседневные задачи.

Многопользовательский режим. Игра поддерживается и развивается своим разработчиком, и в последние годы была добавлена возможность многопользовательской игры.

Stardew Valley предлагает глубокий и захватывающий игровой опыт, сочетая элементы фермерства, ролевой игры и симулятора жизни. Игра позволяет игрокам чувствовать себя настоящими профессиональными садоводами, наслаждаясь процессом и достигнутыми результатами. Благодаря своему уникальному подходу к жанру и постоянной поддержке со стороны разработчика, *Stardew Valley* продолжает оставаться одной из самых популярных и любимых игр в жанре аркадных фермерских симуляторов.

1.4 Игровой движок *Unity*

Unity – это профессиональный игровой движок, позволяющий создавать видеоигры для различных платформ.

Любой игровой движок предоставляет множество функциональных возможностей, которые задействуются в различных играх. Реализованная на конкретном движке игра получает все функциональные возможности, к которым добавляются ее собственные игровые ресурсы и код игрового сценария.

Приложение *Unity* предлагает моделирование физических сред, карты нормалей, преграждение окружающего света в экранном пространстве (*Screen Space Ambient Occlusion, SSAO*), динамические тени. Список можно продолжать долго. Подобные наборы функциональных возможностей есть во многих игровых движках, но *Unity* обладает двумя основными преимуществами над другими передовыми инструментами разработки игр. Это крайне производительный визуальный рабочий процесс и сильная межплатформенная поддержка. Визуальный рабочий процесс – достаточно уникальная вещь, выделяющая *Unity* из большинства сред разработки игр. Альтернативные инструменты разработки зачастую представляют собой набор разрозненных фрагментов, требующих контроля, а в некоторых случаях библиотеки, для работы с которыми нужно настраивать собственную интегрированную среду разработки (*Integrated Development Environment, IDE*), цепочку сборки и прочее в этом роде. В *Unity* же рабочий процесс привязан к тщательно продуманному визуальному редактору. Именно в нем будут компоновать сцены будущей игры,

связывая игровые ресурсы и код в интерактивные объекты. *Unity* позволяет быстро и рационально создавать профессиональные игры, обеспечивая невиданную продуктивность разработчиков и предоставляя в их распоряжение исчерпывающий список самых современных технологий в области видеоигр.

Редактор особенно удобен для процессов с последовательным улучшением, например, циклов создания прототипов или тестирования. Даже после запуска игры остается возможность модифицировать в нем объекты и двигать элементы сцены. Настраивать можно и сам редактор. Для этого применяются сценарии, добавляющие к интерфейсу новые функциональные особенности и элементы меню.

На рисунке 1.3 представлен интерфейс игрового движка *Unity*.



Рисунок 1.3 – Интерфейс движка *Unity*

Дополнением к производительности, которую обеспечивает редактор, служит сильная межплатформенная поддержка набора инструментов *Unity*. В данном случае это словосочетание подразумевает не только места развертывания (игру можно развернуть на персональном компьютере, в интернете, на мобильном устройстве или на консоли), но и инструменты разработки (игры создаются на машинах, работающих под управлением как *Windows*, так и *Mac OS*) [1]. Эта независимость от платформы явилась результатом того, что изначально приложение *Unity* предназначалось исключительно для компьютеров *Mac*, а позднее было перенесено на машины с операционными системами

семейства *Windows*. Первая версия появилась в 2005 году, а к настоящему моменту вышли уже пять основных версий (с множеством небольших, но частых обновлений). Изначально разработка и развертка поддерживались только для машин *Mac*, но через несколько месяцев вышло обновление, позволяющее работать и на машинах с *Windows*. В следующих версиях добавлялись все новые платформы развертывания, например межплатформенный веб-плеер в 2006-м, *iPhone* в 2008-м, *Android* в 2010-м и даже такие игровые консоли, как *Xbox* и *PlayStation*. Позднее появилась возможность развертки в *WebGL* – новом фреймворке для трехмерной графики в веб-браузерах. Немногие игровые движки поддерживают такое количество целевых платформ развертывания, и ни в одном из них развертка на разных платформах не осуществляется настолько просто. Дополнением к этим основным достоинствам идет и третье, менее бросающееся в глаза преимущество в виде модульной системы компонентов, которая используется для конструирования игровых объектов. «Компоненты» в такой системе представляют собой комбинируемые пакеты функциональных элементов, поэтому объекты создаются как наборы компонентов, а не как жесткая иерархия классов. В результате получается альтернативный (и обычно более гибкий) подход к объектно-ориентированному программированию, в котором игровые объекты создаются путем объединения, а не наследования [2].

Оба подхода схематично показаны на рисунке 1.4.



Рисунок 1.4 – Сравнение наследования с компонентной системой

Каждое изменение поведения и новый тип врага требуют серьезной перестройки кода. Комбинируемые компоненты позволяют добавить компонент стрелка куда угодно: как к мобильным, так и к статичным врагам. В

компонентной системе объект существует в горизонтальной иерархии, поэтому объекты состоят из наборов компонентов, а не из иерархической структуры с наследованием, в которой разные объекты оказываются на разных ветках дерева. Разумеется, ничто не мешает написать код, реализующий вашу собственную компонентную систему, но в *Unity* уже существует вполне надежный вариант такой системы, органично встроенный в визуальный редактор. Эта система дает возможность не только управлять компонентами программным образом, но и соединять и разрывать связи между ними в редакторе. Разумеется, возможности не ограничиваются составлением объектов из готовых деталей; в своем коде вы можете воспользоваться наследованием и всеми наработанными на его базе шаблонами проектирования [3, с.283].

1.5 Игровой движок *Unreal Engine*

Unreal Engine (UE) – игровой движок, разрабатываемый и поддерживаемый компанией *Epic Games*.

Движком называют рабочую среду, позволяющую управлять всей системой элементов, из которых состоит игра.

Сегодня движок *Unreal Engine* активно применяется для разработки простых казуальных игр для смартфонов и планшетов, а также для создания полноценных высокобюджетных игр, рассчитанных на массовую аудиторию (их называют AAA-проектами). При этом не потребуется самостоятельно писать код, т. к. система визуального создания скриптов *Blueprints Visual Scripting* значительно упрощает задачу. Если же разработчик желает прописать игровую логику вручную, он может использовать язык программирования C++.

5 апреля 2022 года *Epic Games* порадовала пользователей, представив обновленный движок *Unreal Engine 5*, анонсированный два года назад. Среди главных фишек – максимум фотореализма, увеличенная производительность и новый интерфейс.

Unreal Engine остается популярным более 20 лет, т. к. обладает следующими достоинствами:

- широкий функционал;
- визуальное программирование;
- бесплатная лицензия;
- возможность создать кросс-платформер;
- большая база пользователей.

Epic Games решила дать разработчикам больше, чем простой инструмент – в UE пользователи могут начать работу даже без узкоспециализированных знаний в области языков программирования.

На рисунке 1.5 представлен интерфейс *Unreal Engine*.

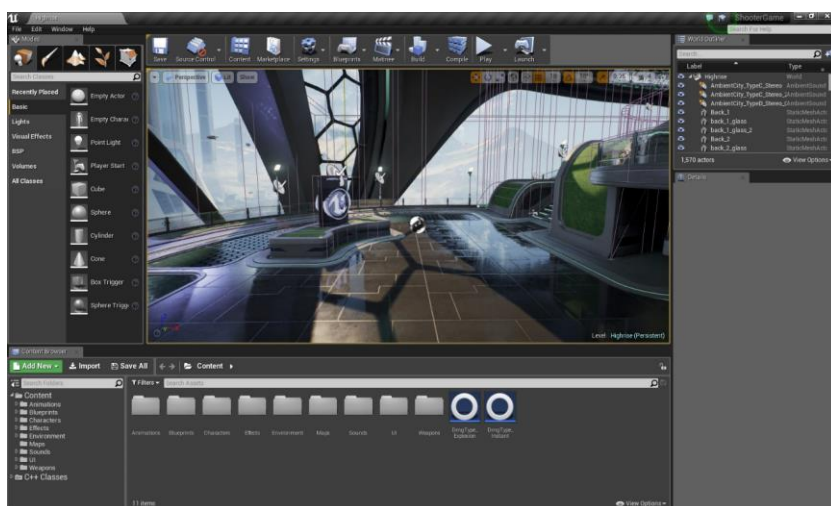


Рисунок 1.5 – интерфейс *Unreal Engine*

Blueprints значительно проще для понимания и использования, чем *C++*, при этом их функции и возможности в большинстве случаев схожи. Однако иногда все же придется прибегнуть к кодированию: для произведения сложных математических расчетов, изменения исходного кода самого движка *UE* и ряда базовых классов проекта.

В игровом мире существуют объекты с уникальными оттенками, фактурами и физическими свойствами. В движке *UE* внешний вид зависит от настроек материалов. Цвет, прозрачность, блеск – задать можно практически любые параметры. При работе над игрой в *UE* материалы можно наносить на любые объекты, вплоть до мелких частиц. Отметим, что речь идет не просто о настройке текстур: материалы открывают более широкие возможности. К

примеру, можно создавать необычные визуальные эффекты, причем UE позволяет делать это прямо в процессе игры.

Пользовательский интерфейс. Игроку важно не только видеть действия своего персонажа и карту, на которой он находится, но и иметь текстовую информацию, а также сведения о количестве очков, пунктах здоровья, инвентаре и т. д. С этой целью разработчики тщательно продумывают пользовательский интерфейс (*User Interface, UI*). В движке *Unreal* для создания UI применяется *Unreal Motion Graphics (UMG)*. Он позволяет выстраивать интуитивно понятный UI, выводить на экран необходимую пользователю информацию, а также менять положение кнопок и текстовых меток.

Анимация. Персонаж любой современной игры подвижен и гибок, умеет бегать и прыгать. Все это возможно благодаря анимированию. В UE начинающие разработчики могут импортировать уже готовые мэши со скелетами персонажей и настройки анимации. Неопытных пользователей, которые желают познакомиться с ПО поближе, приятно удивит *Animation Blueprint* – скрипт, который значительно упрощает работу по созданию паттернов движений персонажа без использования кодинга.

Звук. Для полного погружения в игру недостаточно просто собрать саундтрек из десятка файлов – музыку следует подобрать по тематикам сцен, настроить уровень ее громкости, прописать и расставить по нужным местам диалоги персонажей. В UE можно по-разному настраивать звуковые эффекты, зацикливать музыку и модулировать тон при каждом новом воспроизведении, а также работать с несколькими эффектами одновременно. За последнее отвечает ассет *Sound Cue*.

Система частиц. Данный компонент необходим для создания визуальных эффектов. Взрывы, брызги, искры, туман, снегопад или дождь – в UE все это можно создать, используя систему *Cascade*.

Искусственный интеллект. В компьютерной игре существуют не только главные, но и второстепенные персонажи. Искусственный интеллект (ИИ) отвечает за их решения (увидеть действие и среагировать). Настроить ИИ в UE можно, используя так называемые деревья поведения, *Behavior Trees*. В простые схемы закладываются алгоритмы действий и принятия решений. Здесь не только новичкам, но и профессионалам будет удобнее работать в *Blueprints Visual Scripting*, ведь все деревья визуально напоминают простые блок-схемы. Выстроить их гораздо быстрее и проще, чем писать длинный код.

1.6 Сравнение Unity и Unreal Engine

Первая область сравнения – редакторы для создания уровней, которые очень похожи. В них есть браузеры контента для ассетов, скриптов и других

файлов проекта. Игровые объекты можно перетаскивать в область сцены и, таким образом, добавлять в её иерархию.

Объекты в редакторе сцены изменяются с помощью инструментов перемещения, поворота и масштабирования – они похожи в обоих движках. Свойства *Unity*-объектов отображаются в *Inspector*, а *UE* – в части *Details*. *Jayanam* также сравнивает возможности *Unity Prefabs* с *Blueprints*.

В обоих движках есть статические меши (*static meshes*) – их можно двигать, поворачивать и масштабировать – и скелетные меши (*skeletal meshes*) – геометрические объекты, привязанные к костям скелета и используемые для анимирования персонажей. Их можно создавать в программах вроде *Blender* или *Maya*.

Анимации, включённые для скелетных мешей, также можно импортировать. В *Unity* они прикрепляются к импортированному объекту, как клипы анимации (*animation clips*), а в *UE* называются последовательностями анимации (*animation sequences*). В первом движении управляются с помощью контроллеров анимации (*animation controllers*), а во втором – по тому же принципу действуют анимационные *Blueprints*.

В обоих движках есть стейт-машины, определяющие переходы из одного состояния ассета в другое.

В *UE* система называется *Persona*, а в *Unity* – *Mecanim*. В них возможно применение скелетных мешей одного скелета к другим, но в *Unity* это в основном используется для анимирования гуманоидов. В *UE* анимации можно редактировать, в *Unity* – практически нет, особенно плохо дело обстоит с движениями гуманоидов. Движки не подходят для профессионального анимирования персонажей – лучше использовать программы вроде *Blender* или *Maya*, а результат импортировать в виде *FBX*-файлов. Прикреплённый к объектам материал добавляется в проект, но его свойства вроде шейдера или текстур придётся применять вручную [4].

Для этого в *Unity* нужно задать материалу шейдер и добавить в его слоты текстуры – карты шероховатостей, нормалей или диффузии. Собственные шейдеры придётся писать самостоятельно или с помощью сторонних инструментов вроде *Shader Forge* или *ASE*. А в *UE* встроен очень мощный редактор материалов, основанный, как и система *Blueprints*, на нодах.

Для программирования в *UE* используется язык *C++*, который не все любят из-за сложности и продолжительности компилирования. Однако *Jayanam* считает, что у движка понятный *API* и приемлемый период компиляции. В *UE* очень мощная и проработанная система визуального скриптования – *Blueprints*, с помощью которой можно достичь практически тех же результатов, что и с *C++*.

Unity поддерживает языки *C#* и *UnityScript*. *API* и его концепт очень похож на аналог из *UE*. При использовании управляемого языка вроде *C#*, программист

не обязан использовать указатели (*pointers*), компилирование происходит быстро. В *Unity* нет системы визуального «скриптования», и чтобы использовать что-то подобное, разработчик вынужден покупать сторонние дополнения вроде *Playmaker*.

Для 2D-разработки в *Unity* есть великолепные инструменты – *sprite creator*, *sprite editor* и *sprite packer*. *UE* также поддерживает спрайты в *Paper 2d*, но решения из *Unity* мощнее, кроме того, в последнем есть отдельный физический движок для 2d-объектов.

В *UE* встроен *постпроцессинг*. К сцене можно применять *bloom*-эффект, тонирование и антиалиасинг как глобально, так и к отдельным её частям (при помощи компонента *PostProcessVolume*).

В *Unity* есть стек постпроцессинга, который можно скачать из магазина ассетов движка. Система менее гибкая, чем в *UE* – эффекты применяются только стеком или скриптами к камере.

Sequencer в *UE* можно использовать для создания синематиков. Это мощный инструмент, работающий по принципу добавления объектов на временную шкалу. К настоящему моменту в *Unity 5.6* нет системы для синематиков, но *timeline*-редактор добавили в *Unity 2017*.

Для реализации приложения был выбран игровой движок *Unity*, т.к. он обладает наибольшим инструментарием и информационной базой, которые в один момент являются удобными и простыми [4, с. 441].

1.7 Требования к проектируемому программному обеспечению

Целью разработки является создание программного обеспечения (ПО) для аркадного фермерского симулятора, который предоставит пользователю возможность управлять виртуальной фермой, заниматься сельскохозяйственными работами, взаимодействовать с другими персонажами и развивать свои навыки в различных аспектах фермерской деятельности.

Программное обеспечение должно включать следующие функциональные возможности:

- создание нового игрового состояния: возможность выбрать название нового игрового состояния;
- загрузка выбранного игрового состояния: возможность выбора сохраненного игрового состояния;
- управление фермой: посадка и уход за различными культурами, управление ресурсами и инвентарем;
- социальное взаимодействие: общение с *NPC* (неигровыми персонажами);
- экономическая система: покупка и продажа продуктов, управление бюджетом фермы;

– атмосферное составляющее: смена дня и ночи;

Игровые механики включают:

1. посадка и сбор урожая:

- механика вспашки земли;
- посадка семян различных культур;
- полив растений;
- сбор урожая по мере созревания;

2. механика загрузки и сохранения игрового состояния;

3. общение с NPC;

4. взаимодействие с активным, вспомогательным инвентарем, а также возможность хранения предметов в сундуке или сундуках;

5. экономика:

- продажа продукции;
- покупка семян;

6. смена дня и ночи;

7. возможности настройки игровой среды:

- смена разрешения экрана;
- смена полноэкранного режима на оконный и наоборот;
- громкости музыки и звуковых эффектов;
- смена управления;

8. сохранение и загрузка настроек игровой среды.

Графические и визуальные элементы включают следующие пункты:

1. стиль графики: 2.5D графика (создание 2D-спрайтов на основе 3D моделей с маской освещения и картами нормалей для достижения глубины и детализации, яркие насыщенные цвета для создания уютной и привлекательной атмосферы;

2. персонажи: спрайты персонажей с анимацией движения;

3. окружение: спрайты растительности, зданий и объектов;

4. интерфейс пользователя: интуитивно понятный и легко читаемый интерфейс;

5. эффекты и анимация:

– анимации различных действий персонажа (сбор урожая, вспашка, полив);

- визуальные эффекты для улучшения восприятия;

6. звуковые эффекты и музыка:

- эффекты для действий;
- фоновая музыка, создающая атмосферу спокойствия и умиротворенности.

Входные данные:

– данные о игровом состоянии (игровая валюта);

– команды от пользователя (например, действия по посадке, сбору урожая).

Условия эксплуатации:

– программное обеспечение должно работать на платформе *Windows*;

– поддержка разрешений экрана от 1024 пикселей на 768 пикселей и выше;

– требования к минимальной производительности: 4 ГБ оперативной памяти, процессор с частотой 2 ГГц, видеокарта с поддержкой *OpenGL 3.0*;

– обеспечение возможности сохранения и загрузки игрового процесса.

Требования к надежности и быстродействию:

– программное обеспечение должно обеспечивать стабильную работу без сбоев и утечек памяти;

– быстродействие должно обеспечивать плавный игровой процесс с частотой не менее 30 кадров в секунду.

В качестве игрового движка был выбран *Unity*, так как он обеспечивает мощные инструменты для разработки 2D игр, поддержку множества платформ и имеет обширное сообщество разработчиков. Среда разработки *Visual Studio*, интегрированная в *Unity*, для удобства написания и отладки кода.

Инструментарий для разработки графики был выбран в пользу *Blender* для создания 3D моделей и *Adobe Photoshop* с *Aseprite* для обработки графических элементов.

Система контроля версий *Git* для отслеживания и сохранения изменений в процессе работы над проектом.

Таким образом, разработка программного обеспечения будет осуществляться с использованием проверенных и современных технологий, обеспечивающих надежность, быстродействие и удобство работы как для разработчиков, так и для конечных пользователей.

2 АРХИТЕКТУРА ИГРОВОГО ПРИЛОЖЕНИЯ «*FARMER'S VALLEY*»

Представление надежной и эффективной архитектуры приложения является критическим компонентом в разработке современных приложений. Понимание важности архитектуры и структуры приложений пришло к разработчикам игр не сразу. Это был пошаговый процесс, основанный на накопленном опыте, столкновениях с проблемами и поиске эффективных решений для их решения.

Главная цель разработки архитектуры приложения – определить структуру приложения, выделить его компоненты и определить взаимодействие между ними в приложении. Архитектура позволяет увидеть «общую картинку» приложения и оценить его целостность, а также обеспечить гибкость и расширяемость приложения в будущем.

Основные причины, по которым нужно разрабатывать архитектуру приложения, состоят в следующем:

- четкое определение структуры приложения и его компонентов;
- оценка сложности приложения и определение возможных рисков и проблем;
- обеспечение гибкости и расширяемости приложения в будущем;
- упрощение процесса разработки за счет более четкого понимания того, что нужно разрабатывать и как компоненты приложения взаимодействуют друг с другом;
- улучшение качества кода и снижение затрат на его сопровождение.

Разработка архитектуры может помочь определить, какие компоненты нужны для реализации игры, как они будут взаимодействовать друг с другом, какие алгоритмы нужны для обработки данных и как эти компоненты будут связаны с игровым движком. Это поможет упростить процесс разработки и обеспечить более гибкую и расширяемую архитектуру для будущих доработок и улучшений игры.

Чтобы правильно построить архитектуру приложения стоит использовать метод декомпозиции для правильной оценки задачи и последующем её разбиении на отдельные компоненты с целью оптимизации разработки приложения и дальнейшей поддержки.

Декомпозиция – это разделение большого и сложного на небольшие простые части. При постановке задач декомпозировать – значит разбить абстрактную большую задачу на маленькие задачи, которые можно легко оценить.

2.1 Определение основных функций

Для разработки архитектуры приложения важно выделить основные функции, которые будет реализовывать приложение. С помощью разделения задачи методом декомпозиции можно выделить подзадачи, которые будет представлять приложение. На рисунке 2.1 представлен результат выделения функциональных задач.



Рисунок 2.1 – Основные функциональные задачи

Как видно из рисунка 2.1 удалось определить основной функционал приложения в качестве задач и подзадач с некоторой конкретикой в определенных случаях.

Были выделены следующие задачи и подзадачи:

- а) обновление игровых объектов;
- б) отрисовка графики;
- в) визуальные и звуковые эффекты;
- г) социальное взаимодействие:
 - общение с *NPC*;
- д) экономическая система:

- покупка и продажа продукции;
- е) управление фермой:
 - посадка урожая;
 - сбор урожая;
 - уход за растениями;
- ж) управление ресурсами:
 - активный инвентарь;
 - дополнительный или вспомогательный инвентарь;
 - сундук или сундуки;
- з) смена дня и ночи;
- к) сохранение и загрузка игрового состояния;
- л) настройка игровой среды;
- м) сохранение и загрузка настроек игровой среды.

Описанные функции или подзадачи можно расценивать за компоненты, которые будут реализовывать ту или иную задачу.

За компоненты, описанные в пункте а и б, будет отвечать платформа *Unity*. Она будет обновлять игровые компоненты и рисовать графику.

Компоненты, описанные в пунктах б и в, будут представлять ресурсные компоненты, которые являются своеобразными исходными данными, которые будут импортироваться и находится в рабочей области платформы.

Схема инициализации компонентов, описанных в пунктах в и г показана на рисунке 2.2.

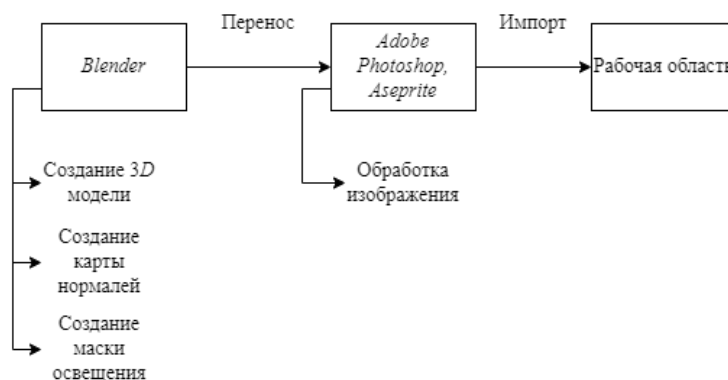


Рисунок 2.2 – Схема инициализации компонентов, описанных в пунктах б и в

Остальные функциональные компоненты следует выделять программными компонентами, которые будут реализовываться средствами написания кода.

Оставшиеся пункты можно разделить на компоненты, реализующие игровое составляющее игрового процесса или механики, другие компоненты будут отвечать за инициализацию и настройку игрового процесса.

2.2 Структурные элементы

При проектировании игрового приложения, возникает необходимость определения основных структурных элементов приложения.

Основные структурные элементы, которые будут использоваться в проекте включают:

- модуль управления конструированием игрового мира;
- модуль управления фермой;
- модуль социального взаимодействия;
- модуль экономической системы;
- модуль управления инвентарем;
- модуль визуальных и звуковых эффектов;
- модуль времени;
- модуль сохранения и загрузки;
- модуль управления игровыми сценами;
- модуль инициализации.

Пользователь будет взаимодействовать с этими модулями путем взаимодействия с *GUI* (*Graphical User Interface*).

Хорошей практикой является разделять работу модуля на работу с *GUI* и логическое составляющее.

Схема модуля показана на рисунке 2.3.



Рисунок 2.3 – Схема работы модуля

Пользователь будет взаимодействовать с вышеописанными модулями через графический пользовательский интерфейс (*GUI*). Разделение работы модуля на взаимодействие с *GUI* и логическую составляющую является хорошей практикой, позволяющей улучшить модульность и управляемость кода. *GUI*

будет служить мостом между пользователем и логикой приложения, предоставляя интуитивно понятные элементы управления и обратную связь.

Модуль управления конструированием игрового мира отвечает за создание и изменение игрового мира. За установку его границ и настройку положения визуальных игровых элементов.

Модуль управления фермой отвечает за любые аспекты управления фермой:

- посадка урожая;
- сбор урожая;
- рыхление земли;
- полив растений;
- сбор продукции по мере роста.

Модуль социального взаимодействия отвечает за инициализацию параметров разговора и оптимизацию расходуемых ресурсов. Реализует общение с *NPC* в формате чата с помощью *GUI*.

Модуль экономической системы отвечает за управление экономическими аспектами игры, такими как покупка необходимой продукции и продажа продукции.

Модуль управления инвентарем отвечает за управлением инвентарем игрока, включая *drag and drop* систему, перемещение объектов между другими объектами инвентаря и сундуками.

Модуль визуальных эффектов отвечает за создание определенного эффекта при взаимодействии с игровыми объектами.

Модуль звуковых эффектов отвечает за воспроизведение звуковых эффектов в определенное время при определенном действии.

Модуль времени отвечает за смену времени суток и отображение текущего времени суток в привычном формате.

Модуль сохранения и загрузки отвечает за сохранение и загрузку игрового состояния. Сохраняет и загружает положение игровых элементов, а также состояние других модулей.

Модуль управления игровыми сценами отвечает за смену игровых сцен.

Модуль инициализации отвечает за распределение необходимых зависимостей для других модулей и является точкой входа при запуске сцены.

2.3 Устройство внутренней архитектуры и схема разделения данных

В проекте внутренняя архитектура основана на модульном подходе, где каждый модуль отвечает за определённую часть функциональности игры. Это позволяет улучшить модульность, тестируемость и расширяемость приложения.

В целях оптимизации программного продукта следует разделить игровое приложение на несколько сцен при этом сократив количество игровых объектов на одной сцене.

Таким образом игровое приложение будет запускаться с начальной сцены (главное меню) и из начальной сцены запускать игровую сцену. На рисунке 2.4 представлена схема сцен.



Рисунок 2.4 – Схема сцен

Исходя из этого нахождение на сцене всех модулей совсем не обязательно.

На рисунке 2.5 представлена схема наличия модулей на начальной и игровой сцене.



Рисунок 2.5 – Схема наличия модулей на начальной и игровой сцене

После запуска сцены первым делом запускается модуль инициализации.

Каждая сцена инициализируется в своей точке входа, за которую отвечает модуль инициализации.

На рисунке 2.6 показана схема инициализации программных модулей после запуска сцены.



Рисунок 2.6 – Схема инициализации программных модулей после запуска сцены

Когда модуль инициализации закончит инициализировать остальные модули, то игровое состояние придет в рабочий режим, тогда работа модулей обновляется с помощью основного цикла игры. В случае, если модуль инициализации по каким-то причинам не сможет проинициализировать остальные модули, то запуск сцены будет отменен.

В этом разделе следует разобрать работу модуля инициализации, так как его функционал обусловлен использованием фреймворка *Zenject*. Этот фреймворк содержит много полезного функционала, однако стоит обратить внимание только на его реализацию архитектурного паттерна проектирования *Dependency Injection*.

Перед тем, как перейти к описанию архитектурного паттерна проектирования *Dependency Injection*, стоит начать с описания понятия инверсии управления (*IoC*).

Инверсия управления (*Inversion of Control, IoC*) – это ключевой принцип объектно-ориентированного программирования, который применяется для снижения связности (зацепления) в программном обеспечении. Данный подход также представляет собой архитектурное решение для интеграции, которое упрощает расширение функциональности системы за счет того, что поток управления программы контролируется фреймворком.

Паттерн внедрения зависимостей (*Dependency Injection Pattern*) – это дизайн-шаблон, который использует принцип *IoC*, чтобы обеспечить, что класс

не принимает участие в создании экземпляра зависимого класса и не управляет его жизненным циклом. Иными словами, процесс создания класса и инициализации его переменных, поступающих через конструктора, или методы, полностью возлагается на платформу, то есть на более высокий уровень.

В рамках реализации архитектурного паттерна *Dependency Injection* во фреймворке *Zenject* используется инструмент *Dependency Injection (DI) Container*.

Dependency Injection (DI) Container – это инструмент, который разрешает (*resolve*) зависимости для их внедрения. Проще говоря, это некий «черный ящик», в котором можно зарегистрировать классы (интерфейсы и их реализации) для последующего разрешения (*resolve*) в нужных местах, таких как конструкторы. Стоит отметить, что внедрение зависимостей возможно не только через конструкторов, но и через методы и свойства. Хотя внедрение через конструктор является самым распространённым.

Для чего нужен *DI*-контейнер:

- создание экземпляров объектов, включая разрешение зависимостей, в том числе и иерархических;
- управление жизненным циклом объектов (*lifetime*);
- доступ из любого места в программе, практически в любом конструкторе.

Таким образом, *DI*-контейнер играет важную роль в создании и управлении зависимостями, облегчая разработчикам задачу по обеспечению гибкости и модульности кода, а также упрощая процесс тестирования и сопровождения приложений.

Для обеспечения эффективного взаимодействия модулей используется схема разделения данных. Модули обмениваются данными через четко определенные интерфейсы, что позволяет поддерживать их независимость и облегчает тестирование и отладку.

Данные в приложении разделяются на несколько типов:

- игровые данные;
- экономические данные;
- визуальные и звуковые данные;
- временные данные;
- данные для загрузки и сохранения.

Игровые данные хранятся в оперативной памяти и относятся к текущему состоянию игрового мира, объектов и персонажей.

Экономические данные относятся к экономике игры и включают данные о предметах инвентаря и транзакции. К примеру, количество и типы ресурсов, стоимость покупки и продажи.

Визуальные данные относятся к визуальному представлению игры, к примеру, настройка графики и активные визуальные эффекты.

Звуковые данные относятся к звуковым эффектам и фоновой музыке.

Данные сохранений относятся к данным, которые используются для загрузки и сохранения игрового состояния.

В качестве модели данных определяются основные сущности и связи между ними, такие как игрок, *NPC*, предметы. Каждая сущность представлена отдельным классом или структурой данных.

На рисунке 2.7 представлена схема взаимодействия сущностей.

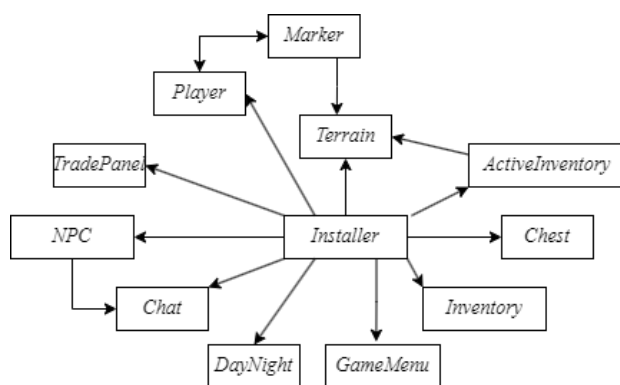


Рисунок 2.7 – Схема взаимодействия сущностей

Сущность *Player* представляет из себя информацию о его местоположении, текущей анимации.

Сущность *Marker*, который представляет из себя цель взаимодействия содержит информацию о своем местоположении по отношению к игроку.

Сущность «*Terrain*» содержит данные о том, где игрок установил семя, полил территорию и в какой стадии роста находится семя.

Сущность *InventoryItem* содержит данные об предмете инвентаря.

Сущность *Inventory* представляет ир себя хранилище данных об всех имеющихся предметах инвентаря.

Сущность *ActiveInventory* по своей сути является сущностью инвентаря, только использует данные о вводе пользователя, чтобы взаимодействовать с сущностью «территория».

Сущность «*Chest*» также по своей сути является сущностью инвентаря, только использует данные о вводе пользователя.

Сущность *NPC* представляет из себя неигрового персонажа, который обрабатывает данные о вводе пользователя для общения с ним в формате чата.

Сущность *Chat* представляет из себя данные о истории общения между игроком и *NPC*.

Сущность «*DayNight*» представляет из себя информацию о текущем времени суток.

Сущность *Installer* представляет из себя входные данные для сущностей игры.

2.4 Архитектура пользовательского интерфейса

Отдельное внимание стоит уделить разработке *GUI* на каждой из сцен. Создание эффективных пользовательских интерфейсов (*UI*) для видеоигр может быть сложным и трудоемким процессом. Дизайн пользовательского интерфейса – невероятно важный компонент разработки игр, так как он определяет первое впечатление игрока от игры, а также его общий опыт. Первым шагом в создании любого пользовательского интерфейса должно быть исследование и уточнение.

После начального этапа исследования и уточнения следующим шагом является создание каркаса. Каркас – это процесс наброска различных элементов пользовательского интерфейса, таких как меню, кнопки и окна. Это включает в себя выбор макета, функциональности и внешнего вида интерфейса.

Начальная сцена будет представлять из себя главное меню, где пользователь с помощью *GUI* будет запускать игровую сцену. На рисунке 2.8 представлен каркас пользовательского интерфейса на начальной сцене.

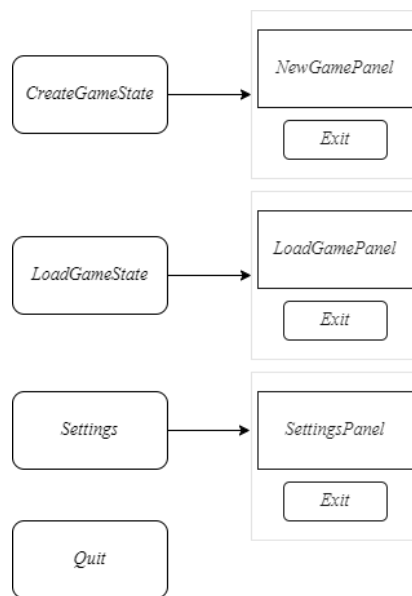


Рисунок 2.8 – Каркас пользовательского интерфейса на начальной сцене

Таким образом, начальная сцена должна содержать элементы пользовательского интерфейса для возможности запуска игровой сцены с учетом выбранного игрового состояния. Также на начальной сцене должны присутствовать элементы пользовательского интерфейса для настройки игровой среды и закрытия приложения. В таком случае начальная сцена должна содержать следующие элементы:

- кнопка, которая отображает панель создания нового игрового состояния;
- панель создания нового игрового состояния;
- кнопка, которая отображает панель загрузки игрового состояния;
- панель загрузки игрового состояния;
- кнопка, которая отображает панель настроек игровой среды.
- панель настроек игровой среды;
- кнопка, с помощью которой можно закрыть игровое приложения.

Важно понимать, что пользовательский интерфейс не должен занимать весь экран, поэтому некоторые элементы *UI* не должны постоянно отображаться на экране.

В игровой сцене также должны присутствовать следующие элементы пользовательского интерфейса:

- кнопка, которая отвечает за продолжение игрового сеанса;
- кнопка, которая отображает панель настроек игровой среды;
- кнопка, которая отвечает за переход на начальную сцену.

Каркас пользовательского интерфейса на игровой сцене показан на рисунке

2.9.

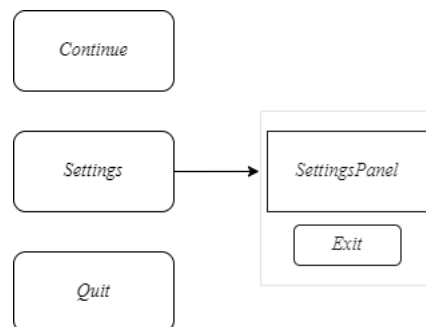


Рисунок 2.9 – Каркас пользовательского интерфейса на игровой сцене

После разработки каркасов управления приложением стоит перейти к разработке еще одного важного элемента пользовательского интерфейса, чтобы управлять игровым процессом на игровой сцене, который называется *HUD*.

HUD, если рассматривать его как цельный, самостоятельный интерфейс, может включать в себя множество различных элементов: полосы здоровья и старины, указатели направления, метки в мире, различные диететические элементы и мета-интерфейсы – все, что требуется для непосредственно игры в игру в зависимости от ее жанра, геймплея, набора задач и игровых ситуаций, которые необходимо решить с помощью *HUD(heads-up display)*.

В игровой сцене необходимо придумать каркас для следующих элементов интерфейса:

- текущее время суток;
- игровая валюта;
- активный инвентарь;
- вспомогательный инвентарь.

На рисунке 2.10 представлен каркас для игрового *HUD*.

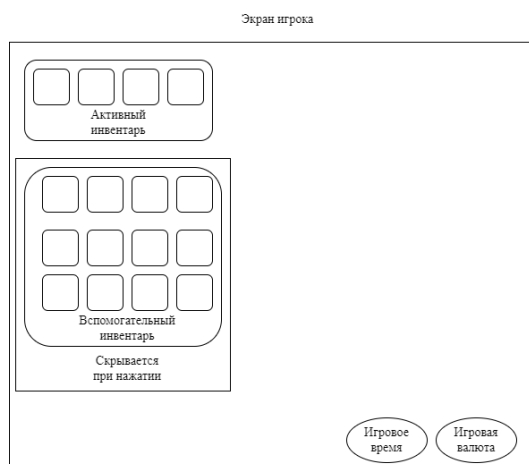


Рисунок 2.10 – Каркас для игрового *HUD*

Такая организация *UI* элементов позволит корректно отображать необходимую информацию, при этом почти не загрождая обзор игрока на игровой мир.

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ИГРОВОГО ПРИЛОЖЕНИЯ «FARMER'S VALLEY»

3.1 Взаимодействие игровых элементов

Приложение реализовано средствами платформы *Unity*. Таким образом, работа модулей игрового приложения организует определенную структуру для работы с компонентами платформы. На рисунке 3.1 представлена работа модулей.



Рисунок 3.1 – Работа программных модулей

Использование программных компонентов платформы позволяет организовать логику взаимодействия всех игровых объектов. Под логикой взаимодействия всех игровых объектов будет подразумеваться графическое составляющее игрового приложения и программное составляющее игрового приложения. Графическим составляющим являются спрайты, визуальные эффекты, настройки графики, анимации. Все эти элементы могут являться частями одного игрового объекта. Такие элементы инициализируются с помощью выбранного программного обеспечения, учитывая возможности платформы. Программное или логическое составляющее заключается в изменении свойств элементов игрового объекта в процессе обновления игрового цикла. Для изменения свойств игрового объекта в процессе обновления игрового цикла разрабатываются специальные классы и структуры данных, влияющие на организацию взаимодействия пользователя с игровым миром.

3.2 Принцип работы модуля управления игровым миром и управления фермой

Модуль управления игровым миром заключается в расстановке игровых объектов, формирующих игровую сцену и ее границы. Его работа тесно завязана с компонентом *Grid*. Компонент *Grid* является игровым объектом, представляющим сетку. Сетка характеризуется наличием ячеек. Ячейка представляет из себя ее координаты, такие как координаты в мировом пространстве и координаты ее центра.

Суть компонента *Grid* заключается в том, чтобы устанавливать игровые объекты на игровое поле исключительно по сетке. По ней выстраиваются такие игровые объекты, как объект семени, объект обработанной земли и земли, политой лейкой, объекты декора.

Модуль управления фермой характеризуется классом *PlacementService*, который содержит вложенные классы *CropData*, *GroundData*.

PlacementService – класс, который отвечает за установку предметов на сетке и следит за соблюдением необходимых условий для установки предмета на сетке. Реализует паттерн *Singleton*.

Все поля класса *PlacementService* показаны в таблице 3.1

Таблица 3.1 – Поля класса *PlacementService*

Название	Модификатор доступа	Тип данных
<i>GroundTilemap</i>	<i>public</i>	<i>Tilemap</i>
<i>CropTilemap</i>	<i>public</i>	<i>Tilemap</i>
<i>WaterTilemap</i>	<i>public</i>	<i>Tilemap</i>
<i>WateredTile</i>	<i>public</i>	<i>Tilemap</i>
<i>TilleableTile</i>	<i>public</i>	<i>Tilemap</i>
<i>TilledTile</i>	<i>public</i>	<i>Tilemap</i>
<i>TillingEffectPrefab</i>	<i>public</i>	<i>VisualEffect</i>
<i>HarvestEffectPool</i>	<i>private</i>	<i>Dictionary<Crop, List<VisualEffect></i>
<i>S_Instance</i>	<i>private</i>	<i>PlacementService</i>
<i>groundData</i>	<i>Private</i>	<i>Dictionary<Vector3Int, GroundData></i>
<i>cropData</i>	<i>private</i>	<i>Dictionary<Vector3Int, CropData></i>

Поля класса *PlacementService* в таблице 3.1 описывают следующую логику:

– *GroundTilemap* – определяет карту расположения для спрайтов земли;

- *CropTile* – определяет карту расположения для спрайтов посева;
- *WaterTilemap* – определяет карту расположения для спрайтов политой земли;
- *WateredTile* – определяет спрайт политой земли;
- *TilleableTile* – определяет спрайт земли, который можно рыхлить;
- *TilledTile* – определяет спрайт взрыхленной земли;
- *TilingEffectPrefab* – определяет эффект взаимодействия;
- *HarvestEffectPool* – содержит эффекты взаимодействия, которые инициализируются заранее;
- *groundData* – содержит данные о состоянии объектов земли;
- *cropData* – содержит данные о состоянии объектов посева.

Чтобы располагать игровые объекты на сетке, соблюдая некоторые условия, содержит методы, описанные в таблице 3.2.

Таблица 3.2 – Методы класса *PlacementService*

Название	Модификатор доступа	Тип возвращаемого значения	Параметры(тип данных)
<i>Awake</i>	<i>Private</i>	<i>Void</i>	–
<i>Start</i>	<i>Private</i>	<i>Void</i>	–
<i>Update</i>	<i>Private</i>	<i>Void</i>	–
<i>IsTillable</i>	<i>Public</i>	<i>Bool</i>	<i>Target(Vector3Int)</i>
<i>IsPlantable</i>	<i>Public</i>	<i>Bool</i>	<i>Target(Vector3Int)</i>
<i>IsTilled</i>	<i>Public</i>	<i>Bool</i>	<i>Target(Vector3Int)</i>
<i>TillAt</i>	<i>Public</i>	<i>void</i>	<i>Target(Vector3Int)</i>
<i>PlantAt</i>	<i>Public</i>	<i>void</i>	<i>Target(Vector3Int)</i> , <i>cropToPlant(Crop)</i>
<i>HarvestAt</i>	<i>Public</i>	<i>Crop</i>	<i>Target(Vector3Int)</i>
<i>WaterAt</i>	<i>Public</i>	<i>Void</i>	<i>Target(Vector3Int)</i>
<i>Save</i>	<i>Private</i>	<i>Void</i>	–
<i>Load</i>	<i>Public</i>	<i>Void</i>	–
<i>Instance</i>	<i>Public</i>	<i>PlacementService</i>	–

Метода класса *PlacementService* в таблице 3.2 описывают следующую логику:

- *Awake* – инициализирует экземпляр класса;
- *Start* – инициализирует стартовый массив эффектов взаимодействия;
- *Update* – обновляет параметры процесса роста посева и мокрой земли;
- *IsTillable* – определяет возможность установки рыхлой земли;
- *IsPlantable* – определяет возможность установки объекта посева;

- *IsTilled* – определяет, стоит ли на определенной ячейке;
- *TillAt* – устанавливает объект рыхлой земли на ячейку;
- *PlantAt* – устанавливает объект посева на ячейку;
- *HarvestAt* – собирает объект посева и проигрывает определенный визуальный эффект на ячейке;
- *WaterAt* – устанавливает объект политой земли на ячейку;
- *Save* – сохраняет данные об объекте посева и позиции рыхлой земли;
- *Load* – устанавливает объекты посева и позиции рыхлой земли исходя из сохраненных данных;
- *Instance* – возвращает экземпляр класса.

CropData – класс, который содержит параметры процесса роста объекта посева. Все поля класса *CropData* показаны в таблице 3.3.

Таблица 3.3 – Поля класса *CropData*

Название	Модификатор доступа	Тип данных
<i>GrowingCrop</i>	<i>Public</i>	<i>Crop</i>
<i>CurrentGrowthStage</i>	<i>Public</i>	<i>Int</i>
<i>GrowthRatio</i>	<i>Public</i>	<i>float</i>
<i>GrowthTimer</i>	<i>Public</i>	<i>Float</i>
<i>HarvestCount</i>	<i>Public</i>	<i>Int</i>
<i>DyingTimer</i>	<i>Public</i>	<i>float</i>
<i>HarvestDone</i>	<i>Public</i>	<i>bool</i>

Поля класса *CropData* из таблицы 3.3 определяют следующую логику:

- *growingCrop* – является экземпляром объекта посева;
- *currentGrowthStage* – содержит текущую стадию роста объекта посева;
- *growthRatio* – коэффициент роста, чем больше этот параметр, тем быстрее увеличивается стадия роста;
- *growthTimer* – время роста;
- *harvestCount* – сколько раз можно собрать;
- *dyingTimer* – время до удаления со сцены;
- *harvestDone* – если финальная стадия роста достигнута, то инициализируется значением *true*, если нет, *false*.

Класс *CropData* содержит методы позволяющие обновлять спрайт исходя из текущей стадии роста, а также методы, которые позволяют инициализировать данные о своем состоянии. Эти данные используются для создания объектов на сетке при загрузке сцены.

Сигнатура методов класса *CropData* описана в таблице 3.4.

Таблица 3.4 – Сигнатура методов класса *CropData*

Название	Модификатор доступа	Тип возвращаемого значения	Параметры(тип данных)
<i>Harvest</i>	<i>Public</i>	<i>Crop</i>	–
<i>Save</i>	<i>Public</i>	<i>Void</i>	<i>placedCropData(Placed CropData)</i>
<i>Load</i>	<i>Public</i>	<i>Void</i>	<i>placedCropData(placedCropData), cropDataBase(cropDataBase)</i>

Методы, описанные в таблице 3.4 содержат следующую логику:

- *Harvest* – инкрементирует параметр текущей стадии роста, возвращает объект посева;
- *Save* – инициализирует структуру данных для сохранения своего состояния;
- *Load* – инициализирует свое состояния исходя из сохраненной структуры данных.

GroundData – класс, который содержит параметры состояния полива объекта земли. Его поля описаны в таблице 3.5.

Таблица 3.5 – Поля класса *GroundData*

Название	Модификатор доступа	Тип данных
<i>WaterDuration</i>	<i>Public</i>	<i>float</i>
<i>WaterTimer</i>	<i>Public</i>	<i>float</i>

Поля класса *GroundData* описывают следующую логику:

- *WaterDuration* – константный параметр, который содержит время нахождения спрайта мокрой земли на ячейке;
- *WaterTimer* – содержит текущее время нахождения объекта на сцене.

3.3 Принцип работы программных модулей управления инвентарем

Модуль управления инвентарем отвечает за управления предметами игрока и их взаимодействием с игровой сценой. Характеризуется классами:

- *InventoryBase*;
- *InventoryCell*;
- *PlayerInventory*;
- *ActiveInventory*;

- *ChestInventory*;
- *InventoryStorage*;
- *InventoryItem*;
- *ProductItem*;
- *SeedBagItem*;
- *WateringItem*;
- *Crop*;
- *HoelItem*;
- *WateringItem*;
- *ItemContextData*;
- *Pack*.

InventoryItem – абстрактный класс, который содержит базовые параметры и методы для реализации логики предмета инвентаря. Поля класса описаны в таблице 3.6.

Таблица 3.6 – Поля класса *InventoryItem*

Название	Модификатор доступа	Тип данных
<i>DisplayName</i>	<i>Public</i>	<i>String</i>
<i>Icon</i>	<i>Public</i>	<i>Sprite</i>
<i>UniqueName</i>	<i>Public</i>	<i>String</i>
<i>Color</i>	<i>Public</i>	<i>Color</i>
<i>MaxStackSize</i>	<i>Public</i>	<i>Int</i>
<i>Count</i>	<i>Public</i>	<i>Int</i>
<i>Consumable</i>	<i>Public</i>	<i>Bool</i>
<i>BuyPrice</i>	<i>Public</i>	<i>Int</i>
<i>IsCountTextActive</i>	<i>Protected</i>	<i>bool</i>
<i>VisualPrefab</i>	<i>Public</i>	<i>GameObject</i>
<i>UseSound</i>	<i>Public</i>	<i>AudioClip[]</i>
<i>PlayerAnimatorTriggerUse</i>	<i>Public</i>	<i>string</i>

Поля класса *InventoryItem* из таблицы 3.6 описывают следующую логику:

- *DisplayName* – имя предмета, которое отображается на ячейке.
- *Icon* – картинка предмета;
- *UniqueName* – уникальное имя предмета;
- *Color* – цвет отображаемого имени объекта;
- *MaxStackSize* – максимальное количественное значение одного предмета;
- *Count* – текущее количество предмета;
- *Consumable* – содержит значение *true*, если это расходуемый предмет, *false*, если нет;
- *BuyPrice* – цена покупки предмета;

– *IsCountTextActive* – true, если нужно отображать текущее количество предмета в ячейке;

– *VisualPrefab* – игровой объект, который отображается в руке игрока при взаимодействии;

– *UseSound* – массив звуковых эффектов, которые проигрываются при взаимодействии с предметом инвентаря;

– *PlayerAnimatorTriggerUse* – название анимации, которая проигрывается при взаимодействии с предметом инвентаря.

Для реализации базовой логики взаимодействия игрока с предметом инвентаря содержит методы, сигнатура которых описана в таблице 3.7.

Таблица 3.7 – Сигнатура методов класса *InventoryItem*

Название	Модификатор доступа	Тип возвращаемого значения	Параметры (тип данных)
<i>InitializeCopy</i>	<i>Public</i>	<i>Void</i>	<i>inventoryItem (InventoryItem)</i>
<i>ApplyCondition</i>	<i>Public</i>	<i>Bool</i>	<i>Target(Vector3Int)</i>
<i>Apply</i>	<i>Public</i>	<i>Bool</i>	<i>Target(Vector3Int)</i>
<i>RenderUI</i>	<i>Public</i>	<i>Void</i>	<i>inventoryCell(InventoryCell)</i>
<i>GetData</i>	<i>Public</i>	<i>Inventory ItemData</i>	–
<i>NeedTarget</i>	<i>Public</i>	<i>Bool</i>	–
<i>Clone</i>	<i>Public</i>	<i>Object</i>	–

Методы класса *InventoryItem* содержат следующую логику:

– *InitializeCopy* – инициализирует другой экземпляр такого же типа своими значениями параметров;

– *ApplyCondition* – абстрактный метод, обязывающий наследников реализовать логику соблюдения условий для применения предмета;

– *Apply* – абстрактный метод, обязывающий наследников реализовать логику применения предмета;

– *RenderUI* – виртуальный метод, который содержит логику того, как отображать предмет в ячейке инвентаря;

– *NeedTarget* – виртуальный метод, который определяет, нужна ли цель для применения предмета;

– *Clone* – абстрактный метод, который обязует наследников реализовать интерфейс *IClonable*;

– *GetData* – виртуальный метод, который возвращает данные об предмете для сохранения или загрузки.

InventoryCell – класс, который представляет собой ячейку, которая отображает предмет инвентаря. Сигнатура полей класса *InventoryCell* описана в таблице 3.8.

Таблица 3.8 – Сигнатура полей класса *InventoryCell*

Название	Модификатор доступа	Тип данных
<i>IconElement</i>	<i>Private</i>	<i>Image</i>
<i>TextElement</i>	<i>Private</i>	<i>TextMeshProUGUI</i>
<i>CountTextElement</i>	<i>Private</i>	<i>TextMeshProUGUI</i>
<i>SelectIconElement</i>	<i>Private</i>	<i>Image</i>
<i>GlobalVisualContext</i>	<i>Public</i>	<i>Transform</i>
<i>OriginVisualContext</i>	<i>Public</i>	<i>Transform</i>
<i>BeginDragSiblingIndex</i>	<i>Public</i>	<i>Int</i>
<i>itemResourceDroper</i>	<i>Private</i>	<i>ItemResourceDropper</i>
<i>mouseCursor</i>	<i>Private</i>	<i>MouseCursor</i>

Поля, описанные в таблице 3.8 содержат следующую логику:

- *IconElement* – содержит экземпляр компонента для управления работой изображения иконки предмета;
- *TextElement* – содержит экземпляр компонента для редактирования отображаемого текста названия предмета;
- *CountTextElement* – содержит экземпляр компонента для редактирования текста, который отображает текущее количество предмета;
- *SelectionElement* – содержит экземпляр для управления изображением, отображающим выбранный предмет;
- *GlobalVisualContext* – содержит информацию о компоненте высшего по иерархии объекта интерфейса;
- *OriginVisualContext* – содержит информацию о родительском компоненте объекта интерфейса по отношению к текущему;
- *BeginDragSiblingIndex* – содержит информацию о порядке предмета в инвентаре в момент его перемещения;
- *ItemResourceDropper* – содержит экземпляр, позволяющий выкидывать предмет из инвентаря;
- *MouseCursor* – содержит экземпляр, который позволяет менять иконку курсора мыши;

Класс *IncentoryCell* содержит методы, которые позволяют реализовать логику взаимодействия игрока с ячейкой инвентаря. Сигнатура методов класса *InventoryCell* показана в таблице 3.9.

Таблица 3.9 – Сигнатура методов класса *InventoryCell*

Название	Модификатор доступа	Тип возвращаемого значения	Параметры (тип данных)
<i>Initialize</i>	<i>Public</i>	<i>Void</i>	<i>Global VisualContext(Transform), Inventory Item(InventoryItem)</i>
<i>RegisterEvents</i>	<i>Public</i>	<i>Void</i>	<i>endDragEvent(Action), beginDrag Event(Action<InventoryCell>)</i>
<i>Update</i>	<i>Private</i>	<i>Void</i>	–
<i>OnDrag</i>	<i>Private</i>	<i>Void</i>	<i>eventData(PointerEventData)</i>
<i>OnBeginDrag</i>	<i>Private</i>	<i>Void</i>	<i>eventData(PointerEventData)</i>
<i>OnEndDrag</i>	<i>Private</i>	<i>Void</i>	<i>eventData(PointerEventData)</i>

Методы, описанные в таблице 3.9, организуют следующую логику:

– *Initialize* – инициализирует экземпляр *InventoryCell* предметом инвентаря и контекстом действия;

– *RegisterEvent* – регистрирует методы инвентаря на события ячейки;

– *Update* – отслеживает текущее количество предмета инвентаря, чтобы удалить его, если его количество равняется нулю;

– *OnDrag* – метод, реализующий интерфейс *IDragHandler*, который перемещает ячейку инвентаря за курсором пользователя;

– *OnBeginDrag* – метод, реализующий интерфейс *IBeginDragHandler*, который меняет контекст перемещения ячейки в начале перемещения ячейки курсором;

– *OnEndDrag* – метод, реализующий интерфейс *IEndDragHandler*, который отслеживает пересечения курсора игрока с другими инвентарями в конце перемещения ячейки, перемещает ячейку в контекст другого инвентаря, если есть пересечения с другим инвентарем;

InventoryBase – абстрактный класс, который представляет сущность хранилища предметов инвентаря, следит за расположением ячеек инвентаря. Содержит все необходимые методы и параметры для реализации логики инвентаря.

Сигнатура полей класса *InventoryBase* описана в таблице 3.10

Таблица 3.10 – Сигнатура полей класса *InventoryBase*

Название	Модификатор доступа	Тип данных
<i>Container</i>	<i>Public</i>	<i>Transform</i>
<i>TotalSize</i>	<i>Protected</i>	<i>Int</i>
<i>CurrentSize</i>	<i>Protected</i>	<i>Int</i>
<i>InventoryItems</i>	<i>Protected</i>	<i>List<InventoryItem></i>
<i>InventoryCellFactory</i>	<i>protected</i>	<i>IInventoryCellFactory</i>
<i>ContextRect</i>	<i>Private</i>	<i>RectTransform</i>
<i>OnBeginDragEvent</i>	<i>Protected</i>	<i>Action<InventoryCell></i>
<i>OnEndDragEvent</i>	<i>Protected</i>	<i>OnendDragEvent</i>
<i>MouseCurosr</i>	<i>Private</i>	<i>MouseCursor</i>

Поля класса *InventoryBase* представляют из себя следующую логику:

- *Container* – компонент, представляющий контекст видимости для ячеек, которые находятся в инвентаре;
- *TotalSize* – содержит информацию о максимальном количестве вмещаемых ячеек в инвентаре;
- *CurrentSize* – содержит информацию о текущем количестве предметов, которые находятся в инвентаре;
- *InventoryItems* – список предметов инвентаря;
- *IInventoryCellFactory* – содержит экземпляр, реализующий интерфейс *IInventoryCellFactory*;
- *ContextRect* – содержит компонент, который представляет собой позицию панели инвентаря в пространстве пользовательского интерфейса;
- *OnBeginDragEvent* – содержит ссылки на методы ячеек, которые вызываются в начале взаимодействия с ячейкой;
- *OnEndDragEvent* – содержит ссылки на методы ячеек, которые вызываются в конце взаимодействия с ячейкой;
- *MouseCurosr* – содержит экземпляр класса *MouseCursor* для управления иконкой курсора мыши.

Для реализации взаимодействия игрока с инвентарем класс *InventoryBase* содержит метода. Сигнатура методов класса *InventoryBase* показана в таблице 3.11.

Таблица 3.11 – Сигнатура методов класса *InventoryBase*

Название	Модификатор доступа	Тип возвращаемого значения	Параметры (тип данных)
1	2	3	4
<i>Initialize</i>	<i>Public</i>	<i>Void</i>	<i>InventoryItems(bool)</i>
<i>RegisterDragEvents</i>	<i>Public</i>	<i>Void</i>	<i>inventoryCell(int)</i>

1	2	3	4
<i>AddItem</i>	<i>Public</i>	<i>Bool</i>	<i>InventoryItem</i> (<i>InventoryItem</i>)
<i>RemoveItem</i>	<i>Public</i>	<i>Void</i>	<i>itemIndex(int)</i>
<i>CreateCellForItem</i>	<i>Private</i>	<i>Void</i>	<i>InventoryItem</i> (<i>InventoryItem</i>)
<i>IsFull</i>	<i>Public</i>	<i>Bool</i>	–
<i>OnBeginDragCell</i>	<i>Protected</i>	<i>Void</i>	–
<i>OnEndDrag</i>	<i>Protected</i>	<i>Void</i>	–
<i>OnDrag</i>	<i>Protected</i>	<i>Void</i>	<i>eventData</i> (<i>PointerEventData</i>)
<i>OverwriteInventoryItemsSequence</i>	<i>Private</i>	<i>List<InventoryItem></i>	–
<i>GetItemsContextData</i>	<i>Public</i>	<i>List<ItemContextData></i>	–
<i>OnBeginDrag</i>	<i>Public</i>	<i>Void</i>	<i>eventData</i> (<i>PointerEventData</i>)
<i>OnEndDrag</i>	<i>Public</i>	<i>Void</i>	<i>eventData</i> (<i>PointerEventData</i>)

Методы, описанные в таблице 3.11 содержат следующую логику:

- *Initialize* – инициализирует инвентарь списком предметов;
- *RegisterDragEvents* – регистрирует методы ячейки на события в инвентаре;
- *AddItem* – создает ячейку инвентаря;
- *RemoveItem* – удаляет ячейку инвентаря по индексу;
- *CreateCellForItem* – создает объект ячейки инвентаря;
- *IsFull* – возвращает *true*, если инвентарь заполнен, *false*, если нет;
- *OnBeginDragCell* – создает пустую ячейку;
- *OnEndDrag* – удаляет пустую ячейку;
- *OnDrag* – реализует интерфейс *IDragHandler*;
- *OverwriteInventoryItemsSequence* – переписывает порядок ячеек;
- *GetItemsContextData* – возвращает список данных о каждой ячейке;
- *OnBeginDrag* – реализует интерфейс *IBeginDragHandler*, меняет иконку курсора;
- *OnEndDrag* – реализует интерфейс *IEndDragHandler*, меняет иконку курсора.

3.3 Принцип работы модуля экономической и социальной системы

Модуль экономической системы содержит классы и структуры данных для реализации механик покупки и продажи продукции. Характеризуется следующими классами: *TradeService*, *TradePanel*, *TradeElement*.

TradeService – класс, который содержит логику для соблюдения условий продажи и покупки продукции. Сигнатура методов класса *TradeService* описана в таблице 3.12.

Таблица 3.12 – Сигнатура методов класса *TradeService*

Название	Модификатор доступа	Тип возвращаемого значения	Параметры (тип данных)
<i>BuyCondition</i>	<i>Public</i>	<i>Bool</i>	<i>Item (InventoryItem)</i>
<i>Buy</i>	<i>Public</i>	<i>Void</i>	<i>Item (InventoryItem)</i> , <i>Amount(int)</i>
<i>SellCondition</i>	<i>Public</i>	<i>Bool</i>	<i>Item (InventoryItem)</i>
<i>Sell</i>	<i>Public</i>	<i>Void</i>	<i>context</i> <i>Data(ItemContextData)</i>

Методы класса *TradeService*, описанные в таблице 3.12 содержат следующую логику:

– *BuyCondition* – возвращает *true*, если есть возможность купить продукцию;

– *Buy* – добавляет в инвентарь игрока предмет, при этом тратит игровую валюту в соответствии со стоимостью покупки продукции;

– *SellCondition* – возвращает *true*, если продукцию можно продать;

– *Sell* – удаляет из инвентаря предмет для продажи и добавляет игровую валюту в размере стоимости продажи предмета.

TradePanel – класс, который представляет панель для осуществления действий пользователя для покупки и продажи продукции. Сигнатура методов класса *TradePanel* описана в таблице 3.13.

Таблица 3.13 – Сигнатура методов класса 3.13

Название	Модификатор доступа	Тип возвращаемого значения	Параметры (тип данных)
1	2	3	4
<i>OnBuy</i>	<i>Public</i>	<i>Void</i>	–
<i>OnSell</i>	<i>Public</i>	<i>Void</i>	–

1	2	3	4
<i>OnExit</i>	<i>Public</i>	<i>Void</i>	–
<i>ClearElements</i>	<i>Private</i>	<i>Void</i>	–

Методы, описанные в таблице 3.13, содержат следующую логику:

- *OnSell* – создает элементы взаимодействия для каждого предмета, который можно продать в инвентаре игрока;
- *OnBuy* – создает элементы взаимодействия на панели для покупки предмета;
- *OnExit* – скрывает панель покупки и продажи;
- *ClearElements* – очищает элементы взаимодействия с панели покупки и продажи.

TradeElement – класс, который представляет элемент взаимодействия на панели покупки и продажи предметов. Сигнатура полей класса *TradeElement* описана в таблице 3.14.

Таблица 3.14 – Сигнатура полей класса *TradeElement*

Название	Модификатор доступа	Тип данных
<i>IconElement</i>	<i>Public</i>	<i>Image</i>
<i>IconName</i>	<i>Public</i>	<i>TextMeshProUGUI</i>
<i>ButtonText</i>	<i>Public</i>	<i>TextMeshProUGUI</i>
<i>ButtonElement</i>	<i>Public</i>	<i>Button</i>

Поля, описанные в таблице 3.14, содержат следующую логику:

- *IconElement* – содержит компонент для управления отображением;
- *IconName* – содержит компонент для управления текстом;
- *ButtonText* – содержит компонент для управления текстом кнопки;
- *ButtonElement* – содержит компонент позволяющий управлять объектом кнопки.

Модуль социальной системы отвечает за взаимодействие пользователя с NPC в формате чата. Характеризуется классами: *ChatService*, *ChatPanel*, *MessagePanel*. В этом модуле используются библиотека *OpenAI* и *SharpToken*. Библиотека *OpenAI* нужна, чтобы организовывать запросы к API ChatGPT. Библиотека *SharpToken* применяется для оптимизации запросов путем подсчитывания количества токенов в текущем контексте.

ChatService – класс, который реализует запросы к API ChatGPT с помощью библиотеки *OpenAI*. Сигнатура методов класса *ChatService* описана в таблице 3.14.

Таблица 3.14 – Сигнатура методов класса *ChatService*

Название	Модификатор доступа	Тип возвращаемого значения	Параметры (тип данных)
<i>AddUserInput</i>	<i>Public</i>	<i>Void</i>	<i>Text(string)</i>
<i>AddSystemInput</i>	<i>Public</i>	<i>Void</i>	<i>Text(string)</i>
<i>GetResponseAsync</i>	<i>Public</i>	<i>String</i>	–
<i>GetContextText</i>	<i>Public</i>	<i>String</i>	–
<i>ClearChat</i>	<i>Public</i>	<i>Void</i>	–

Методы, описанные в таблице 3.14, содержат следующую логику:

- *AddUserInput* – добавляет текст сообщения игрока в роли «user»;
- *AddSystemInput* – добавляет текст сообщения для чат-бота в роли «system»;
- *GetResponseAsync* – асинхронный метод, который получает ответ от чат-бота;
- *ClearChat* – очищает историю сообщений.

MessagePanel – класс, который представляет из себя панель, которая содержит сообщение, введенное игроком или сообщения полученное в качестве ответа чат-бота. Содержит поля, являющиеся компонентами объекта панели сообщения, которые позволяют управлять контентом панели. Реализует логику написания сообщения с помощью методов.

ChatPanel – класс, который представляет собой место расположения панелей сообщения. Содержит методы, реализующие логику создания панели сообщения в результате ввода пользователя.

3.4 Паттерны в приложении «*Farmer's Valley*»

При разработке игрового приложения «*Farmer's Valley*» использовались различные паттерны проектирования для обеспечения гибкости, расширяемости и поддерживаемости кода [5]. Основные паттерны, использованные в проекте, включают:

- *Singleton*;
- *Observer*;
- *FactoryMethod*;
- *State*;
- *ServiceLocator*.

«Одиночка» (*Singleton*, Синглтон) – порождающий паттерн, который гарантирует, что для определенного класса будет создан только один объект, а также предоставит к этому объекту точку доступа.

Класс *LoadedData* реализует этот паттерн проектирования, чтобы иметь возможность обратиться к сохраненным данным при смене сцены.

Паттерн «Наблюдатель» (*Observer*) представляет поведенческий шаблон проектирования, который использует отношение «один ко многим». В этом отношении есть один наблюдаемый объект и множество наблюдателей. И при изменении наблюдаемого объекта автоматически происходит оповещение всех наблюдателей.

Этот паттерн использует класс *SettingsMenu*, чтобы реагировать на изменения настроек игровой среды в пользовательском интерфейсе.

Фабричный метод (*Factory Method*) – это паттерн, который определяет интерфейс для создания объектов некоторого класса, но непосредственное решение о том, объект какого класса создавать происходит в подклассах. То есть паттерн предполагает, что базовый класс делегирует создание объектов классам-наследникам.

В проекте есть множество классов, реализующих этот паттерн. Такие классы создают объекты на игровой сцене, а также элементы интерфейса.

Состояние (*State*) – шаблон проектирования, который позволяет объекту изменять свое поведение в зависимости от внутреннего состояния.

Этот паттерн реализует класс *ItemResource*, который реализует несколько состояний внутри себя изменяя его поведение.

Локатор служб (*Service Locator*) – это шаблон проектирования, используемый в разработке программного обеспечения для инкапсуляции процессов, связанных с получением какого-либо сервиса с сильным уровнем абстракции.

Такой паттерн реализует класс *FactoriesProvider*, который раздает реализации фабрик [6].

4 ВАЛИДАЦИЯ И ВЕРИФИКАЦИЯ РЕЗУЛЬТАТОВ РАБОТЫ ИГРОВОГО ПРИЛОЖЕНИЯ «*FARMER'S VALLEY*»

4.1 Принципы работы пользовательского графического интерфейса и основные механики игрового приложения «*Farmer's Valley*»

После запуска игры открывается сцена с главным меню. На рисунке 4.1 представлен кадр после запуска игры.



Рисунок 4.1 – Кадр после запуска игры

Сцена с главным меню является стартовой сценой с соответствующим графическим интерфейсом.

Пользователь может взаимодействовать со следующими элементами графического интерфейса:

- кнопка *NewGame*;
- кнопка *LoadGame*;
- кнопка *Settings*;
- кнопка *WorkSpace*.

При нажатии на кнопку *NewGame* перед пользователем откроется панель создания нового игрового состояния, где нужно ввести названия нового игрового состояния. После ввода названия нового игрового состояния игрок может нажать на кнопку *Back*, чтобы вернуться к изначальному виду или же нажать на кнопку *Continue*, чтобы перейти к загрузке игровой сцены. Однако в случае нажатия на

кнопку *Continue* пользователь не всегда перейдет к загрузке игровой сцены. Чтобы перейти к загрузке сцены должны быть выполнены некоторые условия:

- название нового игрового состояния не должно начинаться с цифры и содержать пробелы;
- название игрового состояния не должно повторяться.

На рисунке 4.2 показана панель создания нового игрового состояния.

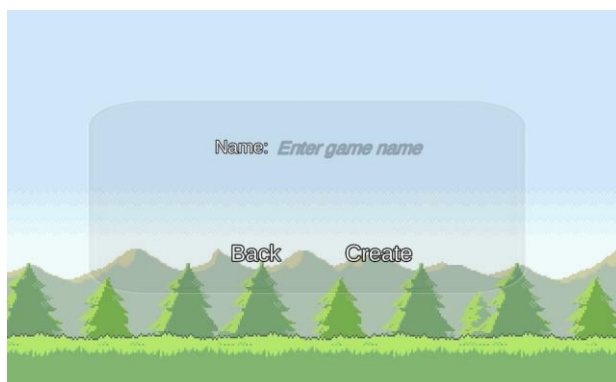


Рисунок 4.2 – Панель создания нового игрового состояния

При нажатии на кнопку *LoadGame* пользователю откроется панель загрузки игрового состояния. На рисунке 4.3 представлена панель загрузки игрового состояния.



Рисунок 4.3 – Панель загрузки игровых состояний

В панели загрузки игровых состояний отображаются все сохраненные игровые состояния, которые имеются на компьютере. Элемент игрового состояния представляет из себя название игрового состояния, игровую валюту, которая осталась у игрока при сохранении игрового состояния, и иконку игрового персонажа. Содержит элементы интерфейса:

- кнопка *Load*;
- кнопка *Delete*.

При нажатии на кнопку *Load* начнется загрузка игровой сцены с учетом сохраненных данных. При нажатии на кнопку *Delete*, сохраненное игровое состояние удалится с компьютера и не появится при открытии панели загрузки игрового состояния. При нажатии на кнопку *Back*, пользователь вернется к изначальному виду.

После нажатия на кнопку *Settings* пользователю откроется панель настроек игровой среды. На рисунке 4.4 представлена панель настройки игровой среды.

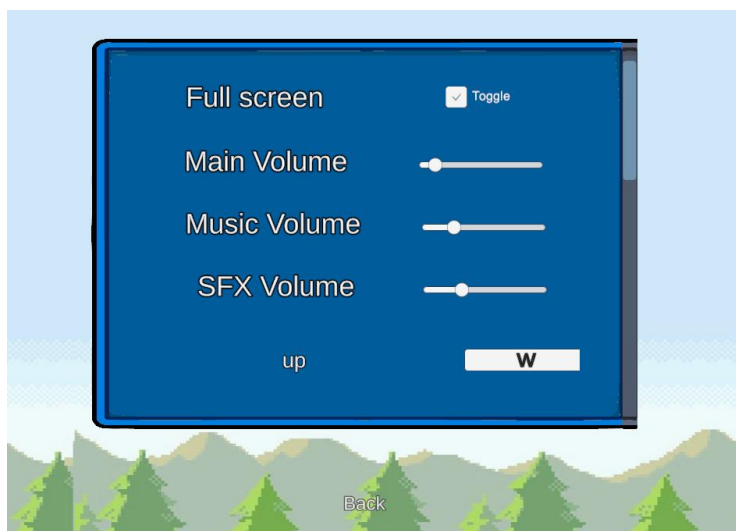


Рисунок 4.4 – Панель настройки игровой среды

На панели настройки игровой среды присутствуют следующие элементы графического интерфейса:

- *Screen Resolution*;
- *Full Screen*;
- *Main Volume*;
- *SFX Volume*;

– элементы смены клавиш взаимодействия с игровым миром.

При взаимодействии с элементом *Screen Resolution* пользователь может выбрать любое из представленных разрешений экрана. При взаимодействии с элементом *Full Screen* пользователь может поменять режим игрового приложения на оконный или полноэкранный. При взаимодействии с элементом *Main Volume* можно настроить громкость звука, которая влияет на остальные значения звука выполняю функции микшера. При взаимодействии с элементом *Music Volume* можно влиять на громкость фоновой музыки напрямую. При взаимодействии с элементом *SFX Volume* можно изменить громкость звуковых эффектов во время ходьбы и остальных.

С помощью элементом смены клавиш взаимодействия можно поменять клавишу взаимодействия путем нажатия на кнопку выбранного действия и нажатия новой клавиши после вывода сообщения об ожидании ввода. Нажав на кнопку *Back* пользователь вернется к исходному виду. Стандартное назначение клавиш в настройках показано в таблице 4.1.

Таблица 4.1 – Стандартное назначение клавиш в настройках клавиш взаимодействия

Клавиша	Действие
<i>I</i>	Выбор первой ячейки
<i>2</i>	Выбор второй ячейки
<i>3</i>	Выбор третьей ячейки
<i>4</i>	Выбор четвертой ячейки
<i>W</i>	Перемещение персонажа вверх
<i>A</i>	Перемещение персонажа вправо
<i>S</i>	Перемещение персонажа вниз
<i>D</i>	Перемещение персонажа влево
<i>LMB</i>	Взаимодействие
<i>Tab</i>	Скрыть/Открыть вспомогательный инвентарь

Клавиши, представленные в таблице 4.1, можно перенастроить как на главной сцене, так и на игровой. При этом, после открытия панели настроек доступные для перенастройки клавиши взаимодействия будут называться в соответствии с выбранной раскладкой клавиатуры во время открытия панели настроек.

При нажатии на клавишу *WorkSpace* пользователь закроет приложение.

После выбора клавиш, загружающих игровую сцену, пользователь увидит кадр, представленный на рисунке 4.5.

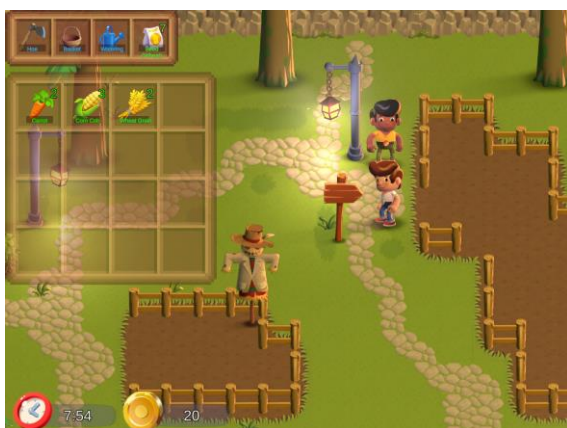


Рисунок 4.5 – Кадр из игровой сцены

Перед глазами игрока появляется игровой уровень. На нем виднеется игровой персонаж и *NPC*. Также можно увидеть земельные участки, ограниченные забором. На рисунке 4.6 представлен результат использования предмета мотыга, лейка и мешок с семенами.



Рисунок 4.6 – Результат использования предмета мотыга, лейка, мешок с семенами

На таких участках игрок может взаимодействовать с предметами инвентаря для рыхления земли, полива земли и посадки семян.

Слева в верхнем углу экрана пользователь может увидеть активный инвентарь и вспомогательный. В них находятся предметы для взаимодействия. Синим цветом обозначены инструменты, такие как мотыга, лейка и корзинка, а зеленым цветом обозначены семена и предметы сбора. Чтобы использовать предметы вспомогательного инвентаря нужно перетащить их в активный инвентарь путем выбора определенного предмета из вспомогательного инвентаря и последующего зажатия левой кнопки мыши для перетаскивания предмета в активный инвентарь. Если конечный инвентарь, в который игрок пытался перенести определенный предмет был заполнен, то выбранный предмет будет помещен в ближайшую ячейку инвентаря в зависимости от расположения курсора мыши.

Нажав на *NPC*, рядом появится панель, на которой находятся графические элементы интерфейса:

- *Talk*;
- *Trade*.

На рисунке 4.7 представлен результат нажатия игроком на *NPC*.

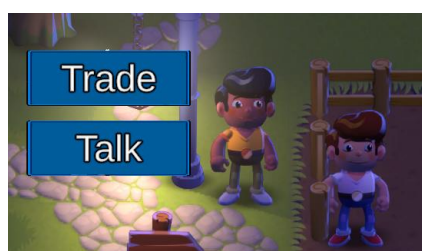


Рисунок 4.7 – Результат нажатия на *NPC*

При нажатии на кнопку *Trade* откроется панель покупки и продажи. С помощью элементов графического интерфейса на панели покупки и продажи игрок может продавать и покупать продукцию. После нажатия на кнопку *Trade* панель взаимодействия с *NPC* будет закрыта и откроется при следующем нажатии на *NPC*.

На панели покупки и продажи находятся следующие элементы графического интерфейса:

- кнопка *Buy*;
- кнопка *Sell*.

В случае нажатия на кнопку *Sell* перед пользователем появляются предметы инвентаря, которые он может продать, чтобы получить игровую валюту.

Результат нажатия на кнопку *Sell* показан на рисунке 4.8.



Рисунок 4.8 – Результат нажатия на кнопку *Sell*

Перед пользователем появляются предметы инвентаря, которые он может продать, чтобы получить игровую валюту. На этих элементах отображается название продукции и сколько игрок получит игровой валюты за продажу текущего количества определенной продукции. Чтобы продать ту или иную продукцию пользователю нужно нажать на кнопку, которая помечена зеленым цветом. После чего будет начислена игровая валюта, а предмет продажи будет удален из инвентаря.

При нажатии на кнопку *Buy* перед пользователем появляются элементы взаимодействия, с помощью которых можно купить и добавить в инвентарь мешок семян. За одно нажатие игрок может приобрести одно семя. В случае, если игровой валюты не хватает, то кнопка покупки для пользователя будет не доступна.

Чтобы закрыть панель покупки и продажи пользователю нужно нажать на кнопку, обозначенную крестиком, тогда панель закроется.

При нажатии на кнопку *Talk* открывается чат-панель, которая содержит элементы графического интерфейса для реализации общения с *NPC* в формате чата. Содержит следующие элементы:

- область для написания текста;
- кнопка *Enter*;
- кнопка *Reset*.

Чат-панель представлена на рисунке 4.9.

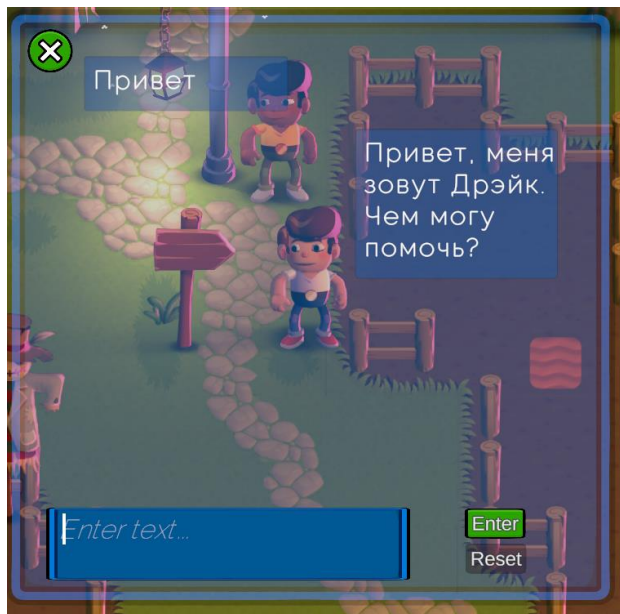


Рисунок 4.9 – Чат-панель

При нажатии на кнопку *Enter* отправляется сообщение из введенного текста в области ввода. Сообщение ограничено сотней символов.

Кнопка *Reset* будет доступна тогда, когда величина контекста общения с *NPC* будет превышена. После ее нажатия очистится история сообщений и будет доступна кнопка *Enter*.

Пользовательский графический интерфейс является основным инструментом пользователя для взаимодействия с игровым миром.

При эксплуатации игры было проанализировано соответствие техническому заданию. Так же была оценена целостность геймплея и удобство в управлении.

4.2 Виды тестирования игр

Перед публикацией финальной версии разработанное игровое приложение было несколько раз протестировано с помощью различных методов тестирования.

Существует перечень видов тестирования игровых приложений:

- функциональное тестирование – тестирование, целью которого является выявление отклонений от функциональных требований к игровому приложению, его фрагменту или отдельной функции [7]. Данное тестирование сводится к многократному запуску и прохождению игры, выявлению неполадок, багов либо несоответствий с заявленным функционалом и поиску методов или способов решения выявленных проблем и устранения неполадок. Данный тип тестирования применялся на протяжении всех этапов разработки игрового приложения и является основным;

- нагрузочное тестирование – тестирование игровых приложений, при котором воссоздаются ситуации, которые требуют большой вычислительной мощности и максимально нагружают устройство, на котором запущено игровое приложение. Таким образом, тестировщик может проверить производительность системы в стрессовой ситуации или на потенциально слабых устройствах. Данный метод облегчает поиск потенциально ненадёжных фрагментов кода, которые требуют оптимизации. Для данного вида тестирования моделировались ситуации, которые потенциально могли перегружать возможности мобильного устройства. Например, в игровую сцену добавлялось большое количество предметов экипировки и противников, что моделировало ситуацию перегрузки сцены графикой, если игрок решил бы пробежаться по локациям и собрать за собой большое количество преследующих его противников для более острых ощущений от игрового процесса. Благодаря подобным тестам удалось убедиться в исправном функционировании игрового приложения при избытке объектов и графики в сцене;

- тестирование на совместимость – вид нефункционального тестирования, основной целью которого является проверка корректной работы продукта в определенном окружении. Проверке подвергалась возможность запуска игрового приложения на мобильных устройствах, имеющих различные технические характеристики.

4.3 Функциональное тестирование

Как было упомянуто выше функциональное тестирование – это вид тестирования ПО, цель которого в проверке реализуемости функциональных требований, а именно способности ПО в определенных условиях корректно

решать поставленные перед ним задачи. Функциональные требования определяют, что именно делает ПО, какие задачи оно решает. Тестирование было проведено путем запуска игрового приложения при определенных тестировщиком условиях для наблюдения и оценки корректности работы различных аспектов игрового приложения. В процессе функционального тестирования производился анализ приложения для выявления несоответствий между существующими деталями функционирования игрового приложения и требованиями к их корректному функционированию.

Тест № 1. Цель: проверить запуск приложения.

Ожидаемый результат: игра должна запускаться на мобильных устройствах с ОС *Android* 4.0 и выше.

Вывод: ожидаемый и полученный результат совпали, цель теста достигнута.

Тест № 2. Цель: проверить соответствие отображения игрового интерфейса на устройстве с отображением в среде разработки.

Ожидаемый результат: на экране отображен интерфейс игры, отображаются: панель игрового интерфейса, содержащая панель активного инвентаря, панель вспомогательного инвентаря, панель с текущим временем суток и игровой валютой. Все элементы интерфейса выглядят уместно и не подвергнуты деформации.

Вывод: ожидаемый и полученный результат совпадают, цель теста достигнута.

Тест № 3. Цель: проверить работоспособность кнопок главного меню.

Ожидаемый результат: в зависимости от нажатой в главном меню кнопки происходит запуск игровой сцены, отображение окон настройки игровой среды создания нового игрового состояния, окна загрузок игровых состояний, или выход из приложения.

Вывод: ожидаемый и полученный результат совпадают, цель теста достигнута.

Тест № 4. Цель: проверить соответствие отображения игрового интерфейса на устройстве с отображением в среде разработки.

Ожидаемый результат: на экране отображен интерфейс игры, отображаются: панель игрового интерфейса, содержащая панель активного инвентаря, панель вспомогательного инвентаря, панель с текущим временем суток и игровой валютой. Все элементы интерфейса выглядят уместно и не подвергнуты деформации.

Вывод: ожидаемый и полученный результат совпадают, цель теста достигнута.

Тест № 5. Цель: проверить работоспособность игрового меню.

Ожидаемый результат: при нажатии кнопки паузы ввод клавиш, отвечающих за взаимодействия с игровой средой, блокируется, появляется панель с функциональными кнопками меню.

Вывод: ожидаемый и полученный результат совпадают, цель теста достигнута.

Тест № 6. Цель: проверить работоспособность передвижения игрока по игровой сцене.

Ожидаемый результат: при нажатии на клавиши перемещения персонаж передвигается, при этом, если клавиши перемещения не нажаты персонаж поворачивается в сторону курсора мыши. Также персонаж игрока не должен проходить сквозь объекты сцены, блокирующие проход.

Вывод: персонаж игрока успешно перемещается по игровой сцене, ожидаемый и полученный результат совпадают, цель теста достигнута.

Тест № 7. Цель: проверить взаимодействие игрока с активным инвентарем. При этом в случае нажатия клавиш выбора предметов должна появляться иконка текущего выбранного предмета.

Ожидаемый результат: при нажатии на один из предметов, текущий предмет передвигается за курсором, если клавиша перетаскивания отпущена, то предмет перемещается в ближайшую ячейку, также при нажатии клавиш выбора предметов, текущий выбранный предмет помечается иконкой выбранного предмета.

Вывод: игрок перемещает предметы инвентаря корректно, при нажатии клавиш выбора предмета текущий выбранный предмет помечается иконкой выбранного предмета, ожидаемый и полученный результат совпадают, цель теста достигнута.

Тест № 8. Цель: проверить корректность функционирования механизма посадки, сбора и полива.

Ожидаемый результат: при выборе предмета мотыга и нажатии на клавишу взаимодействия ставится объект рыхлой земли, при выборе предмета лейка и нажатии на объект рыхлой земли появляется объект политой земли, при выборе мешка с семенами и нажатии на объект рыхлой земли появляется объект семени, при выборе предмета корзинка и нажатии клавиши взаимодействия во время наведения на объект семени, в случае, если семя достигло последней стадии роста, объект семени уничтожается.

Вывод: при взаимодействии с землей с предметом мотыга, ставится объект земли, при взаимодействии с объектом земли с выбранным предметом лейка, появляется объект политой земли, при нажатии на объект рыхлой земли с выбранным предметом мешок с семенем появляется объект семени, при нажатии на объект семени в последней стадии роста с выбранным предметом корзинка

объект семени уничтожается, таким образом, ожидаемый и достигнутый результат совпадают, цель достигнута.

Тест № 9. Цель: проверить функционирование открытия и закрытие панели купли и продажи продукции.

Ожидаемый результат: при нажатии на *NPC* появляется интерфейс с кнопкой *Trade*, нажав на которую появляется панель покупки и продажи, а после нажатия на кнопки закрытия панель покупки и продажи скрывается.

Вывод: панель покупки и продажи продукции корректно открывается и закрывается, ожидаемый и полученный результат совпадают, цель теста достигнута.

Тест № 10. Цель: проверить корректность покупки и продажи предметов в панели торговли.

Ожидаемый результат при нажатии на кнопку *Buy* в панели появляются предметы для покупки, при нажатии на клавишу покупки одного из предметов тратится игровая валюта, а предмет появляется в инвентаре, при нажатии кнопки *Sell* появляются предметы для продажи из инвентаря игрока, при нажатии на клавишу продажи одного из предметов определенный предмет инвентаря удаляется и начисляется игровая валюта:

Вывод: при покупке и продаже определенный предметов корректно начисляется и снимается игровая валюта, определенные предметы добавляются и удаляются из инвентаря, ожидаемый и полученный результат совпадают, цель теста достигнута.

Тест № 11. Цель: проверить функционал открытия и закрытия чат-панели.

Ожидаемый результат: при нажатии кнопкой взаимодействия на *NPC* появляется кнопка *Talk*, при нажатии на которую открывается чат-панель, при нажатии на кнопку закрытия чат-панели, панель закрывается.

Вывод: чат-панель корректно открывается и закрывается, ожидаемый и полученный результат совпадают, цель теста достигнута.

Тест № 12. Цель: проверить работу чат-панели, функциональность отправки и получения сообщений.

Ожидаемый результат: игрок вводит сообщение в области ввода текста, после чего нажимает на кнопку *Enter* и сообщение появляется в панели справа, во время получения ответа кнопка *Enter* блокируется, после чего снова активируется, при достижении максимально доступного размера контекста активируется кнопка *Reset*, при нажатии на которую история сообщений удаляется, после чего кнопка *Enter* вновь оказывается доступной для взаимодействия.

Вывод: ввод и вывод сообщения отображается корректно, при превышении количества контекста доступна кнопка *Reset*, при нажатии на которую история сообщения очищается, после чего кнопка *Enter* снова доступна для отправки

сообщения, ожидаемый и полученный результат совпадают, цель теста достигнута.

4.3 Юзабилити-тестирование

Юзабилити-тестирование – это исследование, выполняемое с целью определения степени удобства и понятности приложения для его потенциальных пользователей, другими словами, это метод оценки удобства продукта в использовании, основанный на привлечении пользователей в качестве тестирующих, испытателей и суммировании полученных от них выводов. Юзабилити-тестирование является распространенным подходом при тестировании приложений.

Метод исследования «Мысли вслух». «Мысли вслух» – один из самых популярных методов юзабилити-тестирования был выбран в качестве метода исследования. Метод «Мысли вслух» позволяет понять, как пользователь подходит к интерфейсу и какими соображениями он руководствуется, используя функционал приложения.

План проведения тестирования:

- планирование – разработка заданий и набор участников тестирования;
- проведение тестирования;
- анализ полученных данных.

Методика проведения тестирования. Участник тестирования получает список заданий целиком. Участник должен ознакомиться со списком, после чего приступить к самостоятельному выполнению, проговаривая свои мысли, чувства, мнения, ощущения, возникающие в процессе взаимодействия с игрой.

Задания для респондентов продемонстрированы в таблице 4.1.

Таблица 4.1 – Протокол заданий

№ задания	Отправная точка	Задание
1	2	3
1	Сцена меню	Ознакомиться с главным меню
2	Сцена меню	Запустить игровую сцену
3	Игровая сцена	Переместить персонажа
4	Игровая сцена	Открыть и закрыть вкладки инвентаря
5	Игровая сцена	Использовать предмет мотыга
6	Игровая сцена	Использовать предмет лейка
7	Игровая сцена	Использовать предмет корзинка
8	Игровая сцена	Посадить семя
9	Игровая сцена	Поговорить с NPC
10	Игровая сцена	Продать предмет

1	2	3
11	Игровая сцена	Купить предмет
12	Игровая сцена	Засадить один из участков
13	Игровая сцена	Изменить настройки звука и разрешения
14	Игровая сцена	Поменять управление взаимодействиями

Было принято решение отбирать для тестирования людей различного возраста, увлечений и игрового опыта.

Все респонденты успешно справились с задачами, отметив удобство интерфейса. Гармоничность оформления и интересный геймплей.

5 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ИГРОВОГО ПРИЛОЖЕНИЯ «FARMER'S VALLEY»

5.1 Техничко-экономическое обоснование целесообразности разработки программного продукта и оценка его конкурентоспособности

Игровое приложение предназначено для удовлетворения досуга пользователя и время проведения. В ходе использования пользователь погружается в процессе игры в фермерскую деятельность.

Существуют другие крупные проекты, которые целенаправленно ориентированы на ту же целевую аудиторию. Эти проекты являются аналогом или базовым эталоном для разработанного программного обеспечения, но они имеют большие недостатки, выраженные низкой частотой обновлений, стоимостью, отсутствием возможности играть на различных операционных системах. Исходя из анализа существующих проектов можно сделать вывод о том, что разработка продукта целесообразна.

Техническая прогрессивность разрабатываемого программного продукта определяется коэффициентом эквивалентности ($K_{эк}$). Расчет данного коэффициента осуществляется путем сравнения технического уровня разрабатываемого программного продукта по отношению к эталонному уровню программного продукта данного направления с использованием формулы (Д.1).

Результат расчета коэффициента эквивалентности приведен в таблице 5.1. Полученное значение коэффициента эквивалентности больше единицы, следовательно, разрабатываемый программный продукт является технически прогрессивным.

Таблица 5.1 – Расчет коэффициентов эквивалентности

Наименование параметра	Вес параметра, β	Значения параметра			$\frac{P_6}{P_3}$	$\frac{P_H}{P_3}$	$\beta \frac{P_6}{P_3}$	$\beta \frac{P_H}{P_3}$
		P_6	P_H	P_3				
Объем памяти	0,3	10	8	7	1,43	1,1	0,43	0,33
Время обработки данных	0,4	0,8	0,5	0,2	4	2,5	1,6	1
Отказы	0,6	2	1	1	2	1	1,2	0,6
Итого							3,23	1,93
Коэффициент эквивалентности							$3,23/1,93 = 1,673$	

Далее рассчитывается коэффициент изменения функциональных возможностей ($K_{ф.в}$) нового программного обеспечения по формуле (Д.3). Расчет коэффициента изменения функциональных возможностей нового программного продукта приведен в таблице 5.2.

Таблица 5.2 – Расчет коэффициента изменения функциональных возможностей

Наименование показателя	Балльная оценка базового ПП	Балльная оценка нового ПП
1	2	3
Объем памяти	4	4
Быстродействие	3	4
Удобство интерфейса	2	5
Степень утомляемости	3	2
Производительность труда	2	4
Итого	17	23
Коэффициент функциональных возможностей	$23/17 = 1,35$	

Новый программный продукт превосходит по своим функциональным возможностям базовый в 1,35 раза.

Конкурентоспособность нового программного продукта по отношению к базовому можно оценить с помощью интегрального коэффициента конкурентоспособности, учитывающего все ранее рассчитанные показатели. Для расчета конкурентоспособности нового программного продукта по отношению к базовому была использована соответствующая формула (Д.4).

Коэффициент цены потребления рассчитывается как отношение договорной цены нового программного продукта к договорной цене базового (таблица 5.3).

Новый программный продукт превосходит по своим функциональным возможностям базовый в 1,35 раза.

Конкурентоспособность нового программного продукта по отношению к базовому можно оценить с помощью интегрального коэффициента конкурентоспособности, учитывающего все ранее рассчитанные показатели. Для расчета конкурентоспособности нового программного продукта по отношению к базовому была использована соответствующая формула (Д.4).

Коэффициент цены потребления рассчитывается как отношение договорной цены нового программного продукта к договорной цене базового (таблица 5.3).

Таблица 5.3 – Расчет уровня конкурентоспособности нового ПП

Коэффициенты	Значение
1	2
Коэффициент эквивалентности	1,673
Коэффициент изменения функциональных возможностей	1,35
Коэффициент соответствия нормативам	1
Коэффициент цены потребления	0,90
Интегральный коэффициент конкурентоспособности	$(1,673 \cdot 1,35 \cdot 1)/0,90 = 2,5$

Интегральный коэффициент конкурентоспособности ($K_{\text{и}}$) больше единицы, это значит, что новый программный продукт является более конкурентоспособным, чем базовый.

5.2 Оценка трудоемкости работ по созданию программного обеспечения

Общий объем программного обеспечения (V_o) определяется исходя из количества и объема функций, реализуемых программой, по каталогу функций программного обеспечения по формуле (Д.5).

Уточненный объем программного обеспечения (V_y) определяется по формуле (Д.6).

Результаты произведённых вычислений объема функций ПО представлены в таблице 5.4.

Таблица 5.4 – Перечень и объем функций ПО

Код функций	Наименование (содержание) функций	Объем функции строк исходного кода	
		по каталогу (V_o)	уточненный (V_y)
1	2	3	4
101	Организация ввода информации	150	300
102	Контроль, предварительная обработка и ввод информации	688	800

Продолжение таблицы 5.4

1	2	3	4
107	Организация ввода-вывода информации в интерактивном режиме	320	500
109	Управление вводом-выводом	2400	665
305	Формирование файла	2460	70
303	Обработка файлов	1100	620
405	Система настройки ПО	370	100
501	Монитор ПО (управление работой компонентов)	1340	800
506	Обработка ошибочных сбойных ситуаций	1720	400
507	Обеспечение интерфейса между компонентами	1820	260
702	Расчетные задачи (расчет режимов обработки)	1330	300
706	Предварительная обработка, печать	470	350
707	Графический вывод результатов	590	720
709	Изменение состояния ресурсов в интерактивном режиме	630	480
Итого		15388	6365

ПО относится ко второй категории сложности.

На основании принятого к расчету (уточненного) объема (V_y) и категории сложности ПО определяется нормативная трудоемкость ПО (T_n) выполняемых работ, представлена в таблице 5.5.

Таблица 5.5 – Нормативная трудоемкость на разработку ПО (T_n)

Уточнённый объем, V_y	3-я категория сложности ПО	Номер нормы
6365	263	53

Дополнительные затраты труда, связанные с повышением сложности разрабатываемого ПО, учитываются посредством коэффициента повышения сложности ПО (K_c). K_c рассчитывается по формуле (Д.7):

$$K_c = 1 + 0,06 = 1,06.$$

Влияние фактора новизны на трудоемкость учитывается путем умножения нормативной трудоемкости на соответствующий коэффициент, учитывающий новизну ПО (K_n).

Разработанная программа обладает категорией новизны Б, а значение $K_n = 0,72$.

Степень использования в разрабатываемом ПО стандартных модулей определяется их удельным весом в общем объеме ПО.

В данном программном комплексе используется до 45% стандартных модулей, что соответствует значению коэффициента $K_t = 0,65$.

Программный модуль разработан с помощью объектно-ориентированных технологий, что соответствует коэффициенту, учитывающему средства разработки ПО, $K_{y.p} = 0,55$. Значения коэффициентов удельных весов трудоемкости стадий разработки ПО в общей трудоемкости ПО определяются с учетом установленной категории новизны ПО и приведены в таблице 5.6.

Добавлено примечание ([П1]):

Таблица 5.6 – Значения коэффициентов удельных весов трудоемкости стадий разработки ПО в общей трудоемкости

Категория новизны ПО	Без применения CASE-технологий				
	Стадии разработки ПО				
	ТЗ	ЭП	ТП	РП	ВН
	Значения коэффициентов				
	$K_{т.з}$	$K_{э.п}$	$K_{т.п}$	$K_{р.п}$	$K_{в.н}$
Б	0,10	0,20	0,30	0,30	0,10

Нормативная трудоемкость ПО (T_n) выполняемых работ по стадиям разработки корректируется с учетом коэффициентов: коэффициента повышения сложности ПО (K_c), коэффициента новизны ПО (K_n), коэффициента степени использования стандартных модулей (K_t), коэффициент средств разработки ПО ($K_{y.p}$). Данные коэффициенты определяются для стадии ТЗ по формуле (Д.8), для стадии ЭП по формуле (Д.9), для стадии ТП по формуле (Д.10), для стадии РП по формуле (Д.11), для стадии ВН по формуле (Д.12). Коэффициенты K_n , K_c и $K_{y.p}$ вводятся на всех стадиях разработки, а коэффициент K_t вводится только на стадии РП.

Для уменьшения общей трудоёмкости разработки введем коэффициент используемости разработанного ПО, который равен 0,1.

$$T_{y.т.з} = 263 \cdot 0,1 \cdot 1,06 \cdot 0,72 \cdot 0,55 = 11 \text{ чел.-дн.},$$

$$T_{y.э.п} = 263 \cdot 0,2 \cdot 1,06 \cdot 0,72 \cdot 0,55 = 22 \text{ чел.-дн.},$$

$$T_{y.т.п} = 263 \cdot 0,3 \cdot 1,06 \cdot 0,72 \cdot 0,55 = 33 \text{ чел.-дн.},$$

$$T_{y.р.п} = 263 \cdot 0,3 \cdot 1,06 \cdot 0,72 \cdot 0,65 \cdot 0,55 = 21 \text{ чел.-дн.},$$

$$T_{y.в.н} = 263 \cdot 0,1 \cdot 1,06 \cdot 0,72 \cdot 0,55 = 11 \text{ чел.-дн.}$$

Общая трудоемкость разработки программного обеспечения (T_o) определяется суммированием нормативной (скорректированной) трудоемкости программного обеспечения на всех стадиях разработки программного обеспечения по формуле (Д.13):

$$T_o = 11 + 22 + 33 + 21 + 11 = 98 \text{ чел.-дн.}$$

Параметры расчетов по определению нормативной и скорректированной трудоемкости программного обеспечения на всех стадиях разработки и общей трудоемкости разработки программного обеспечения представлены в таблице 5.7.

Таблица 5.7 – Расчет общей трудоемкости разработки ПО

Показатели	Стадии разработки					Итого
	ТЗ	ЭП	ТП	РП	ВН	
1	2	3	4	5	6	7
Общий объем ПО (V_o), кол-во строк <i>LOC</i>	–	–	–	–	–	15388
Общий уточненный объем ПО (V_y), кол-во строк <i>LOC</i>	–	–	–	–	–	6365
Категория сложности разрабатываемого ПО	–	–	–	–	–	3
Нормативная трудоемкость разработки ПО (T_n), чел.-дн.	–	–	–	–	–	263
Коэффициент повышения сложности ПО (K_c)	1,06	1,06	1,06	1,06	1,06	–
Коэффициент, учитывающий новизну ПО (K_n)	0,72	0,72	0,72	0,72	0,72	–

Продолжение таблицы 5.7

1	2	3	4	5	6	7
Коэффициент, учитывающий степень использования стандартных модулей (K_T)	–	–	–	0,65	–	–
Коэффициент, учитывающий средства разработки ПО ($K_{y.p}$)	0,55	0,55	0,55	0,55	0,55	–
Коэффициенты удельных весов трудоемкости стадий разработки ПО ($K_{т.з}, K_{э.п}, K_{т.п}, K_{р.п}, K_{в.н}$)	0,10	0,20	0,30	0,30	0,10	1,0
Распределение скорректированной (с учетом $K_c, K_n, K_t, K_{y.p}$) трудоемкости ПО по стадиям, чел.-дн.	11	22	33	21	11	–
Общая трудоемкость разработки ПО (T_o), чел.-дн.	–	–	–	–	–	98

Таким образом, были вычислены параметры расчетов по определению нормативной и скорректированной трудоемкости разработки программного обеспечения на всех стадиях разработки, а также общая трудоемкость разработки программного обеспечения.

5.3 Расчёт затрат на разработку программного продукта

Суммарные затраты на разработку программного обеспечения ($З_p$) определяются по формуле (Д.14). Параметры расчета производственных затрат на разработку программного обеспечения приведены в таблице 5.8.

Таблица 5.8 – Параметры расчета производственных затрат на разработку ПО

Параметр	Единица измерения	Значение
1	2	3
Тарифная ставка	руб.	250
Доплата за стаж	% (руб.)	19,5 (93,57)
Премия	%	5

Продолжение таблицы 5.8

1	2	3
Доплата по контракту	%	0,5 от оклада
Разряд разработчика	—	12
Тарифный коэффициент	—	2,84
Коэффициент $K_{ув}$	—	1,5
Норматив отчислений на доп. зарплату разработчиков ($H_{доп}$)	%	20
Численность обслуживающего персонала	чел.	1
Средняя годовая ставка арендных платежей ($C_{ар}$) (по результатам мониторинга предложений по аренде помещений)	руб./м ²	13,69
Площадь помещения (S)	м ²	12
Количество ПЭВМ ($Q_{эвм}$)	шт.	1
Затраты на приобретение единицы ПЭВМ	руб.	2000
Стоимость одного кВт-часа электроэнергии ($C_{эл}$)	руб.	0,337
Коэффициент потерь рабочего времени ($K_{пот}$)	—	0,2
Затраты на технологию ($З_{тех}$)	руб.	—
Норматив общепроизводственных затрат ($H_{доп}$)	%	10

Расходы на оплату труда разработчиков с отчислениями ($З_{тр}$) определяются по формуле (Д.15).

Основная заработная плата разработчиков рассчитывается по формуле (Д.16).

Средняя часовая тарифная ставка определяется по формуле (Д.17).

Часовая тарифная ставка определяется путем деления месячной тарифной ставки на установленный при восьмичасовом рабочем дне фонд рабочего времени 168 ч ($F_{мес}$), формула (Д.18).

$$C_{.ч} = \frac{250 \cdot 2,84}{168} = 4,2 \text{ руб.},$$

$$C_{ср.ч} = \frac{4,2 \cdot 1}{1} = 4,2 \text{ руб.},$$

$$ЗП_{осн} = 4,2 \cdot 98 \cdot 1,5 = 617,4 \text{ руб.}$$

Дополнительная заработная плата рассчитывается по формуле (Д.19):

$$ЗП_{\text{доп}} = 617,4 \cdot 20 \div 100\% = 123,5 \text{ руб.}$$

Отчисления от основной и дополнительной заработной платы рассчитываются по формуле (Д.20):

$$\text{ОТЧ}_{\text{с.н}} = \frac{(617,4 + 123,5) \cdot 34\%}{100\%} = 251 \text{ руб.},$$

$$З_{\text{тр}} = 617,4 + 123,5 + 251 = 992 \text{ руб.}$$

Годовые затраты на аренду помещения определяются по формуле (Д.27):

$$З_{\text{ар}} = 13,69 \cdot 12 = 164,28 \text{ руб.}$$

Сумма годовых амортизационных отчислений ($З_{\text{ам}}$) определяется по формуле (Д.28):

$$З_{\text{ам}} = 2000 \cdot (1 + 0,12) \cdot 1 \cdot 0,125 = 280 \text{ руб.}$$

Стоимость электроэнергии, потребляемой за год, определяется по формуле (Д.29).

Действительный годовой фонд времени работы ПЭВМ ($F_{\text{эвм}}$) рассчитывается по формуле (Д.30):

$$F_{\text{эвм}} = (365 - 112) \cdot 8 \cdot 1 \cdot (1 - 0,2) = 1619 \text{ ч.},$$

$$З_{\text{эвм}} = \frac{0,44 \cdot 1619 \cdot 0,337 \cdot 0,9}{1} = 216,1 \text{ руб.}$$

Эп

Затраты на материалы ($З_{\text{эвм}}$), необходимые для обеспечения нормальной работы ПЭВМ, составляют около 1% от балансовой стоимости ЭВМ, и определяются по формуле (Д.31):

$$З_{\text{в.м}} = 2000 \cdot (1 + 0,12) \cdot 0,01 = 22,4 \text{ руб.}$$

Затраты на текущий и профилактический ремонт ($З_{\text{тр}}$) принимаются равными 5% от балансовой стоимости ЭВМ и вычисляются по формуле (Д.32):

$$З_{\text{т.р}} = 2000 \cdot (1 + 0,12) \cdot 0,08 = 179,2 \text{ руб.}$$

Прочие затраты, связанные с эксплуатацией ЭВМ ($Z_{пр}$), состоят из амортизационных отчислений на здания, стоимости услуг сторонних организаций и составляют 5 % от балансовой стоимости и определяются по формуле (Д.33):

$$Z_{пр} = 2000 \cdot (1 + 0,12) \cdot 0,05 = 112 \text{ руб.}$$

Для расчета машинного времени ЭВМ ($t_{эвм}$ в часах), необходимого для разработки и отладки проекта специалистом, следует использовать формулу (Д.34):

$$t_{эвм} = 33 \cdot 8 \cdot 1 = 264 \text{ ч.},$$

Затраты машинного времени ($Z_{м.в}$) определяются по формуле (Д.21):

$$Z_{м.в} = 0,5 \cdot 1 \cdot 264 = 132 \text{ руб.}$$

Стоимость машино-часа определяется по формуле (Д.22):

$$C_{ч} = \frac{280 + 216,1 + 22,4 + 179,2 + 112}{1619} = 0,5 \text{ руб./ч.}$$

Расчет затрат на изготовление эталонного экземпляра ($Z_{эт}$) осуществляется по формуле (Д.35):

$$Z_{эт} = (992 + 132) \cdot 0,05 = 56,2 \text{ руб.}$$

Так же рассчитываются затраты на материалы ($Z_{мат}$) по формуле (Д.36):

$$Z_{мат} = 2000 \cdot (1 + 0,12) \cdot 0,01 = 22,4 \text{ руб.}$$

Общепроизводственные затраты ($Z_{общ.пр}$) рассчитываются по формуле (Д.37):

$$Z_{общ.пр} = \frac{617,4 \cdot 10}{100} = 61,74 \text{ руб.}$$

И наконец, непроизводственные затраты ($Z_{непр}$) рассчитываются по формуле (Д.38):

$$Z_{непр} = \frac{617,4 \cdot 5}{100} = 30,85 \text{ руб.}$$

Итого получаем суммарные затраты на разработку:

$$З_p = 992 + 56,2 + 132 + 22,4 + 61,74 + 30,85 = 1295,2 \text{ руб.}$$

5.4 Расчёт договорной цены и частных экономических эффектов от производства и использования программного продукта

Оптовая цена программного продукта ($Ц_{\text{опт}}$) определяется по., формулам (Д.39) и (Д.40):

$$П_p = \frac{1295,2 \cdot 30}{100} = 388,5 \text{ руб.},$$

$$Ц_{\text{опт}} = 1295,2 + 388,5 = 1683,7 \text{ руб.}$$

Прогнозируемая отпускная цена программного продукта рассчитывается по формуле (Д.41). Налог на добавленную стоимость ($P_{\text{ндс}}$) рассчитывается по формуле (Д.42):

$$P_{\text{ндс}} = \frac{(1295,2 + 388,5) \cdot 20}{100} = 336,6 \text{ руб.},$$

$$Ц_{\text{отп}} = 1295,2 + 388,5 + 336,6 = 2020 \text{ руб.}$$

Годовой экономический эффект от производства нового ПО ($Э_{\text{пр}}$) определяется по разности приведённых затрат на базовый и новый варианты.

Начальная отпускная цена рассчитывается как объем аудитории на цену отпуска и определяется по формуле (Д.43).

Годовой экономический эффект от производства нового программного обеспечения ($Э_{\text{пр}}$) определяется по формуле (Д.44).

При стартовой аудитории в тысячу человек, сумма дохода будет полностью равна цене отпуска. При этом каждая последующая продажа будет приносить доходность с реализации. Ожидаемая цена за единицу лицензии приложения – 4,5 руб.

В качестве модели распространения разрабатываемого ПО была принята модель *free-to-play*, следовательно цена на использование продукта составляет 0 руб. Для получения прибыли планируется использование так называемой рекламы в интернете. Средняя стоимость рекламы за 100 показов составляет 1,5 руб. Прибыль от перехода пользователя по баннеру варьируется в зависимости от региона: стоимость перехода от американского пользователя составляет 0,4 руб, стоимость же перехода в России и Беларуси составляет 0,09 руб. Следовательно, американские пользователи являются приоритетнее, так как разница в прибыли за переход по баннеру составляет примерно 0,31 руб.

Доход от реализации *free-to-play* игр рассчитывается по формуле (Д.45):

$$П_p = 40 \cdot 0,18 \cdot 30 \cdot 12 = 2592 \text{ руб.}$$

Прибыль за реализацию за 12 месяцев составит = 2592 руб.

Таким образом можно рассчитать сроки окупаемости проекта. Прибыль берется с учетом налога на доходы физических лиц (далее НДФЛ), который равняется 13% от прибыли.

$$T_{\text{пр}} = 2020 \div 2255 = 0,89 \text{ лет с учетом НДФЛ.}$$

Рентабельность проекта рассчитывается по формуле (Д.46):

$$P = ((2255 - 2020) \div 2255) \cdot 100 = 10,4\%,$$

Расчеты экономического эффекта определены в таблице 5.10.

Таблица 5.10 – Расчет годового экономического эффекта от производства нового программного продукта

Наименование параметра	Условн. обознач.	Базовый вариант	Новый вариант
Оптовая цена, руб.	C_i	2400	2020
Норматив рентабельности	R_i	0,15	0,15
Себестоимость производства, руб.	$C_{\text{пр}i}$	$2400 / (1+0,15) = 2086$	1683,7
Удельные капитальные вложения, руб.	K_p	2000	
Нормативный коэффициент капитальных вложений	E_n	0,15	0,15
Расчет			
Удельные приведенные затраты на производство ПО, руб.	$Z_{\text{пр}i}$	$2086 + 2000 \cdot 0,15 = 2386$	1683,7
Годовой экономический эффект от производства нового ПП, руб.	$\Delta_{\text{пр}}$	$2386 - 1683,7 = 702$	
Прирост прибыли, руб.	$\Delta\Pi_{\text{пр}}$	$(2020 - 1683,7) - (2400 - 2086) = 22,3$	

Технико-экономические показатели проекта определены в таблице 5.11.

Таблица 5.11 – Технико-экономические показатели проекта

Наименование показателя	Единица измерения	Проектный вариант
1	2	3
Интегральный коэффициент конкурентоспособности	—	2,5
Коэффициент эквивалентности	—	1,673
Коэффициент изменения функциональных возможностей	—	1,35
Коэффициент соответствия нормативам	—	1
Коэффициент цены потребления	—	0,90
Общая трудоемкость разработки ПО	чел.- дн	98
Затраты на оплату труда разработчиков	руб.	992
Затраты машинного времени	руб.	132
Затраты на изготовление эталонного экземпляра	руб.	56,2
Затраты на технологию	руб.	—
Затраты на материалы	руб.	22,4
Общепроизводственные затраты	руб.	61,74
Непроизводственные (коммерческие) затраты	руб.	30,85
Суммарные затраты на разработку ПО (Z_p)	руб.	1295,2
Оптовая цена ПП ($\Pi_{\text{опт}}$) без НДС	руб.	1683,7
Оптовая цена ПП ($\Pi_{\text{опт}}$) с НДС	руб.	2020
Экономический эффект от производства нового ПП	руб.	702

Таким образом, по результатам проведенной оценки, установлено, что реализация проекта обоснована и является экономически целесообразной. Об этом свидетельствуют экономический эффект от производства нового программного продукта ($\mathcal{E}_{\text{пр}} = 702$ руб.). Все данные приведены в итоговой таблице 5.11.

6 ОХРАНА ТРУДА И ТЕХНИКА БЕЗОПАСНОСТИ

6.1 Требования к производственному освещению

Производственное освещение играет существенную роль в обеспечении безопасных, комфортных и производительных условий труда. Неправильное освещение может привести к утомлению глаз, снижению производительности, травмам и профессиональным заболеваниям. В этой работе будут рассмотрены ключевые требования к производственному освещению.

К производственному освещению предъявляются следующие нормативные требования:

1. освещенность рабочих мест должна соответствовать установленным нормативам, нормы освещенности устанавливаются в зависимости от категории зрительных работ, системы освещения, вида работ и других факторов;
2. равномерность освещения: освещение должно быть равномерным и без резких теней, недопустимо наличие слепящей блескости от источников света и отраженных от блестящих поверхностей предметов.
3. спектральный состав света: спектральный состав искусственного света должен быть близок к дневному;
4. размещение и мощность светильников: светильники должны быть правильно размещены и иметь соответствующую мощность.

Производственное освещение можно классифицировать на несколько видов:

- общее освещение: создает общий световой фон в производственном помещении;
- местное освещение: освещает отдельные рабочие места;
- аварийное освещение: обеспечивает освещение в случае отключения рабочего освещения;
- комбинированное освещение: сочетает в себе общее и местное освещение.

При выборе системы освещения необходимо учитывать следующие факторы:

- категорию зрительных работ;
- вид работ;
- размеры и конфигурацию производственного помещения;
- наличие источников естественного света;
- требования к цветопередаче;
- экономичность системы освещения;

Системы производственного освещения должны регулярно контролироваться и обслуживаться. Контроль включает в себя измерение

Добавлено примечание ([П2]): Раздел не может содержать только один подраздел: ни одного или два и больше

Добавлено примечание ([Мо3R2]): Разделы могут состоять из одного или нескольких подразделов. 3.2.1

освещенности, проверку работоспособности светильников и электропроводки. Обслуживание включает чистку светильников, замену ламп и другие работы.

На рисунке 6.1 представлен наглядный визуальный пример освещения на производстве.



Рисунок 6.1 – Наглядный визуальный пример освещения на производстве

Производственное освещение влияет на здоровье и работоспособность. Рациональное производственное освещение оказывает положительное влияние на здоровье и работоспособность работников. Недостаточная освещенность может привести к:

- утомлению глаз;
- снижению производительности труда;
- повышению риска травматизма;
- развитию профессиональных заболеваний;
- слепящая блескость может вызвать
- утомление глаз.
- снижение остроты зрения.
- развитие близорукости.

Неправильный спектральный состав искусственного света может привести к:

- нарушению цветовосприятия;
- снижению работоспособности;
- повышению утомляемости;
- примеры из практики

Учитывая приведенные факторы, в качестве примера, можно описать системы освещения на рабочем месте, соответствующее всем нормативным требованиям.

В металлообрабатывающем цехе, где работники выполняют точные задачи, такие как фрезерование, токарная обработка и шлифование, система освещения состоит из общего люминесцентного освещения и локальных светодиодных светильников на каждом рабочем месте. Уровень освещенности соответствует стандартам, освещение равномерное и без бликов, спектральный состав света близок к дневному.

Также, в качестве примера, можно провести анализ причин несоответствия системы освещения на рабочем месте нормативным требованиям и предложения по ее исправлению.

При осмотре столярного цеха выявлено, что уровень освещенности на некоторых рабочих местах не соответствует стандартам. Также наблюдалась слепящая блескость от окон и блестящих поверхностей станков. Для устранения этих нарушений было предложено установить дополнительные локальные светильники на проблемных рабочих местах и использовать жалюзи на окнах.

В ходе изучения требований к производственному освещению была изучена информация о методах и расчетах освещенности, которая может быть полезна для проектирования и обслуживания систем производственного освещения.

Методы измерения освещенности:

- люксметр: ручной прибор, измеряющий освещенность в люксах (лк);
- фотометр: более точный прибор, который может измерять освещенность в люксах, свечах на квадратную фут (fc) или других единицах.

Формулы для расчета освещенности:

- закон обратных квадратов;
- метод по точкам.

Закон обратных квадратов звучит так: освещенность в точке обратно пропорциональна квадрату расстояния от источника света до точки.

Метод по точкам заключается в расчете освещенности в каждой точке системы освещения с учетом вклада всех источников света.

Программное обеспечение для расчета освещенности:

1. *DIALux* – профессиональное программное обеспечение для проектирования освещения, которое можно использовать для расчета освещенности, моделирования световых сцен и создания отчетов об освещении.

2. *Relux* – программное обеспечение для проектирования освещения с функциями, похожими на *DIALux*.

Производственное освещение является важнейшим фактором, влияющим на здоровье, производительность и безопасность работников. Соблюдение нормативных требований, правильный выбор системы освещения и регулярный контроль, и обслуживание систем освещения являются ключом к созданию безопасных, комфортных условий.

7 ЭНЕРГОСБЕРЕЖЕНИЕ И РЕСУРСОСБЕРЕЖЕНИЕ ПРИ ЭКСПЛУАТАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

7.1 Энергосбережение и ресурсосбережение при внедрении и эксплуатации программного обеспечения

Энерго- и ресурсосбережение представляют собой ключевые направления деятельности, направленные на рациональное использование ресурсов и минимизацию затрат на их потребление. В условиях ограниченности природных ресурсов и роста их стоимости, данные направления становятся все более актуальными.

Энергосбережение определяется как комплекс мер, направленных на эффективное использование топливно-энергетических ресурсов и вовлечение возобновляемых источников энергии. Актуальность энергосбережения обусловлена растущими ценами на традиционные энергоресурсы и истощением их мировых запасов.

Стоит рассмотреть понятие и задачи энергосбережения. Ресурсосбережение включает организационные, экономические, технические, научные и информационные меры, направленные на рациональное использование ресурсов на всех стадиях жизненного цикла изделий [8]. Задачи ресурсосбережения включают:

- сбережение топлива и энергии.
- рациональное использование материальных ресурсов.
- максимальное сохранение природных ресурсов.
- сохранение равновесия между развитием производств и потреблением вторичных материальных ресурсов.
- совершенствование систем управления качеством продукции и услуг.
- экономически эффективное и безопасное использование вторичных ресурсов.

Экономия электроэнергии – важный аспект, влияющий как на производственные процессы, так и на бытовую сферу. Эффективное энергопотребление можно достигнуть через различные меры:

– оптимизация использования компьютеров и периферийных устройств. Использование ПК преимущественно в дневное время позволяет снизить расходы на электроэнергию, учитывая более высокие тарифы в ночное время. Также обновление оборудования и использование устройств с меньшим энергопотреблением помогает сократить затраты.

– уменьшение времени работы неиспользуемых устройств. Выключение компьютеров и периферийных устройств при длительном простое снижает энергопотребление;

Добавлено примечание ([П4]): Раздел не может содержать только один подраздел: ни одного или два и больше

Добавлено примечание ([Mo5R4]): Разделы могут состоять из одного или нескольких подразделов. 3.2.1

– энергосберегающие технологии в ПО.

Применение оптимизированных алгоритмов и структур данных уменьшает нагрузку на вычислительные ресурсы и снижает потребление электроэнергии.

Дальше следует рассмотреть принципы ресурсосбережения при разработке ПО [9].

Для эффективного ресурсосбережения в процессе разработки программного обеспечения необходимо:

– снижение объемов используемых ресурсов. Оптимизация текстур, уменьшение количества полигонов в трехмерных моделях, исключение дублирования идентичных ресурсов;

– оптимизация графического интерфейса. Использование выдержанной, неконтрастной цветовой гаммы и плавных анимаций уменьшает нагрузку на глаза пользователей и снижает энергопотребление.

Рассмотрим пример экономии электроэнергии при использовании мобильных версий приложений. Среднее энергопотребление настольного ПК составляет около 350 Вт·ч, тогда как смартфон или планшет потребляют около 15 Вт·ч. Использование мобильных устройств может значительно снизить энергопотребление.

Формула для расчета экономии электроэнергии:

$$\mathcal{E} = (T_{\text{ПК}} - T_{\text{м}}) \cdot P \cdot C_{\text{эл}} \cdot K_{\text{и}}, \quad (7.1)$$

где $T_{\text{ПК}}$ – среднее энергопотребление персонального настольного компьютера или иного подобного устройства, кВт·ч; $T_{\text{м}}$ – среднее энергопотребление мобильного устройства, кВт·ч; P – время работы используемого устройства, ч; $C_{\text{эл}}$ – стоимость одного кВт·ч электроэнергии, руб; $K_{\text{и}}$ – коэффициент использования.

В среднем электропотребление при стандартной загрузке среднестатистического персонального компьютера составляет 350 Вт·ч. Энергопотребление смартфона или планшета составляет 15 Вт·ч. По состоянию на 2 мая 2022 года стоимость 1 кВт·ч. электроэнергии в Республике Беларусь для физических лиц равен 0,3551 рубля [12]. Коэффициент использования принят за 0,6. Из выше сказанного, стоимость сэкономленной электроэнергии за один месяц при игре 4 часа каждый день составит:

$$\mathcal{E} = (0,35 - 0,015) \cdot 120 \cdot 0,3551 \cdot 0,6 = 8,56 \text{ руб.}$$

Экономия электроэнергии за нужный период времени можно рассчитать по формуле:

$$\mathcal{E}_{\text{пер}} = \mathcal{E} \cdot n, \quad (7.2)$$

где \mathcal{E} – сэкономленная электроэнергия за месяц, руб.; n – период использования автоматизированной системы, месяцев.

$$\mathcal{E}_{\text{пер}} = 8,56 \cdot 12 = 102,8 \text{ руб.}$$

Данный расчет показывает, что при использовании кроссплатформенности разработанного игрового приложения можно сэкономить на электроэнергии до 8,5 рублей в месяц, а за год 102,8 рублей.

Энерго- и ресурсосбережение при разработке и эксплуатации программного обеспечения являются важными аспектами повышения экономической и экологической эффективности. Оптимизация кода, использование энергосберегающего оборудования и рациональное управление ресурсами позволяют снизить затраты и улучшить устойчивость информационных систем. Эти меры способствуют не только экономии ресурсов, но и сохранению окружающей среды [10].

ЗАКЛЮЧЕНИЕ

Результатом дипломной работы является игровое 2D приложение «*Farmer's Valley*», представляющего собой аркадный фермерский симулятор в *top-down* проекции на платформе *Unity*. В процессе выполнения работы были достигнуты следующие результаты и решены поставленные задачи:

- проведен анализ современных методов разработки игровых приложений и выбраны наиболее подходящие средства и технологии для создания приложения;
- проведен аналитический обзор игр в таком жанре;
- разработан графический интерфейс пользователя, реализующий взаимодействие пользователя с программой;
- разработана эффективная и гибкая архитектура приложения, определены его компоненты и взаимодействие между ними, что обеспечило удобство разработки и возможность дальнейшего расширения приложения;
- проведено тестирование приложения, которое подтвердило его стабильность, надежность и эффективность, обнаруженные ошибки были исправлены, а функционал приложения проверен на соответствие заявленным требованиям;

Функционал игрового приложения реализован средствами платформы *Unity*.

Разработанное игровое приложение «*Farmer's Valley*» представляет собой продукт, объединяющий в себе передовые технологии и увлекательный игровой процесс. Его отличительные особенности включают высококачественную графику, интересный игровой процесс и возможность социального взаимодействия с *NPC*.

Проведенное тестирование показало, что игровое приложение выполняет свои функции, игровые механики функционируют должным образом. Кроме того, система является масштабируемой.

Как рекомендация по использованию результатов дипломной работы, предлагается дальнейшее развитие и улучшение приложения путем добавления новых функций, механик и контента.

Список использованных источников

1. Хокинг, Д. В. Unity в действии. Мультиплатформенная разработка на C# / Д. В. Хокинг – СПб. : Питер, 2019. – 215 с.
2. Гибсон, Б. Д. Unity и C#. Геймдев от идеи до реализации / Б. Д. Гибсон – СПб. : Питер, 2019. – 228 с.
3. Куксон, А. А. Разработка игр на Unity / А. А. Куксон – М. : Бомбора, 2019. – 298 с.
4. Арам Куксон, Д. Д. Разработка игр на UnrealEngine 4. 4-е изд. / А. Л. Куксон. – СПб. : Питер, 2017. – 917 с.
5. Фримен, Э. К. Паттерны проектирования / Э. К. Фримен – СПб. : Питер, 2011. – 176 с.
6. Feature driven development. – Электронные данные. – 2017. – Режим доступа: <https://intellect.icu/feature-driven-development-5180>. – Дата доступа: 16.05.2022.
7. Различные виды тестирования ПО. – Электронные данные. – Режим доступа: <https://www.atlassian.com/ru/continuous-delivery/software-testing/types-of-software-testing>. – Дата доступа: 16.05.2022.
8. Андрижиевский, А. А. Энергосбережение и энергетический менеджмент : учеб. пособие для студ. / А. А. Андрижиевский, В. И. Володин. – 2-е изд., испр. – Мн. : Вышэйшая школа, 2017. – 294 с.
9. Ляшенко, Д. Д. Ресурсосбережение: инновации в энергосбережении / Д. Д. Ляшенко, В. И. Мартынович // Международное научное обозрение проблем и перспектив современной науки и образования Сборник статей по материалам XXX Международной научно-практической конференции, 2018. – С.32.
10. Организация энергосбережения (энергоменеджмент). Решения ЗСМК-НКМКНТМК-ЕВРАЗ : учебное пособие / под ред. В. В. Кондратьева – М.: ИНФРАМ, 2017. – 108 с.

ПРИЛОЖЕНИЕ А (обязательное)

Программный код приложения

Листинг класса ChatContextOptimizer:

```
using System;
using SharpToken;
using UnityEngine;

namespace Assets.Scripts.GameSO.Chat
{
    public class ChatContextOptimizer
    {
        string _encodingModelId;
        int _restContextSize;
        GptEncoding _gptEncoding;
        public ChatContextOptimizer(string encodingModelId, int contextSize)
        {
            _encodingModelId = encodingModelId;
            _restContextSize = contextSize;
        }

        public bool TryInitialize()
        {
            try
            {
                _gptEncoding = GptEncoding.GetEncoding(_encodingModelId);

                return true;
            }
            catch (Exception ex)
            {
                Debug.LogError(ex);

                return false;
            }
        }

        public bool IsNeedReset(string text)
        {
            var encodedCount = _gptEncoding.CountTokens(text);

            Debug.Log($"Current context size: {encodedCount}");

            return encodedCount >= _restContextSize;
        }
    }
}
```

Листинг класса IMessagePanelFactory:

```

using UnityEngine;

namespace Scripts.ChatAssistant
{
    public interface IMessagePanelFactory
    {
        MessagePanel CreateMessagePanel(Transform parent);
    }
}

```

Листинг класса MessagePanelFactory:

```

using UnityEngine;
using Zenject;

namespace Scripts.ChatAssistant
{
    public class MessagePanelFactory : IMessagePanelFactory
    {
        MessagePanel _messagePanelTemplate;
        IInstantiator _instantiator;
        public MessagePanelFactory(MessagePanel messagePanelTemplate, IInstantiator instantiator)
        {
            _messagePanelTemplate = messagePanelTemplate;
            _instantiator = instantiator;
        }
        public MessagePanel CreateMessagePanel(Transform parent)
        {
            MessagePanel panel = _instantiator.InstantiatePrefabForComponent<MessagePanel>
                (_messagePanelTemplate, parent);
            return panel;
        }
    }
}

```

Листинг класса MessagePanelsFactories:

```

using System.Collections.Generic;
using UnityEngine;

namespace Scripts.ChatAssistant
{
    public enum MessagePanelType
    {
        Player,
        Assistant
    }
    public class MessagePanelsFactories
    {
        Dictionary<MessagePanelType, IMessagePanelFactory> _messagePanels;
        public MessagePanelsFactories()
        {
            _messagePanels = new Dictionary<MessagePanelType, IMessagePanelFactory>();
        }
    }
}

```

```

    }

    public void Add(MessagePanelType type, IMessagePanelFactory factory)
    {
        if(_messagePanels.TryAdd(type, factory) == false)
        {
            Debug.LogError($"Failed to add {typeof(IMessagePanelFactory)} as {type}");
        }
    }

    public IMessagePanelFactory GetByType(MessagePanelType type)
    {
        if (_messagePanels.TryGetValue(type, out IMessagePanelFactory messagePanelFactory))
        {
            return messagePanelFactory;
        }
        else
            return default;
    }
}

```

Листинг класса ChatPanel:

```

using PimDeWitte.UnityMainThreadDispatcher;
using Scripts.SO.Chat;
using System;
using System.Threading.Tasks;
using TMPPro;
using UnityEngine;
using UnityEngine.UI;
using Zenject;
using Assets.Scripts.GameSO.Chat;

namespace Scripts.ChatAssistant
{
    public class ChatPanel : MonoBehaviour
    {
        [SerializeField] private string Name;
        Transform _container;
        MessagePanelsFactories _messagePanelsFactories;

        ChatService _chatService;
        ChatContextOptimizer _chatContextOptimizer;

        [SerializeField] private TMP_InputField _inputField;
        [SerializeField] private Button _enterButton;
        [SerializeField] private Button _resetButton;
        [SerializeField] private GameObject _chatPanel;

        InputService _inputService;
        ChatSODataBase _chatSOData;

        [Inject]
    }
}

```



```

public void Construct(MessagePanelsFactories messagePanelsFactories,
    ChatSODataBase chatSODataBase,
    InputService inputService)
{
    _chatSOData = chatSODataBase;
    _messagePanelsFactories = messagePanelsFactories;
    _inputService = inputService;
}

private void Start()
{
    ChatSO chatSO = _chatSOData.GetItemByName(Name);

    if(chatSO == null)
    {
        Debug.LogError($"Cannot get chat SO Object by name {Name}");
        Destroy(gameObject);
    }
    else
    {
        _chatService = new ChatService(chatSO);

        _chatContextOptimizer = new ChatContextOptimizer(chatSO.EncodingModelID,
            chatSO.ResetContextSize);

        if(_chatContextOptimizer.TryInitialize() == false)
        {
            Debug.LogError($"Cannot initialize chat context optimizer by name {Name}");
            Destroy(gameObject);
        }
    }
    _container = gameObject.transform;
    _resetButton.interactable = false;
}

public async Task CreateMessagePanelAsync(IMessagePanelFactory messagePanelFactory, string text)
{
    MessagePanel messagePanel = messagePanelFactory.CreateMessagePanel(_container);

    ResizeMessagePanel(messagePanel, text);

    await messagePanel.WriteTextAsync(text);
}
public void CreateMessagePanel(IMessagePanelFactory messagePanelFactory, string text)
{
    MessagePanel messagePanel = messagePanelFactory.CreateMessagePanel(_container);

    ResizeMessagePanel(messagePanel, text);

    messagePanel.WriteText(text);
}

void ResizeMessagePanel(MessagePanel messagePanel, string text)

```

```

{
    messagePanel.TextField.text = text;

    messagePanel.TextField.ForceMeshUpdate();

    var size = messagePanel.TextField.GetPreferredValues();

    float panelHeight = size.y;

    messagePanel.Size = new Vector2(messagePanel.Size.x, panelHeight + messagePanel.Size.y);
}

public async void OnEnterText()
{
    if (_inputField.text.Length > 0)
    {
        IMessagePanelFactory playerPanelFactory =
            _messagePanelsFactories.GetByType(MessagePanelType.Player);

        CreateMessagePanel(playerPanelFactory, _inputField.text);

        _enterButton.interactable = false;

        _chatService.AddUserInput(_inputField.text);

        _inputField.text = String.Empty;
        await ConstructResponse();
    }
}

public void OnReset()
{
    _chatService.ClearChat();
    ClearMessagePanels();
    _enterButton.interactable = true;
    _resetButton.interactable = false;
}

async Task ConstructResponse()
{
    await Task.Run(() =>
    {
        UnityMainThreadDispatcher.Instance().Enqueue(async () =>
        {
            string text = "";
            IMessagePanelFactory assistantPanelFactory =
                _messagePanelsFactories.GetByType(MessagePanelType.Assistant);
            try
            {
                text = await _chatService.GetResponseAsync();
            }
            catch (Exception ex)
            {
                text += ex.Message;
            }
            finally
            {

```

```

        await CreateMessagePanelAsync(assistantPanelFactory, text)
        .ContinueWith((t) =>
        {
            UnityMainThreadDispatcher.Instance().Enqueue(() =>
            {
                _enterButton.interactable = true;
                Debug.Log("End construct response!");

                if(_chatContextOptimizer.IsNeedReset(_chatService.GetContextText()))
                {
                    _enterButton.interactable = false;
                    _resetButton.interactable = true;
                }
            });
        });
    });
}

public void OnExit()
{
    _inputService.UnlockGamePlayControls();
    _chatPanel.SetActive(false);
}

void ClearMessagePanels()
{
    foreach(Transform messagePanel in _container)
    {
        Destroy(messagePanel.gameObject);
    }
}
}
}

```

Листинг класса MessagePanel:

```

using PimDeWitte.UnityMainThreadDispatcher;
using Scripts.SO.Chat;
using System;
using System.Threading.Tasks;
using TMPro;
using UnityEngine;
using Zenject;

namespace Scripts.ChatAssistant
{
    public class MessagePanel : MonoBehaviour
    {
        [SerializeField] private RectTransform _rectTransform;
        [SerializeField] private RectTransform _childRect;
        [SerializeField] private RectTransform _textRect;
        [SerializeField] private TextMeshProUGUI _textField;
    }
}

```

```

public TextMeshProUGUI TextField => _textField;
public float FontSize => TextField.fontSize;
public Vector2 Size
{
    get
    {
        return _rectTransform.rect.size;
    }
    set
    {
        _rectTransform.sizeDelta = new Vector2(value.x, value.y);
        _childRect.sizeDelta = new Vector2(_childRect.rect.size.x, value.y);
        _textRect.sizeDelta = new Vector2(_textRect.rect.size.x, value.y);
    }
}

MessageSO _messageSO;

[Inject]
public void Construct(MessageSO messageSO)
{
    _messageSO = messageSO;
}

public async Task WriteTextAsync(string text)
{
    await Task.Run(() =>
    {
        UnityMainThreadDispatcher.Instance().Enqueue(async () =>
        {
            _textField.text = String.Empty;
            foreach (var ch in text)
            {
                _textField.text += ch;
                await Task.Delay(_messageSO.WriteSpeed);
            }
        });
    });
}

public void WriteText(string text)
{
    _textField.text = String.Empty;
    foreach (var ch in text)
    {
        _textField.text += ch;
    }
}
}

```

Листинг класса ChatService:

```

using Scripts.SO.Chat;
using OpenAI_API;

```

```

using OpenAI_API.Chat;
using System.Threading.Tasks;
using System.Text;

namespace Scripts.ChatAssistant
{

    public class ChatService
    {
        ChatSO _chatSO;
        OpenAIAPI _api;
        APIAuthentication _authentication;

        Conversation _chat;

        StringBuilder _messagesTextBuilder;

        public ChatService(
            ChatSO chatSO)
        {
            _chatSO = chatSO;
            _authentication = new APIAuthentication(_chatSO.APIKey);

            _api = new OpenAIAPI(_authentication);
            _api.ApiUrlFormat = _chatSO.Url;
            _chat = _api.Chat.CreateConversation();
            _chat.Model.ModelID = _chatSO.ModelId;
            _chat.RequestParameters.Temperature = _chatSO.Temperature;
            _chat.RequestParameters.MaxTokens = _chatSO.MaxTokens;

            _messagesTextBuilder = new StringBuilder();

            AddSystemInput(_chatSO.SystemMessage);
        }

        public void AddUserInput(string text)
        {
            _chat.AppendUserInput(text);
        }

        void AddSystemInput(string text)
        {
            _chat.AppendSystemMessage(text);
        }

        public async Task<string> GetResponseAsync()
        {
            return await _chat.GetResponseFromChatbotAsync();
        }

        public string GetContextText()
        {
            _messagesTextBuilder.Clear();

            foreach (var message in _chat.Messages)
            {
                _messagesTextBuilder.Append(message.TextContent);
            }
        }
    }
}

```

```

        return _messagesTextBuilder.ToString();
    }
    public void ClearChat()
    {
        var messages = _chat.Messages;
        for (int i = 0; i < messages.Count; i++)
        {
            if (messages[i].Role.ToString() == "user" ||
                messages[i].Role.ToString() == "assistant")
            {
                messages.Remove(messages[i]);
            }
        }
    }
}

```

Листинг класса DayNightCycleHandler:

```

using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Rendering.Universal;
using UnityEngine.UIElements;

#if UNITY_EDITOR
using UnityEditor;
using UnityEditor.UIElements;
#endif

namespace Scripts.DayNight
{
    public class DayNightCycleHandler : MonoBehaviour
    {
        private static DayNightCycleHandler s_instance;
        public Transform LightsRoot;

        [Header("Day Light")]
        public Light2D DayLight;
        public Gradient DayLightGradient;

        [Header("Night Light")]
        public Light2D NightLight;
        public Gradient NightLightGradient;

        [Header("Ambient Light")]
        public Light2D AmbientLight;
        public Gradient AmbientLightGradient;

        [Header("RimLights")]
        public Light2D SunRimLight;
        public Gradient SunRimLightGradient;
        public Light2D MoonRimLight;
        public Gradient MoonRimLightGradient;
    }
}

```

```

[Tooltip("The angle 0 = upward, going clockwise to 1 along the day")]
public AnimationCurve ShadowAngle;
[Tooltip("The scale of the normal shadow length (0 to 1) along the day")]
public AnimationCurve ShadowLength;

public List<Shadow> m_Shadows { get; set; } = new();
private List<FarmerLightInterpolator> m_LightBlenders = new();

private void Awake()
{
    if(s_instance == null)
    {
        s_instance = this;
    }
}

public void Tick(float ratio)
{
    UpdateLight(ratio);
}

public void UpdateLight(float ratio)
{
    DayLight.color = DayLightGradient.Evaluate(ratio);
    NightLight.color = NightLightGradient.Evaluate(ratio);

#if UNITY_EDITOR
    //the test between the define will only happen in editor and not in build, as it is assumed those will be set
    //in build. But in editor we may want to test without those set. (those were added later in development so
    //some test scene didn't have those set and we wanted to be able to still test those)
    if (AmbientLight != null)
#endif
        AmbientLight.color = AmbientLightGradient.Evaluate(ratio);

#if UNITY_EDITOR
    if (SunRimLight != null)
#endif
        SunRimLight.color = SunRimLightGradient.Evaluate(ratio);

#if UNITY_EDITOR
    if (MoonRimLight != null)
#endif
        MoonRimLight.color = MoonRimLightGradient.Evaluate(ratio);

    LightsRoot.rotation = Quaternion.Euler(0, 0, 360.0f * ratio);

    UpdateShadow(ratio);
}

void UpdateShadow(float ratio)
{
    {
        var currentShadowAngle = ShadowAngle.Evaluate(ratio);
        var currentShadowLength = ShadowLength.Evaluate(ratio);

        var opposedAngle = currentShadowAngle + 0.5f;
        while (currentShadowAngle > 1.0f)
            currentShadowAngle -= 1.0f;
    }
}

```

```

foreach (var shadow in m_Shadows)
{
    var t = shadow.transform;
    //use 1.0-angle so that the angle goes clo
    t.eulerAngles = new Vector3(0, 0, currentShadowAngle * 360.0f);
    t.localScale = new Vector3(1, 1f * shadow.BaseLength * currentShadowLength, 1);
}

foreach (var handler in m_LightBlenders)
{
    handler.SetRatio(ratio);
}
}

public static void RegisterShadow(Shadow shadow)
{
#ifdef UNITY_EDITOR
    //in the editor when not running, we find the instance manually. Less efficient but not a problem at edit time
    //allow to be able to previz shadow in editor
    if (!Application.isPlaying)
    {
        var instance = GameObject.FindObjectOfType<DayNightCycleHandler>();
        if (instance != null)
        {
            instance.m_Shadows.Add(shadow);
        }
    }
    else
    {
#endif
        s_instance.m_Shadows.Add(shadow);
#ifdef UNITY_EDITOR
    }
#endif
}

public static void UnregisterShadow(Shadow shadow)
{
#ifdef UNITY_EDITOR
    //in the editor when not running, we find the instance manually. Less efficient but not a problem at edit time
    //allow to be able to previz shadow in editor
    if (!Application.isPlaying)
    {
        var instance = GameObject.FindObjectOfType<DayNightCycleHandler>();
        if (instance != null)
        {
            instance.m_Shadows.Remove(shadow);
        }
    }
    else
    {
#endif
        if (s_instance != null)
            s_instance.m_Shadows.Remove(shadow);
#ifdef UNITY_EDITOR
    }
#endif
}

```



```

    }

    public static void RegisterLightBlender(FarmerLightInterpolator interpolator)
    {
#ifdef UNITY_EDITOR
        //in the editor when not running, we find the instance manually. Less efficient but not a problem at edit time
        //allow to be able to previz shadow in editor
        if (!Application.isPlaying)
        {
            var instance = FindObjectOfType<DayNightCycleHandler>();
            if (instance != null)
            {
                instance.m_LightBlenders.Add(interpolator);
            }
        }
        else
        {
#endif
            s_instance.m_LightBlenders.Add(interpolator);
#ifdef UNITY_EDITOR
        }
#endif
    }

    public static void UnregisterLightBlender(FarmerLightInterpolator interpolator)
    {
#ifdef UNITY_EDITOR
        //in the editor when not running, we find the instance manually. Less efficient but not a problem at edit time
        //allow to be able to previz shadow in editor
        if (!Application.isPlaying)
        {
            var instance = FindObjectOfType<DayNightCycleHandler>();
            if (instance != null)
            {
                instance.m_LightBlenders.Remove(interpolator);
            }
        }
        else
        {
#endif
            if (s_instance != null)
                s_instance.m_LightBlenders.Remove(interpolator);
#ifdef UNITY_EDITOR
        }
#endif
    }

    [System.Serializable]
    public struct DayCycleHandlerSaveData
    {
        public float TimeOfTheDay;
    }

#ifdef UNITY_EDITOR
    // Wrapping a custom editor between UNITY_EDITOR define check allow to keep it in the same
    // file as this part will be stripped when building for standalone (where Editor class doesn't exist).

```

```

// Don't forget to also wrap the UnityEditor using at the top of the file between those define check too.

// Show a slider that allow to test a specific time to help define colors.
[CustomEditor(typeof(DayNightCycleHandler))]
class DayCycleEditor : Editor
{
    private DayNightCycleHandler m_Target;

    public override VisualElement CreateInspectorGUI()
    {
        m_Target = target as DayNightCycleHandler;

        var root = new VisualElement();

        InspectorElement.FillDefaultInspector(root, serializedObject, this);

        var slider = new Slider(0.0f, 1.0f);
        slider.label = "Test time 0:00";
        slider.RegisterValueChangedCallback(evt =>
        {
            m_Target.UpdateLight(evt.newValue);

            slider.label = $"Test Time {DayNightService.GetTimeAsString(evt.newValue)} ({evt.newValue:F2})";
            SceneView.RepaintAll();
        });

        //registering click event, it's very catch all but not way to do a change check for control change
        root.RegisterCallback<ClickEvent>(evt =>
        {
            m_Target.UpdateLight(slider.value);
            SceneView.RepaintAll();
        });

        root.Add(slider);

        return root;
    }
}
#endif
}

```

Листинг класса DayNightEventHandler:

```

using UnityEngine;
using UnityEngine.Events;
using UnityEngine.UIElements;
#if UNITY_EDITOR
using UnityEditor.UIElements;
using UnityEditor;
#endif

namespace Scripts.DayNight
{
    [DefaultExecutionOrder(999)]

```

```

public class DayNightEventHandler : MonoBehaviour
{
    [System.Serializable]
    public class DayEvent
    {
        public float StartTime = 0.0f;
        public float EndTime = 1.0f;

        public UnityEvent OnEvents;
        public UnityEvent OffEvent;

        public bool IsInRange(float t)
        {
            return t >= StartTime && t <= EndTime;
        }
    }

    public DayEvent[] Events;

    private void Start()
    {
        DayNightService.RegisterEventHandler(this);
    }

    private void OnDisable()
    {
        DayNightService.RemoveEventHandler(this);
    }
    public void OnOnLight()
    {
    }

}

#if UNITY_EDITOR
[CustomPropertyDrawer(typeof(DayNightEventHandler.DayEvent))]
public class DayEvent : PropertyDrawer
{
    public override VisualElement CreatePropertyGUI(SerializedProperty property)
    {
        {
            var dayHandler = GameObject.FindObjectOfType<DayNightCycleHandler>();

            // Create property container element.
            var container = new VisualElement();

            if (dayHandler != null)
            {
                var minProperty = property.FindPropertyRelative(nameof(DayNightEventHandler.DayEvent.StartTime));
                var maxProperty = property.FindPropertyRelative(nameof(DayNightEventHandler.DayEvent.EndTime));

                var slider = new MinMaxSlider(
                    $"Day range {DayNightService.GetTimeAsString(minProperty.floatValue)} - {DayNightService.GetTimeAsString(maxProperty.floatValue)}",
                    minProperty.floatValue, maxProperty.floatValue, 0.0f, 1.0f);

                slider.RegisterValueChangedCallback(evt =>
                {

```

```

        minProperty.floatValue = evt.newValue.x;
        maxProperty.floatValue = evt.newValue.y;

        property.serializedObject.ApplyModifiedProperties();

        slider.label =
            $"Day range {DayNightService.GetTimeAsString(minProperty.floatValue)} -
            {DayNightService.GetTimeAsString(maxProperty.floatValue)}";
    });

    var evtOnProperty = property.FindPropertyRelative(nameof(DayNightEventHandler.DayEvent.OnEvents));
    var evtOffProperty = property.FindPropertyRelative(nameof(DayNightEventHandler.DayEvent.OffEvent));

    container.Add(slider);

    container.Add(new PropertyField(evtOnProperty, "On Event"));
    container.Add(new PropertyField(evtOffProperty, "Off Event"));
    }
    else
    {
        container.Add(new Label("There is no DayCycleHanlder in the scene and it is needed"));
    }

    return container;
}
}
#endif
}

```

Листинг класса DayNightService:

```

using System.Collections.Generic;
using TMPro;
using UnityEngine;

namespace Scripts.DayNight
{
    [DefaultExecutionOrder(-10)]
    public class DayNightService : MonoBehaviour
    {
        private static DayNightService s_instance;
        [SerializeField] private TextMeshProUGUI _timerText;
        private DayNightCycleHandler _dayCycleHandler;
        [Header("Time settings")]
        [Min(1.0f)]
        public float DayDurationInSeconds;

        public float StartingTime = 0.0f;
        // Will return the ratio of time for the current day between 0 (00:00) and 1 (23:59).
        public float CurrentDayRatio => m_CurrentTimeOfDay / DayDurationInSeconds;
        private float m_CurrentTimeOfDay;
        public float CurrentTimeOfDay;
        private bool m_IsTicking;
    }
}

```

```

private List<DayNightEventHandler> m_EventHandlers = new();

private void Awake()
{
    if(s_instance == null)
    {
        s_instance = this;
    }
}

private void Start()
{
    m_IsTicking = true;
    _dayCycleHandler = GetComponent<DayNightCycleHandler>();
}

private void Update()
{
    _timerText.text = GetTimeAsString(CurrentDayRatio);

    CurrentTimeOfTheDay = m_CurrentTimeOfTheDay;
    if (m_IsTicking)
    {
        float previousRatio = CurrentDayRatio;
        m_CurrentTimeOfTheDay += Time.deltaTime;

        while (m_CurrentTimeOfTheDay > DayDurationInSeconds)
            m_CurrentTimeOfTheDay -= DayDurationInSeconds;

        foreach (var handler in m_EventHandlers)
        {
            foreach (var evt in handler.Events)
            {
                bool prev = evt.IsInRange(previousRatio);
                bool current = evt.IsInRange(CurrentDayRatio);

                if (prev && !current)
                {
                    evt.OffEvent.Invoke();
                }
                else if (!prev && current)
                {
                    evt.OnEvents.Invoke();
                }
            }
        }

        if (_dayCycleHandler != null)
            _dayCycleHandler.Tick(CurrentDayRatio);
    }
}

public static int GetHourFromRatio(float ratio)
{
    var time = ratio * 24.0f;
    var hour = Mathf.FloorToInt(time);
}

```

```

        return hour;
    }

    public static int GetMinuteFromRatio(float ratio)
    {
        var time = ratio * 24.0f;
        var minute = Mathf.FloorToInt((time - Mathf.FloorToInt(time)) * 60.0f);

        return minute;
    }

    /// <summary>
    /// Return in the format "xx:xx" the given ration (between 0 and 1) of time
    /// </summary>
    /// <param name="ratio"></param>
    /// <returns></returns>
    public static string GetTimeAsString(float ratio)
    {
        var hour = GetHourFromRatio(ratio);
        var minute = GetMinuteFromRatio(ratio);

        return $" {hour} : {minute:00} ";
    }

    /// <summary>
    /// Will return the current time as a string in format of "xx:xx"
    /// </summary>
    /// <returns></returns>
    public string CurrentTimeAsString()
    {
        return GetTimeAsString(CurrentDayRatio);
    }

    public static void RegisterEventHandler(DayNightEventHandler handler)
    {
        foreach (var evt in handler.Events)
        {
            if (evt.IsInRange(s_instance.CurrentDayRatio))
            {
                evt.OnEvents.Invoke();
            }
            else
            {
                evt.OffEvent.Invoke();
            }
        }
    }

    s_instance.m_EventHandlers.Add(handler);
}

public static void RemoveEventHandler(DayNightEventHandler handler)
{
    s_instance.m_EventHandlers.Remove(handler);
}
}
}

```

Листинг класса FarmerLightInterpolator:

```

using System;
using UnityEngine;
using UnityEngine.Rendering.Universal;
using UnityEngine.UIElements;
using System.Linq;

#if UNITY_EDITOR
using UnityEditor;
using UnityEditor.UIElements;
#endif

namespace Scripts.DayNight
{
    [DefaultExecutionOrder(999)]
    [ExecuteInEditMode]
    public class FarmerLightInterpolator : MonoBehaviour
    {
        [Serializable]
        public class LightFrame
        {
            public Light2D ReferenceLight;
            public float NormalizedTime;
        }

        [Tooltip("The light of which the shape will be changed according to the defined frames")]
        public Light2D TargetLight;
        public LightFrame[] LightFrames;

        private void OnEnable()
        {
            DayNightCycleHandler.RegisterLightBlender(this);
        }

        private void OnDisable()
        {
            DayNightCycleHandler.UnregisterLightBlender(this);
        }

        public void SetRatio(float t)
        {
            if (LightFrames.Length == 0)
                return;

            int startFrame = 0;

            while (startFrame < LightFrames.Length - 1 && LightFrames[startFrame + 1].NormalizedTime < t)
            {
                startFrame += 1;
            }

            if (startFrame == LightFrames.Length - 1)
            {
                //the last frame is the "start frame" so there is no frame to interpolate TO, so we just use the last settings
                Interpolate(LightFrames[startFrame].ReferenceLight, LightFrames[startFrame].ReferenceLight, 0.0f);
            }
        }
    }
}

```

```

    }
    else
    {
        float frameLength = LightFrames[startFrame + 1].NormalizedTime -
LightFrames[startFrame].NormalizedTime;
        float frameValue = t - LightFrames[startFrame].NormalizedTime;
        float normalizedFrame = frameValue / frameLength;

        Interpolate(LightFrames[startFrame].ReferenceLight, LightFrames[startFrame + 1].ReferenceLight,
normalizedFrame);
    }
}

void Interpolate(Light2D start, Light2D end, float t)
{
    TargetLight.color = Color.Lerp(start.color, end.color, t);
    TargetLight.intensity = Mathf.Lerp(start.intensity, end.intensity, t);

    var startPath = start.shapePath;
    var endPath = end.shapePath;

    var newPath = new Vector3[startPath.Length];

    for (int i = 0; i < startPath.Length; ++i)
    {
        newPath[i] = Vector3.Lerp(startPath[i], endPath[i], t);
    }

    TargetLight.SetShapePath(newPath);
}

public void DisableAllLights()
{
    foreach (var frame in LightFrames)
    {
        frame.ReferenceLight?.gameObject.SetActive(false);
    }
}
}

#if UNITY_EDITOR
[CustomEditor(typeof(FarmerLightInterpolator))]
public class LightInterpolatorEditor : Editor
{
    private FarmerLightInterpolator m_Interpolator;
    private DayNightCycleHandler m_DayCycleHandler;

    private Slider m_PreviewSlider;

    private void OnEnable()
    {
        m_Interpolator = target as FarmerLightInterpolator;
        m_Interpolator.DisableAllLights();
        m_Interpolator.SetRatio(0);

        m_DayCycleHandler = GameObject.FindObjectOfType<DayNightCycleHandler>();
        if (m_DayCycleHandler != null)
        {

```



```

        m_DayCycleHandler.UpdateLight(0.0f);
    }

    EditorApplication.update += SceneUpdate;
}

private void OnDisable()
{
    EditorApplication.update -= SceneUpdate;
}

void SceneUpdate()
{
    if (m_PreviewSlider == null ||
        m_Interpolator == null ||
        m_DayCycleHandler == null)
        return;

    m_Interpolator.SetRatio(m_PreviewSlider.value);
    m_DayCycleHandler.UpdateLight(m_PreviewSlider.value);

    SceneView.RepaintAll();
}

public override VisualElement CreateInspectorGUI()
{
    var root = new VisualElement();

    CustomUI(root);

    m_PreviewSlider = new Slider("Preview time 00:00 (0)", 0.0f, 1.0f);
    m_PreviewSlider.RegisterValueChangedCallback(evt =>
    {
        m_Interpolator.SetRatio(m_PreviewSlider.value);
        if (m_DayCycleHandler != null)
        {
            m_PreviewSlider.label = $"Preview time {DayNightService.GetTimeAsString(m_PreviewSlider.value)}
({m_PreviewSlider.value})";
            m_DayCycleHandler.UpdateLight(m_PreviewSlider.value);
        }
    });

    root.Add(m_PreviewSlider);

    return root;
}

void CustomUI(VisualElement root)
{
    var targetLight = serializedObject.FindProperty(nameof(FarmerLightInterpolator.TargetLight));
    var frames = serializedObject.FindProperty(nameof(FarmerLightInterpolator.LightFrames));

    var list = new ListView();

    list.showFoldoutHeader = false;
    list.showBoundCollectionSize = false;
    list.reorderable = true;

```

```

list.virtualizationMethod = CollectionVirtualizationMethod.DynamicHeight;
list.reorderMode = ListViewReorderMode.Animated;
list.showBorder = true;
list.headerTitle = "Title";
list.showAlternatingRowBackgrounds = AlternatingRowBackground.All;
list.selectionType = SelectionType.Single;
list.showAddRemoveFooter = true;

list.style.flexGrow = 1.0f;

list.Bind(serializedObject);
list.bindingPath = frames.propertyPath;

list.makeItem += () =>
{
    var foldout = new Foldout();
    var refLight = new PropertyField()
    {
        name = "ReferenceLightEntry"
    };
    var timeEntry = new PropertyField()
    {
        name = "TimeEntry"
    };

    foldout.Add(refLight);
    foldout.Add(timeEntry);

    return foldout;
};

list.bindItem += (element, i) =>
{
    var entry = frames.GetArrayElementAtIndex(i);

    (element as Foldout).text = entry.displayName;

    var refLight = entry.FindPropertyRelative(nameof(FarmerLightInterpolator.LightFrame.ReferenceLight));

    element.Q<PropertyField>("ReferenceLightEntry")
        .BindProperty(refLight);
    element.Q<PropertyField>("TimeEntry")
        .BindProperty(entry.FindPropertyRelative(nameof(FarmerLightInterpolator.LightFrame.NormalizedTime)));
};

list.unbindItem += (element, i) =>
{
    if (element.userData != null)
    {
        DestroyImmediate(element.userData as Editor);
    }
};

list.selectionChanged += objs =>
{

```

```

        if (!objs.Any())
            return;

        var first = objs.First() as SerializedProperty;
        m_PreviewSlider.value =
first.FindPropertyRelative(nameof(FarmerLightInterpolator.LightFrame.NormalizedTime)).floatValue;
    };

    list.itemsChosen += objects =>
    {
        var first = objects.First() as SerializedProperty;

        var target = first.FindPropertyRelative(nameof(FarmerLightInterpolator.LightFrame.ReferenceLight))
            .objectReferenceValue as Light2D;

        if (target != null) target.gameObject.SetActive(true);

        Selection.activeObject = target;
    };

    root.Add(list);
}
}
#endif
}

```

Листинг класса GameMenu:

```

using Assets.Scripts;
using Scripts.FarmGameEvents;
using Scripts.MainMenuCode;
using Scripts.MouseHandle;
using Scripts.SaveLoader;
using UnityEngine;
using UnityEngine.SceneManagement;
using Zenject;

namespace Scripts.GameMenuCode
{
    public class GameMenu : MonoBehaviour
    {
        [SerializeField] GameObject _menuPanel;
        InputService _inputService;
        GameState _gameDataState;
        GameStateSaveLoader _gameDataSaveLoader;
        SettingsMenu _settingsMenu;
        MouseCursor _mouseCursor;
        [Inject]
        public void Construct(InputService inputService,
            GameState gameDataState,
            SettingsMenu settingMenu,
            GameStateSaveLoader gameDataSaveLoader,
            MouseCursor mouseCursor)
        {
        }
    }
}

```

```

{
    _inputService = inputService;
    _gameDataState = gameDataState;
    _settingsMenu = settingMenu;
    _gameDataSaveLoader = gameDataSaveLoader;
    _mouseCursor = mouseCursor;
}

private void Start()
{
    if (_menuPanel.activeSelf == true)
    {
        _menuPanel.SetActive(false);
    }
    if(LoadedData.Instance() != null &&
        LoadedData.IsSettingsDataDefault == false)
    {
        _settingsMenu.Load(LoadedData.Instance().SettingsData);
    }
    _settingsMenu.InitializeKeyBindings();
}
public void OnContinue()
{
    if (_menuPanel.activeSelf)
    {
        _inputService.UnlockGamePlayControls();
        _menuPanel.SetActive(false);
        _mouseCursor.ChangeCursor(CursorType.Default);
    }
}
private void Update()
{
    if (_inputService.IsOpenCloseMenu())
    {
        if (_menuPanel.activeSelf == false)
        {
            _inputService.LockGamePlayControls();
            _menuPanel.SetActive(true);
            _mouseCursor.ChangeCursor(CursorType.Menu);
            return;
        }
        if (_menuPanel.activeSelf == true)
        {
            _inputService.UnlockGamePlayControls();
            _menuPanel.SetActive(false);
            _mouseCursor.ChangeCursor(CursorType.Default);
            return;
        }
    }
    if(Input.GetMouseButtonDown(0))
    {
        _mouseCursor.ImitateClick();
    }
}
public void OnSettings()
{
    _inputService.LockMenuControls();
}

```

```

        _settingsMenu.Activate();
        _menuPanel.SetActive(false);
    }
    public void OnBackFromSettings()
    {
        _inputService.UnlockMenuControls();
        _settingsMenu.Back();
        _menuPanel.SetActive(true);
    }
    public void OnExit()
    {
        GameEvents.InvokeExitTheGameEvent();

        _gameDataSaveLoader.SaveGameState(_gameDataState);

        _settingsMenu.Save();
        SceneManager.LoadScene(GameConfiguration.MainMenuSceneName);
    }
}

```

Листинг класса ChatContextOptimizer:

```

using System;
using SharpToken;
using UnityEngine;

namespace Assets.Scripts.GameSO.Chat
{
    public class ChatContextOptimizer
    {
        string _encodingModelId;
        int _restContextSize;
        GptEncoding _gptEncoding;
        public ChatContextOptimizer(string encodingModelId, int contextSize)
        {
            _encodingModelId = encodingModelId;
            _restContextSize = contextSize;
        }

        public bool TryInitialize()
        {
            try
            {
                _gptEncoding = GptEncoding.GetEncoding(_encodingModelId);

                return true;
            }
            catch (Exception ex)
            {
                Debug.LogError(ex);

                return false;
            }
        }
    }
}

```

```

public bool IsNeedReset(string text)
{
    var encodedCount = _gptEncoding.CountTokens(text);

    Debug.Log($"Current context size: {encodedCount}");

    return encodedCount >= _restContextSize;
}
}
}

```

Листинг класса ChatSO:

```

using Scripts.InventoryCode;
using System;
using UnityEngine;

namespace Scripts.SO.Chat
{
    [CreateAssetMenu(fileName = "ChatSO", menuName = "SO/ChatAssistant/ChatSO")]
    public class ChatSO : ScriptableObject, IDatabaseItem
    {
        public string Name;
        public string APIKey;
        public string Url;
        public string ModelId;
        [Multiline]
        public string SystemMessage;
        [Space]
        [Header("Request parameters:")]
        [Range(0f, 1f)]
        public double Temperature;
        [Header("Сколько токенов использовать для ответа")]
        [Range(0, 200)]
        public int MaxTokens;
        [Header("Название модели токенизатора")]
        public string EncodingModelId;
        [Header("Величина активного сеанса для очистки")]
        [Range(1500, 20000)]
        public int ResetContextSize;

        public string UniqueName => Name;
    }
}

```

Листинг класса ChatSODatabase:

```

using Scripts.InventoryCode;
using UnityEngine;

namespace Scripts.SO.Chat
{
    [CreateAssetMenu(fileName = "ChatSODatabase", menuName = "SO/ChatAssistant/DataBase")]

```

```

public class ChatSODataBase : DataBase<ChatSO>
{
    }
}

```

Листинг класса MessageSO:

```

using UnityEngine;

namespace Scripts.SO.Chat
{
    [CreateAssetMenu(fileName = "MessageSO", menuName = "SO/ChatAssistant/MessageSO")]
    public class MessageSO : ScriptableObject
    {
        public int WriteSpeed;
    }
}

```

Листинг класса InventoryInfo:

```

using UnityEngine;

namespace Scripts.SO.Inventory
{
    [CreateAssetMenu(fileName = "InventoryInfo", menuName = "SO/InventoryInfo")]
    public class InventoryInfo : ScriptableObject
    {
        public int TotalSize;
    }
}

```

Листинг класса ItemSourceSO:

```

using UnityEngine;
namespace Scripts.InventoryCode
{
    [CreateAssetMenu(fileName = "ItemSource", menuName = "SO/ItemSource")]
    public class ItemSourceSO : ScriptableObject
    {
        [Header("Скорость следования за игроком")]
        public float FollowSpeed;
        [Header("Дистанция, на которой предмет начинает следовать за игроком")]
        public float FollowDistance;
        [Header("Скорость, с которой вылетает ресурс из инвентаря")]
        public float PushSpeed;
        public float GravityScale;
        [Header("Дистанция, на которую перемещается ресурс, который выпал с игрока")]
        public float PushDistance;
        [Header("Дистанция, от игрока к ресурсу, при которой ресурс подбирается")]
        public float PickupDistance;
        [Header("Дистанция между конечной точкой броска и текущей, чтобы перейти в состояние <На земле>")]
        public float DistanceToChangeGroundState;
    }
}

```

```
}  
}
```

Листинг класса UseConditionSO:

```
using UnityEngine;  
  
namespace Scripts.SO.Inventory  
{  
    [CreateAssetMenu(fileName = "InventoryItem",  
        menuName = "SO/InventoryItems/UseCondition")]  
    public class UseConditionSO : ScriptableObject  
    {  
        public float InteractDistance;  
        [ColorUsage(true)]  
        public Color TrueConditionColor;  
        [ColorUsage(true)]  
        public Color FalseConditionColor;  
    }  
}
```

Листинг класса MouseIconSO:

```
using UnityEngine;  
using Scripts.MouseHandle;  
namespace Scripts.SO.MouseSO  
{  
    [CreateAssetMenu(fileName = "MouseIcon", menuName = "SO/Mouse/MouseIcon")]  
    public class MouseIconSO : ScriptableObject  
    {  
        public CursorType CursorType;  
        public Texture2D CursorIcon;  
    }  
}
```

Листинг класса PlayerSO:

```
using UnityEngine;  
namespace Scripts.SO.Player  
{  
    [CreateAssetMenu(fileName = "PlayerSO", menuName = "SO/PlayerSO")]  
    public class PlayerSO : ScriptableObject  
    {  
        public float Speed;  
        public int StartMoney;  
    }  
}
```

Листинг класса InputService:

```
using System;  
using UnityEngine;
```



```

using UnityEngine.InputSystem;
public class InputService : IDisposable
{
    const int c_firstItemCellIndex = 0;
    const int c_secondItemCellIndex = 1;
    const int c_thirdItemCellIndex = 2;
    const int c_fourthItemCellIndex = 3;
    InputActions _inputActions;

    public InputService()
    {
        _inputActions = new InputActions();
        _inputActions.Enable();
        InputActionAsset inputActionsAsset = _inputActions.asset;
    }

    public Vector2 GetMovement()
    {
        return _inputActions.PlayerMap.Movement.ReadValue<Vector2>();
    }

    public bool IsChosenCell(out int index)
    {
        index = 0;
        bool wasPerformed = false;
        if(_inputActions.InventoryMap.Choosefirstcell.WasPerformedThisFrame())
        {
            index = c_firstItemCellIndex;
            wasPerformed = true;
        }
        if(_inputActions.InventoryMap.Choosesecondcell.WasPerformedThisFrame())
        {
            index = c_secondItemCellIndex;
            wasPerformed = true;
        }
        if(_inputActions.InventoryMap.Choosethirdcell.WasPerformedThisFrame())
        {
            index = c_thirdItemCellIndex;
            wasPerformed = true;
        }
        if(_inputActions.InventoryMap.Choosefourthcell.WasPerformedThisFrame())
        {
            index = c_fourthItemCellIndex;
            wasPerformed = true;
        }
        return wasPerformed;
    }

    public bool IsLBK()
    {
        return _inputActions.PlayerMap.Interact.WasPerformedThisFrame();
    }

    public bool IsOpenCloseMenu()
    {
        return _inputActions.MenuActions.OpenClosegamemenu
            .WasPerformedThisFrame();
    }
}

```

```

public void LockGamePlayControls()
{
    _inputActions.PlayerMap.Disable();
}
public void UnlockGamePlayControls()
{
    _inputActions.PlayerMap.Enable();
}
public void LockMenuControls()
{
    _inputActions.MenuActions.Disable();
}
public void UnlockMenuControls()
{
    _inputActions.MenuActions.Enable();
}
public bool IsOpenCloseBackPack()
{
    return _inputActions.InventoryMap.OpenClosebackpack.WasPerformedThisFrame();
}
public void Dispose()
{
    _inputActions.Disable();
    _inputActions.Dispose();
}
public InputActionAsset GetInputActionAsset()
{
    return _inputActions.asset;
}
}

```

Листинг класса ChatAssistantInstaller:

```

using Zenject;
using UnityEngine;
using Scripts.ChatAssistant;
using Scripts.SO.Chat;

namespace Scripts.Installers
{
    public class ChatAssistantInstaller : MonoInstaller
    {
        [SerializeField] private ChatSODatabase _chatSODatabase;
        [SerializeField] private MessageSO _messageSO;
        [SerializeField] private MessagePanel _assistantMessagePanel;
        [SerializeField] private MessagePanel _playerMessagePanel;
        MessagePanelsFactories _messagePanelsFactories;
        public override void InstallBindings()
        {
            _messagePanelsFactories = new MessagePanelsFactories();
            _chatSODatabase.Init();
            Container.BindInstance(_chatSODatabase).AsSingle();
            Container.BindInstance(_messageSO).AsSingle();

            BindFactories();
        }
    }
}

```

```

        BindPanels();
    }

    void BindFactories()
    {
        _messagePanelsFactories.Add(MessagePanelType.Player, new MessagePanelFactory(_playerMessagePanel,
            Container));
        _messagePanelsFactories.Add(MessagePanelType.Assistant, new
MessagePanelFactory(_assistantMessagePanel,
            Container));

        Container.BindInstance(_messagePanelsFactories).AsSingle();
    }
    void BindPanels()
    {
        Container.Bind<ChatPanel>().
            FromComponentInHierarchy()
            .AsTransient();
        Container.Bind<MessagePanel>().AsTransient();
    }
}
}
}

```

Листинг класса GameInstaller:

```

using HappyHarvest;
using Scripts.GameMenuCode;
using Scripts.InteractableObjects;
using Scripts.InventoryCode;
using Scripts.MainMenuCode;
using Scripts.PlacementCode;
using Scripts.PlayerCode;
using Scripts.SaveLoader;
using Scripts.SO.Player;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Tilemaps;
using Scripts.SellBuy;
using Zenject;
using TMPro;
using Scripts.Inventory;
using Scripts.Sounds;
using Scripts.MouseHandle;

namespace Scripts.Installers
{
    public class GameInstaller : MonoInstaller
    {
        [Header("Grid:")]
        [SerializeField] private Grid _grid;
        [Header("Player binds:")]
        [Space]
        [SerializeField] private Transform _playerTransform;
        [SerializeField] PlayerSO PlayerSO;
        [Space]
    }
}

```

```

[Header("Game state binds:")]
[Space]
[Header("Дефолтные объекты на уровне")]
[SerializeField] private List<PlacementItem> _placementItems;
GameDataState _gameDataState;
[Space]
[Header("Item Usage:")]
[Space]
[Header("Elements map:")]
[SerializeField] private Tilemap _gameElementsMap;
[Header("Sand map:")]
[SerializeField] private Tilemap _sandMap;
[Header("KursorObject:")]
[SerializeField] private GameObject _kursorObject;
[Space]
[Header("InteractableObjects:")]
[Space]
[Header("Chest inventory:")]
[SerializeField] private ChestInventory _chestInventoryTemplate;
[Header("Chest object:")]
[SerializeField] private Chest _chestTemplate;
[Header("Sand rule tile:")]
[SerializeField] private RuleTile _sandRuleTile;
[Header("Seed template:")]
[SerializeField] private Seed _seedTemplate;
[Header("Tree template:")]
[SerializeField] private InteractableObjects.Tree _treeTemplate;
[Header("Crop data base:")]
[SerializeField] private CropDataBase _cropDataBase;
[Header("Item data base:")]
[SerializeField] private InventoryItemDataBase _itemDataBase;
[Header("Trading:")]
[SerializeField] private BuyItemsDatabase _buyItemsDatabase;
[SerializeField] private SellTradeElement SellTradeElementTemplate;
[SerializeField] private BuyTradeElement BuyTradeElementTemplate;
[SerializeField] private TextMeshProUGUI MoneyText;
[Header("Settings Panel")]
[SerializeField] private SettingsPanel _settingsPanel;
[SerializeField] private KeyBindingPanel keyBindingPanelTemplate;
InputService _inputService;
FactoriesProvider _factoryProvider;
[Header("Mouse Handles:")]
[SerializeField] private MouseIconDatabase _mouseIconDatabase;

public override void InstallBindings()
{
    _inputService = new InputService();
    _factoryProvider = new FactoriesProvider();
    BindInputService();
    BindGameMenu();
    BindGameDataState();
    BindPlayer();
    BindItemsUsage();
    BindChest();
    BindInteractableObjectsFactories();
    BindGrid();
}

```

```

        BindDataBases();
        BindTradeLogic();
        BindSounds();
        BindCursor();
        BindSeller();
        Container.Bind<NextLevel.NextLevel>()
            .FromComponentInHierarchy()
            .AsSingle();
        Container.BindInstance(_factoryProvider).AsSingle();
    }
    void BindSeller()
    {
        Container.Bind<Seller>().AsSingle();
    }
    void BindCursor()
    {
        MouseCursor mouseCursor = new MouseCursor(_mouseIconDatabase);
        mouseCursor.ChangeCursor(CursorType.Default);

        Container.BindInstance(mouseCursor).AsSingle();
    }
    void BindSounds()
    {
        Container.Bind<SoundService>()
            .FromComponentInHierarchy()
            .AsSingle();
        Container.Bind<StepSoundHandler>()
            .FromComponentInHierarchy()
            .AsSingle();
    }
    void BindTradeLogic()
    {
        Container.BindInstance(_buyItemsDatabase).AsSingle();
        PlayerMoney playerMoney = new PlayerMoney(PlayerSO.StartMoney, MoneyText);
        Container.BindInstance(playerMoney).AsSingle();
        SellTradeElementFactory sellTradeElementFactory = new
        SellTradeElementFactory(SellTradeElementTemplate,
            Container);
        BuyTradeElementFactory buyTradeElementFactory = new
        BuyTradeElementFactory(BuyTradeElementTemplate, Container);
        _factoryProvider.RegisterFabric(sellTradeElementFactory);
        _factoryProvider.RegisterFabric(buyTradeElementFactory);

        Container.Bind<TradeService>().AsSingle();
        Container.Bind<TradePanel>()
            .FromComponentInHierarchy()
            .AsSingle();
        Container.Bind<TradeElement>()
            .To<BuyTradeElement>().AsTransient();
        Container.Bind<TradeElement>()
            .To<SellTradeElement>().AsTransient();
    }
    void BindDataBases()
    {
        _itemDataBase.Init();
        _cropDataBase.Init();
        Container.BindInstance(_cropDataBase).AsSingle();
    }

```

```

        Container.BindInstance(_itemDataBase).AsSingle();
    }
    void BindGrid()
    {
        Container.BindInstance(_grid).AsSingle();
        Container.Bind<MarkerController>()
            .FromComponentInHierarchy()
            .AsSingle();
        Container.Bind<PlacementService>()
            .FromComponentInHierarchy()
            .AsSingle();
    }
    void BindInputService()
    {
        Container.BindInstance(_inputService).AsSingle();
    }
    void BindGameMenu()
    {
        Container.Bind<GameMenu>()
            .FromComponentInHierarchy()
            .AsSingle();
        _factoryProvider.RegisterFabric(new KeyBindingPanelFactory(keyBindingPanelTemplate, Container));
        SettingsMenu settingsMenu = new(_settingsPanel, _inputService, true);
        Container.BindInstance(settingsMenu).AsSingle();
    }
    void BindGameDataState()
    {
        if (LoadedData.IsGameStateDefault)
        {
            if (LoadedData.Instance() == null)
            {
                _gameDataState = new GameDataState("Editor mode");
            }
            else
            {
                _gameDataState = LoadedData.Instance().GameDataState;
            }
        }
        else
        {
            _gameDataState = LoadedData.Instance().GameDataState;
            foreach (var placementItem in _placementItems)
            {
                Destroy(placementItem.gameObject);
            }
        }
        Container.BindInstance(_gameDataState).AsSingle();
        Container.Bind<GameDataSaveLoader>().AsSingle();
        Container.Bind<SandTilePlacementSaver>().AsSingle();
    }
    void BindPlayer()
    {
        Container.Bind<Player>()
            .FromComponentInHierarchy()
            .AsSingle();
        Container.BindInstance(_playerTransform)
            .WithId("PlayerTransform")

```

```

        .AsTransient();
Container.BindInstance(PlayerSO).AsTransient();
Container.Bind<CharacterAnimationEventHandler>()
    .FromComponentInHierarchy()
    .AsSingle();
}
void BindItemsUsage()
{
    Container.BindInstance(_gameElementsMap)
        .AsTransient();

    ItemPlacementMap itemPlacementMap =
        new ItemPlacementMap(_gameElementsMap);
    SandTilePlacementMap sandTilePlacementMap =
        new SandTilePlacementMap(_sandMap);
    SeedPlacementMap seedPlacementMap =
        new SeedPlacementMap(_gameElementsMap);
    PlacementMapsContainer placementMapsContainer
        = new PlacementMapsContainer(
            itemPlacementMap,
            sandTilePlacementMap,
            seedPlacementMap);

    _kursorObject.SetActive(false);

    Container.BindInstance(_kursorObject).WithId("KursorObject");

    Container.BindInstance(placementMapsContainer).AsSingle();

    Container.Bind<PlacementItem>().To<Chest>().AsTransient();
    Container.Bind<PlacementItem>().To<Seed>().AsTransient();
}
void BindChest()
{
    Container.Bind<PlacementItem>()
        .To<Chest>()
        .FromComponentInHierarchy()
        .AsTransient();

    Container.BindInstance(_chestInventoryTemplate)
        .AsTransient();
    Container.Bind<IInventoryPanelFactory>()
        .To<InventoryChestPanelFactory>()
        .WhenInjectedInto<Chest>();

    IChestFactory chestFactory = new ChestFactory(Container, _chestTemplate);
    Container.BindInstance(chestFactory).AsTransient();
}
void BindInteractableObjectsFactories()
{
    SeedFactory seedFactory = new SeedFactory(Container, _seedTemplate);
    SandFactory sandFactory = new SandFactory(_sandMap, _sandRuleTile);
    ChestFactory chestFactory = new ChestFactory(Container, _chestTemplate);

    OakSeedFactory oakSeedFactory =
        new OakSeedFactory(Container, _treeTemplate);

```

```

        _factoryProvider.RegisterFabric(seedFactory);
        _factoryProvider.RegisterFabric(sandFactory);
        _factoryProvider.RegisterFabric(chestFactory);
        _factoryProvider.RegisterFabric(oakSeedFactory);
    }
}
}

```

Листинг класса InventoryInstaller:

```

using Scripts.InventoryCode;
using Scripts.InventoryCode.ItemResources;
using Scripts.SO.Inventory;
using UnityEngine;
using Zenject;

namespace Scripts.Installers
{
    public class InventoryInstaller : MonoInstaller
    {
        [Header("Canvas")]
        [SerializeField] Transform DragParent;
        [Header("Cell prefab")]
        [SerializeField] private InventoryCell CellTemplate;
        [Header("Empty cell prefab")]
        [SerializeField] private GameObject EmptyCellTemplate;
        [Space]
        [Header("Active inventory")]
        [SerializeField] private InventoryInfo _activeInventoryInfo;
        [Space]
        [Space]
        [Header("Backpack inventory")]
        [SerializeField] private InventoryInfo _storageInventoryInfo;
        [Space]
        [Header("ItemResourceSO")]
        [SerializeField] private ItemSourceSO ItemSourceSO;
        [Header("Item resource prefab")]
        [SerializeField] private ItemResource ItemResourcePrefab;
        [Header("Player Inventory object:")]
        [SerializeField] private PlayerInventory _playerInventory;

        public override void InstallBindings()
        {
            Container.BindInstance(_playerInventory).AsSingle();
            BindInventories();
            BindCellTemplate();
            BindGlobalVisualContext();
            BindFactories();
            BindItemResourceLogic();
            BindCells();
        }
        void BindGlobalVisualContext()
        {
            Container.BindInstance(DragParent)
                .WithId("DragParent")

```



```

        .AsTransient();
    }
    void BindCellTemplate()
    {
        Container.BindInstance(CellTemplate)
            .WithId("CellTemplate")
            .AsTransient();
        Container.BindInstance(EmptyCellTemplate)
            .WithId("EmptyCellTemplate")
            .AsTransient();
    }
    void BindInventories()
    {
        Container.Bind<InventoryBase>().To<InventoryStorage>()
            .FromComponentInHierarchy()
            .AsTransient();

        Container.BindInstance(_storageInventoryInfo)
            .WithId("InventoryStorageInfo")
            .WhenInjectedInto<InventoryStorage>();

        Container.BindInstance(_activeInventoryInfo)
            .WithId("ActiveInventoryInfo")
            .WhenInjectedInto<ActiveInventory>();

        Container.Bind<InventoryStorage>()
            .To<ActiveInventory>()
            .FromComponentInHierarchy()
            .AsSingle();

        Container.Bind<Pack>()
            .FromComponentInHierarchy()
            .AsSingle();
    }
    void BindFactories()
    {
        Container.Bind<IInventoryCellFactory>()
            .To<StorageCellFactory>()
            .WithArguments(CellTemplate, EmptyCellTemplate)
            .WhenInjectedInto<InventoryStorage>();
    }
    void BindItemResourceLogic()
    {
        Container.Bind<ItemResourceDroper>().AsSingle();
        Container.BindInstance(ItemSourceSO).AsTransient();
        Container.BindInstance(ItemResourcePrefab).AsTransient();
        Container.Bind<ItemResource>().AsTransient();
        Container.Bind<IItemResourceFactory>()
            .To<ItemResourceFactory>()
            .WithArguments(Container, ItemResourcePrefab)
            .WhenInjectedInto<ItemResourceDroper>();
    }
    void BindCells()
    {
        Container.Bind<InventoryCell>()
            .FromComponentInHierarchy()
            .AsTransient();
    }

```

```

    }
}

```

Листинг класса ItemsDataInstaller:

```

using System.Collections.Generic;
using Zenject;
using UnityEngine;
using Scripts.InventoryCode;
using Scripts.SO.InteractableObjects;

namespace Scripts.Installers
{
    public class ItemsDataInstaller : MonoInstaller
    {
        [Header("Все предметы инвентаря в игре")]
        [SerializeField] List<InventoryItem> _inventoryItemAssetList;
        [Header("Все данные для семян:")]
        [SerializeField] List<SeedSO> _seedDataList;

        public override void InstallBindings()
        {
            //BindInventoryItemsDictionary();
            BindSeedDataDictionary();
        }
        void BindInventoryItemsDictionary()
        {
            Dictionary<string, InventoryItem> InventoryItemsDictionary
                = new Dictionary<string, InventoryItem>();

            foreach (var item in _inventoryItemAssetList)
            {
                InventoryItemsDictionary[item.DisplayName] = item;
            }

            Container.BindInstance(InventoryItemsDictionary).AsSingle();
        }
        void BindSeedDataDictionary()
        {
            Dictionary<string, SeedSO> SeedDataDictionary
                = new Dictionary<string, SeedSO>();

            foreach (var seedSO in _seedDataList)
            {
                SeedDataDictionary[seedSO.name] = seedSO;
            }

            Container.BindInstance(SeedDataDictionary).AsSingle();
        }
    }
}

```

Листинг класса MainMenuInstaller:

```

using Zenject;
using UnityEngine;
using Scripts.MainMenuCode;
using Scripts.MainMenuScripts;
using Scripts.InteractableObjects;
using Scripts.MouseHandle;

namespace Scripts.Installers
{
    class MainMenuInstaller : MonoInstaller
    {
        [Header("Главное меню")]
        [SerializeField] private GameObject _menuObject;
        [Header("Меню загрузки")]
        [SerializeField] private GameStatePanel _statePanelTemplate;
        [SerializeField] private GameObject _loadMenuPanel;
        [SerializeField] private Transform _content;
        [Header("Меню создания")]
        [SerializeField] private NewGamePanel _newGamePanel;
        [Header("Меню настроек")]
        [SerializeField] private SettingsPanel _settingsPanel;
        [SerializeField] private KeyBindingPanel _keyBindingPanelTemplate;
        FactoriesProvider _factoriesProvider;
        [Header("Mouse Handles:")]
        [SerializeField] private MouseIconDatabase _mouseIconDatabase;
        public override void InstallBindings()
        {
            Container.Bind<InputService>().AsSingle();
            _factoriesProvider = new FactoriesProvider();
            BindMainMenu();
            BindLoadMenu();
            BindSettingsMenu();
            BindNewGameMenu();
            BindCursor();
            Container.BindInstance(_factoriesProvider)
                .AsSingle();
        }
        void BindCursor()
        {
            {
                MouseCursor mouseCursor = new MouseCursor(_mouseIconDatabase);
                mouseCursor.ChangeCursor(CursorType.Default);

                Container.BindInstance(mouseCursor).AsSingle();
            }
        }
        void BindMainMenu()
        {
            {
                Container.BindInstance(_menuObject)
                    .WithId("MainMenuObject")
                    .AsTransient();
                Container.Bind<MainMenu>()
                    .FromComponentInHierarchy()
                    .AsSingle();
            }
        }
        void BindLoadMenu()
    }
}

```

```

{
    Container.BindInstance(_statePanelTemplate)
        .WithId("StatePanelTemplate").AsTransient();
    Container.BindInstance(_loadMenuPanel).WithId("LoadMenuPanel")
        .AsTransient();
    Container.BindInstance(_content).WithId("LoadMenuContent")
        .AsTransient();
    Container.Bind<IGameStatePanelFactory>()
        .To<StatePanelFactory>()
        .WhenInjectedInto<LoadMenu>();
    Container.Bind<LoadMenu>().AsSingle();
}
void BindNewGameMenu()
{
    Container.BindInstance(_newGamePanel)
        .WithId("NewGamePanel")
        .AsTransient();
    Container.Bind<NewGameMenu>().AsSingle();
}
void BindSettingsMenu()
{
    KeyBindingPanelFactory keyBindingPanel = new KeyBindingPanelFactory(_keyBindingPanelTemplate,
        Container);
    _factoriesProvider.RegisterFabric(keyBindingPanel);
    Container.Bind<KeyBindingPanel>().AsTransient();
    InputService inputService = new InputService();
    SettingsMenu settingsMenu = new SettingsMenu(_settingsPanel, inputService, true);

    Container.BindInstance(settingsMenu).AsSingle();
    Container.Bind<MenuSound>()
        .FromComponentInHierarchy()
        .AsSingle();
}
}
}

```

Листинг класса NextLevelInstaller:

```

using HappyHarvest;
using Scripts.MainMenuCode;
using Scripts.SellBuy;
using Scripts.Sounds;
using Zenject;

namespace Scripts.Installers
{
    public class NextLevelInstaller : MonoInstaller
    {
        public override void InstallBindings()
        {
            Container.Bind<SoundService>()
                .FromComponentInHierarchy()
                .AsSingle();
            Container.Bind<BasicAnimalMovement>()

```

```

        .FromComponentInHierarchy()
        .AsTransient();
    Container.Bind<Seller>()
        .FromComponentInHierarchy()
        .AsSingle();
    SettingsMenu settingsMenu =
        new SettingsMenu(null, null, false);
    Container.BindInstance(settingsMenu).AsSingle();
    Container.Bind<InputService>().AsSingle();
}
}
}

```

Листинг класса Chest:

```

using Assets.Scripts.Placement;
using Scripts.InventoryCode;
using Scripts.PlacementCode;
using Scripts.SaveLoader;
using System.Collections.Generic;
using UnityEngine;
using Zenject;

namespace Scripts.InteractableObjects
{
    [RequireComponent(typeof(Animator))]

    public class Chest : PlacementItem, IOccupyingOneCell, ISaveLoadPlacementItem, IInteractable
    {
        Animator _animator;
        int _aniIsCloseCode;
        int _aniIsOpenCode;
        /// <summary>
        /// True если открыт, false если закрыт
        /// </summary>
        bool _openCloseFlag;
        List<InventoryItem> _inventoryItems;
        IInventoryPanelFactory _inventoryStoragePanelFactory;
        InventoryBase _chestStorage;

        [Inject]
        public void ConstructChest(
            IInventoryPanelFactory inventoryChestPanelFactory)
        {
            _inventoryStoragePanelFactory = inventoryChestPanelFactory;
        }

        protected override void Start()
        {
            _animator = GetComponent<Animator>();
            _aniIsCloseCode = Animator.StringToHash("IsClose");
            _aniIsOpenCode = Animator.StringToHash("IsOpen");
            if(_inventoryItems == null)
            {
                _inventoryItems = new List<InventoryItem>();
            }
        }
    }
}

```

```

    }
    _chestStorage = _inventoryStoragePanelFactory.Create(_inventoryItems);
    _chestStorage.gameObject.SetActive(false);
    _openCloseFlag = true;

    base.Start();
}

void Update()
{

}

private void OnMouseDown()
{
    if (_openCloseFlag == true)
    {
        _animator.SetBool(_aniIsOpenCode, _openCloseFlag);
        _animator.SetBool(_aniIsCloseCode, false);
        _chestStorage.gameObject.SetActive(true);
    }
    else
    {
        _animator.SetBool(_aniIsCloseCode, true);
        _animator.SetBool(_aniIsOpenCode, _openCloseFlag);
        _inventoryItems = _chestStorage.GetItems();
        _chestStorage.gameObject.SetActive(false);
    }
    _openCloseFlag = !_openCloseFlag;
}
public void Initialize(List<InventoryItem> inventoryItems)
{
    _inventoryItems = inventoryItems;
}
public Vector3Int GetOccupyingCell()
{
    return PlacePosition;
}

public override PlacementItemData GetData()
{
    ChestData chestData = new ChestData();
    var data = base.GetData();
    chestData.SetPosition(data.GetPosition());
    chestData.UpdateItems(_chestStorage.GetItems());
    return chestData;
}
}
}

```

Листинг класса IChestFactory:

```

using Scripts.InventoryCode;
using System.Collections.Generic;

```

```

namespace Scripts.InteractableObjects
{
    public interface IChestFactory : IGameObjectFactory< Chest, List<InventoryItem>>
    {
    }
}

```

Листинг класса IGameObjectFactory:

```

namespace Scripts.InteractableObjects
{
    public interface IGameObjectFactory<out T, in D>
    {
        T Create(D createData);
    }
}

```

Листинг класса IDatabaseItem:

```

using System.Collections.Generic;
using UnityEngine;

namespace Scripts.InventoryCode
{
    public interface IDatabaseItem
    {
        string UniqueName { get; }
    }
    public class DataBase<T> : ScriptableObject
        where T : class, IDatabaseItem
    {
        public List<T> Items;
        public Dictionary<string, T> _itemsDictionary;
        public void Init()
        {
            _itemsDictionary = new Dictionary<string, T>();

            foreach (var item in Items)
            {
                if(item != null)
                {
                    _itemsDictionary.TryAdd(item.UniqueName, item);
                }
            }
        }
        public T GetItemByName(string name)
        {
            if (_itemsDictionary.TryGetValue(name,
                out var item))
            {
                return item;
            }
            else
                return null;
        }
    }
}

```

```

    }
}

```

Листинг класса BuyItemsDatabase:

```

using Scripts.InventoryCode;
using UnityEngine;

namespace Scripts.Inventory
{
    [CreateAssetMenu(fileName = "BuyDataBase", menuName = "SO/DataBases/Buy")]
    public class BuyItemsDatabase : DataBase<SeedBagItem>
    {
    }
}

```

Листинг класса CropDataBase:

```

using UnityEngine;

namespace Scripts.InventoryCode
{
    [CreateAssetMenu(fileName = "CropDataBase", menuName = "SO/DataBases/Crop")]
    public class CropDataBase : DataBase<Crop>
    {
    }
}

```

Листинг класса InventoryItemDataBase:

```

using UnityEngine;

namespace Scripts.InventoryCode
{
    [CreateAssetMenu(fileName = "ItemsDataBase", menuName = "SO/DataBases/Item")]
    public class InventoryItemDataBase :
        DataBase<InventoryItem>
    {
    }
}

```

Листинг класса InventoryItemDataBase:

```

using UnityEngine;

namespace Scripts.InventoryCode
{
    [CreateAssetMenu(fileName = "ItemsDataBase", menuName = "SO/DataBases/Item")]
    public class InventoryItemDataBase :

```



```

        DataBase<InventoryItem>
    {
    }
}

```

Листинг класса ActiveInventory:

```

using Scripts.SO.Inventory;
using Zenject;
using Scripts.PlacementCode;
using Scripts.PlayerCode;
using Scripts.Sounds;
using Random = UnityEngine.Random;

namespace Scripts.InventoryCode
{
    public class ActiveInventory : InventoryStorage
    {
        InputService _inputService;
        int _chosenIndex;
        Player _player;
        SoundService _soundService;
        public InventoryItem ChosenItem { get; private set; }
        MarkerController _markerController;

        [Inject]
        public void ConstructActive(InputService inputService,
            [Inject(Id = "ActiveInventoryInfo")] InventoryInfo inventoryInfo,
            MarkerController markerController,
            Player player,
            SoundService soundService,
            IInventoryCellFactory inventoryCellFactory)
        {
            _markerController = markerController;
            _soundService = soundService;
            _inputService = inputService;
            _player = player;
            base.ConstructStorage(inventoryInfo, inventoryCellFactory);
        }

        public override void ConstructStorage([Inject(Id = "ActiveInventoryInfo")] InventoryInfo inventoryInfo,
            IInventoryCellFactory inventoryCellFactory)
        {
        }
        protected override void Start()
        {
            _chosenIndex = -1;
            _markerController.InteractMarker.Hide();
        }
        public override void Update()
        {
            if (_inputService.IsChosenCell(out _chosenIndex) &&
                _chosenIndex < CurrentSize)
            {
                SelectCellByIndex(_chosenIndex);
            }
        }
    }
}

```

```

    }

    ApplyItem();
}
void ApplyItem()
{
    if (ChosenItem != null &&
        ChosenItem.ApplyCondition(_markerController.CurrentTarget))
    {
        if(ChosenItem is ProductItem)
        {
            _markerController.InteractMarker.Hide();
        }
        else
        {
            _markerController.InteractMarker.Activate();
        }
    }

    if(_inputService.IsLBK())
    {
        bool used = ChosenItem.Apply(_markerController.CurrentTarget);
        if(used)
        {
            _player.UseItemVisual(ChosenItem);

            if(ChosenItem.Consumable)
            {
                ChosenItem.Count--;
            }

            if(ChosenItem.UseSound != null &&
                ChosenItem.UseSound.Length > 0)
            {
                _soundService.PlaySFXAt(_player.transform.position,
                    ChosenItem.UseSound[Random.Range(0, ChosenItem.UseSound.Length)], false);
            }
        }
    }
}
else
{
    _markerController.InteractMarker.Hide();
}
}

protected override void OnEndDrag()
{
    base.OnEndDrag();

    SelectFirstCell();
}
void SelectCellByIndex(int index)
{
    ChosenItem = InventoryItems[index];
    ChosenItem.IsSelected = true;
    DeactivateOther(ChosenItem);
}

```

```

    }
    void SelectFirstCell()
    {
        if (InventoryItems.Count > 0)
        {
            ChosenItem = InventoryItems[0];
            ChosenItem.IsSelected = true;
            DeactivateOther(ChosenItem);
        }
    }
    void DeactivateOther(InventoryItem selectedItem)
    {
        foreach (var item in InventoryItems)
        {
            if (selectedItem.Equals(item))
                continue;
            item.IsSelected = false;
        }
    }
    public override void OnDragInto(InventoryCell inventoryCell)
    {
    }
}
}

```

Листинг класса ChestInventory:

```

using UnityEngine;
using UnityEngine.EventSystems;

namespace Scripts.InventoryCode
{
    public class ChestInventory : InventoryStorage
    {
        private RectTransform rectTransform;
        private Canvas _canvas;
        protected override void Start()
        {
            base.Start();
            rectTransform = GetComponent<RectTransform>();
            _canvas = GetComponentInParent<Canvas>();
        }
        public override void OnDrag(PointerEventData eventData)
        {
            base.OnDrag(eventData);
            Vector2 localPoint;
            RectTransformUtility.ScreenPointToLocalPointInRectangle(
                _canvas.transform as RectTransform,
                Input.mousePosition,
                _canvas.worldCamera,
                out localPoint
            );
        }
    }
}

```

```

        rectTransform.localPosition = localPoint;
    }

    public override void Update()
    {
    }
}

```

Листинг класса InventoryBase:

```

using Scripts.Inventory;
using Scripts.MouseHandle;
using System;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;
using Zenject;

namespace Scripts.InventoryCode
{
    public abstract class InventoryBase : MonoBehaviour, IDragHandler, IBeginDragHandler, IEndDragHandler
    {
        [SerializeField] protected Transform ContainerField;
        public Transform Container
        {
            get { return ContainerField; }
            private set { ContainerField = value; }
        }
        protected int TotalSize;
        protected int CurrentSize => InventoryItems.Count;
        protected List<InventoryItem> InventoryItems;
        protected IInventoryCellFactory _inventoryCellFactory;
        private Transform _globalVisualContext;
        private RectTransform _contextRect;

        protected Action<InventoryCell> OnBeginDragEvent;
        protected Action OnEndDragEvent;

        private GameObject _tmpEmptyCell;
        MouseCursor _mouseCursor;

        [Inject]
        public void Construct([Inject(Id = "DragParent")] Transform dragParent,
            MouseCursor mouseCursor)
        {
            _globalVisualContext = dragParent;
            _mouseCursor = mouseCursor;
        }
        private void Awake()
        {
            _contextRect = gameObject.GetComponent<RectTransform>();
        }
    }
}

```

```

    }
    protected virtual void OnEnable()
    {
        DragExtension.RegisterInventoryRectTransform
            (_contextRect);

    }
    protected virtual void OnDisable()
    {
        DragExtension.UnregisterInventoryRectTransform(_contextRect);
    }
    private void OnDestroy()
    {
        OnBeginDragEvent -= OnBeginDragCell;
        OnEndDragEvent -= OnEndDrag;
    }
    protected virtual void Start()
    {
    }
    public void Initialize(List<InventoryItem> inventoryItems)
    {
        OnBeginDragEvent += OnBeginDragCell;
        OnEndDragEvent += OnEndDrag;
        InventoryItems = new List<InventoryItem>(TotalSize);
        InventoryItems.InsertRange(0, inventoryItems);
        foreach (var item in inventoryItems)
        {
            CreateCellForItem(item);
        }
        foreach (var item in inventoryItems)
        {
            item.IsSelected = false;
        }
    }

    public void RegisterDragEvents(InventoryCell inventoryCell)
    {
        inventoryCell.RegisterEvents(OnEndDragEvent, OnBeginDragEvent);
    }

    public bool AddItem(InventoryItem newItem)
    {
        InventoryItem newItemCopy = (InventoryItem)newItem.Clone();
        int remainingToFit = newItemCopy.Count;

        //first we check if there is already that item in the inventory
        for (int i = 0; i < InventoryItems.Count; ++i)
        {
            if (InventoryItems[i].UniqueName == newItemCopy.UniqueName
                && InventoryItems[i].Count < newItemCopy.MaxStackSize)
            {
                int fit = Mathf.Min(newItemCopy.MaxStackSize
                    - InventoryItems[i].Count,
                    remainingToFit);
                InventoryItems[i].Count += fit;
                remainingToFit -= fit;
            }
        }
    }

```

```

        if (remainingToFit == 0)
            return true;
    }
}

newItemCopy.Count = remainingToFit;
CreateCellForItem(newItemCopy);
InventoryItems = OverwriteInventoryItemsSequence();

return remainingToFit == 0;
}
public void RemoveItem(int itemIndex)
{
    for (int i = 0; i < Container.childCount; i++)
    {
        if(i == itemIndex)
        {
            Transform cell = Container.GetChild(i);
            Destroy(cell.gameObject);
        }
    }
}
void CreateCellForItem(InventoryItem inventoryItem)
{
    InventoryCell newCell = _inventoryCellFactory.Create(Container);
    newCell.Initialize(_globalVisualContext, inventoryItem);
    RegisterDragEvents(newCell);
}
/// <summary>
/// Возвращает false, если инвентарь заполнен
/// </summary>
/// <returns></returns>
public bool IsFull()
{
    if (CurrentSize >= TotalSize)
    {
        return true;
    }
    else
    {
        return false;
    }
}

protected virtual void OnBeginDragCell(InventoryCell inventoryCell)
{
    _tmpEmptyCell = _inventoryCellFactory.CreateEmpty(inventoryCell);
}
protected virtual void OnEndDrag()
{
    if (_tmpEmptyCell != null)
        Destroy(_tmpEmptyCell);
    InventoryItems = OverwriteInventoryItemsSequence();
}

```

```

    }

    public virtual void OnDrag(PointerEventData eventData)
    {

    }

    List<InventoryItem> OverwriteInventoryItemsSequence()
    {
        List<InventoryItem> items = new List<InventoryItem>(TotalSize);
        for (int i = 0; i < Container.childCount; i++)
        {
            var cellObject = Container.GetChild(i);
            InventoryCell inventoryCell;
            if (cellObject.TryGetComponent(out inventoryCell))
            {
                items.Add(inventoryCell.InventoryItem);
            }
        }
        return items;
    }
    public List<InventoryItem> GetItems()
    {
        return InventoryItems;
    }
    public List<ItemContextData> GetItemsContextData()
    {
        List<ItemContextData> itemContextDatas = new List<ItemContextData>();

        for (int i = 0; i < Container.childCount; i++)
        {
            var name = Container.name;
            var index = i;
            if (Container.GetChild(index).TryGetComponent(out InventoryCell cell))
            {
                ItemContextData itemContextData = new ItemContextData(
                    cell.InventoryItem, index, name);

                itemContextDatas.Add(itemContextData);
            }
        }
        return itemContextDatas;
    }

    public virtual void OnDragInto(InventoryCell inventoryCell)
    {

    }

    public void OnBeginDrag(PointerEventData eventData)
    {
        _mouseCursor.ChangeCursor(CursorType.Drag);
    }

    public void OnEndDrag(PointerEventData eventData)
    {

```

```

        _mouseCursor.ChangeCursor(CursorType.Default);
    }
}
}

```

Листинг класса InventoryStorage:

```

using Scripts.SO.Inventory;
using UnityEngine.EventSystems;
using Zenject;
using UnityEngine;

namespace Scripts.InventoryCode
{
    public class InventoryStorage : InventoryBase, IPointerEnterHandler, IPointerExitHandler
    {
        public static bool IsMouseStay { get; private set; }
        [Inject]
        public virtual void ConstructStorage(
            [Inject(Id = "InventoryStorageInfo")] InventoryInfo inventoryInfo,
            IInventoryCellFactory inventoryCellFactory)
        {
            TotalSize = inventoryInfo.TotalSize;
            _inventoryCellFactory = inventoryCellFactory;
        }
        protected override void OnEnable()
        {
            base.OnEnable();
        }
        protected override void OnDisable()
        {
            base.OnDisable();
        }
        protected override void OnEndDrag()
        {
            base.OnEndDrag();
        }
        public override void OnDragInto(InventoryCell inventoryCell)
        {
            base.OnDragInto(inventoryCell);
            inventoryCell.InventoryItem.IsSelected = false;
        }

        public void OnPointerEnter(PointerEventData eventData)
        {
            Debug.Log(" OnPointerEnter");
            IsMouseStay = true;
        }

        public void OnPointerExit(PointerEventData eventData)
        {
            Debug.Log(" OnPointerExit");
            IsMouseStay = false;
        }
    }
}

```



```

    }
    public virtual void Update()
    {

    }
}

```

Листинг класса InventoryCellFactory:

```

using UnityEngine;

namespace Scripts.InventoryCode
{
    public interface IInventoryCellFactory
    {
        InventoryCell Create(Transform visualContext);
        GameObject CreateEmpty(InventoryCell inventoryCell);
    }
}

```

Листинг класса StorageCellFactory:

```

using UnityEngine;
using Zenject;

namespace Scripts.InventoryCode
{
    public class StorageCellFactory : IInventoryCellFactory
    {
        InventoryCell _inventoryCellTemplate;
        GameObject _emptyCellPrefab;
        DiContainer _container;
        public StorageCellFactory(DiContainer diContainer,
            InventoryCell inventoryCellTemplate,
            GameObject emptyCellPrefab)
        {
            _inventoryCellTemplate = inventoryCellTemplate;
            _emptyCellPrefab = emptyCellPrefab;
            _container = diContainer;
        }

        public InventoryCell Create(Transform visualContext)
        {
            return _container.InstantiatePrefabForComponent<InventoryCell>
                (_inventoryCellTemplate, visualContext);
        }

        public GameObject CreateEmpty(InventoryCell inventoryCell)
        {
            int siblingIndex = inventoryCell.transform.GetSiblingIndex();
            var cell = GameObject.Instantiate(_emptyCellPrefab, inventoryCell.OriginVisualContext);
            cell.name = "SiblingCell";
        }
    }
}

```

```

        cell.transform.SetSiblingIndex(siblingIndex);
        return cell;
    }
}

```

Листинг класса ItemResourceFactory:

```

using UnityEngine;
namespace Scripts.InventoryCode.ItemResources
{
    public interface IItemResourceFactory
    {
        ItemResource Create(InventoryItem inventoryItem, Vector3 position);
    }
}

```

Листинг класса ItemResourceFactory:

```

using UnityEngine;
using Zenject;

namespace Scripts.InventoryCode.ItemResources
{
    public class ItemResourceFactory : IItemResourceFactory
    {
        ItemResource _itemResourceTemplate;
        DiContainer _container;
        public ItemResourceFactory(DiContainer diContainer, ItemResource itemResourceTemplate)
        {
            _itemResourceTemplate = itemResourceTemplate;
            _container = diContainer;
        }
        public ItemResource Create(InventoryItem inventoryItem,
            Vector3 position)
        {
            var item = _container.InstantiatePrefabForComponent<ItemResource>
                (_itemResourceTemplate,
                position, Quaternion.identity, null);
            item.Initialize(inventoryItem);
            return item;
        }
    }
}

```

```

using UnityEngine;
using Zenject;

namespace Scripts.InventoryCode.ItemResources
{
    public class ItemResourceDroper
    {

```

```

Transform _playerTransform;
IItemResourceFactory _factory;
public ItemResourceDroper(
    [Inject(Id = "PlayerTransform")] Transform playerTransform,
    IItemResourceFactory itemResourceFactory)
{
    _playerTransform = playerTransform;
    _factory = itemResourceFactory;
}
public void DropByPlayer(InventoryItem inventoryItem)
{
    ItemResource itemResource = _factory.Create(inventoryItem, _playerTransform.position);
}
}
}

```

Листинг класса ItemResource:

```

using UnityEngine;
using Zenject;

namespace Scripts.InventoryCode.ItemResources
{
    [RequireComponent(typeof(Rigidbody2D))]
    public class ItemResource : MonoBehaviour
    {
        [SerializeField] SpriteRenderer ItemVisualRenderer;
        ItemSourceSO _itemSourceSO;
        ItemResourceStateMachine _stateMachine;
        Transform _playerTransform;
        Sprite _icon;
        public PlayerInventory PlayerInventory { get; private set; }
        public InventoryItem InventoryItem { get; private set; }
        public Rigidbody2D Rigidbody { get; private set; }
        public SpriteRenderer SpriteRenderer
        {
            get { return ItemVisualRenderer; }
        }

        public PushState PushState { get; private set; }
        public FollowState FollowState { get; private set; }
        public OnGroundState OnGroundState { get; private set; }

        [Inject]
        public void Construct(ItemSourceSO itemSourceSO,
            [Inject(Id = "PlayerTransform")] Transform playerTransform,
            PlayerInventory playerInventory)
        {
            PlayerInventory = playerInventory;
            _itemSourceSO = itemSourceSO;
            _playerTransform = playerTransform;
        }

        private void Start()

```

```

{
    Rigidbody = GetComponent<Rigidbody2D>();
    Rigidbody.gravityScale = _itemSourceSO.GravityScale;
    SpriteRenderer.sprite = _icon;
    InitializeStateMachine();
}
private void Update()
{
    _stateMachine.Perform();
}
private void FixedUpdate()
{
    _stateMachine.FixedPerform();
}
public void Initialize(InventoryItem inventoryItem)
{
    InventoryItem = inventoryItem;
    _icon = inventoryItem.Icon;
}
void InitializeStateMachine()
{
    _stateMachine = new ItemResourceStateMachine(this);

    PushState = new PushState(_stateMachine, this, _itemSourceSO);
    FollowState = new FollowState(_stateMachine, this, _itemSourceSO);
    OnGroundState = new OnGroundState(_stateMachine, this, _itemSourceSO);
    _stateMachine.Initialize(PushState);
}
public bool PickupResource()
{
    return PlayerInventory.TryPickupResource(this);
}
public float GetDistanceToPlayer()
{
    return Vector2.Distance(_playerTransform.position,
        Rigidbody.position);
}
public Vector2 GetPlayerDirectionVector()
{
    Vector2 playerPos = _playerTransform.position;
    Vector2 itemPos = transform.position;
    Vector2 direction = playerPos - itemPos;
    return direction.normalized;
}
}
}

```

Листинг класса ItemResourceState:

```

using Scripts.StateMachine;

namespace Scripts.InventoryCode.ItemResources
{
    public abstract class ItemResourceState : IUniversalState
    {

```

```

protected ItemResourceStateMachine StateMachine { get; set; }
protected ItemResource ItemResource;
protected ItemSourceSO SourceSO;
public ItemResourceState(ItemResourceStateMachine universalStateMachine,
    ItemResource itemResource,
    ItemSourceSO sourceSO)
{
    StateMachine = universalStateMachine;
    ItemResource = itemResource;
    SourceSO = sourceSO;
}
public abstract void Enter();
public abstract void Exit();
public abstract void Perform();
public abstract void FixedPerform();
}
}

```

Листинг класса FollowState:

```

using UnityEngine;

namespace Scripts.InventoryCode.ItemResources
{
    public class FollowState : ItemResourceState
    {
        Vector2 _direction;
        Rigidbody2D _rigidBody;
        public FollowState(ItemResourceStateMachine universalStateMachine,
            ItemResource itemResource,
            ItemSourceSO sourceSO)
            : base(universalStateMachine, itemResource, sourceSO)
        {
            _rigidBody = itemResource.RigidBody;
        }

        public override void Enter()
        {
        }

        public override void Exit()
        {
        }

        public override void FixedPerform()
        {
            Vector2 newPos = Vector2.Lerp(_rigidBody.position,
                _rigidBody.position + _direction,
                Time.fixedDeltaTime * SourceSO.FollowSpeed);
            _rigidBody.MovePosition(newPos);
        }

        public override void Perform()
        {
            _direction = ItemResource.GetPlayerDirectionVector();
        }
    }
}

```

```

float distance = ItemResource.GetDistanceToPlayer();

if (distance >= SourceSO.FollowDistance)
{
    StateMachine.ChangeOnGroundState();
}
if (distance <= SourceSO.PickupDistance)
{
    if (ItemResource.PickupResource())
    {
        ItemResource.Destroy(ItemResource.gameObject);
    }
}
}
}
}

```

Листинг класса OnGroundState:

```

using UnityEngine;

namespace Scripts.InventoryCode.ItemResources
{
    public class OnGroundState : ItemResourceState
    {
        public OnGroundState(ItemResourceStateMachine universalStateMachine,
            ItemResource itemResource,
            ItemSourceSO sourceSO)
            : base(universalStateMachine, itemResource, sourceSO)
        {
        }

        public override void Enter()
        {
        }

        public override void Exit()
        {
        }

        public override void FixedPerform()
        {
        }

        public override void Perform()
        {
            float distance = ItemResource.GetDistanceToPlayer();
            if (distance <= SourceSO.FollowDistance)
            {
                Debug.Log("Вижу игрока!!");
                StateMachine.ChangeFollowState();
            }
        }
    }
}

```

Листинг класса PushState:

```
using UnityEngine;

namespace Scripts.InventoryCode.ItemResources
{
    public class PushState : ItemResourceState
    {
        float _pushSpeed;
        float _pushDistance;
        Vector2 _direction;
        Rigidbody2D _rigidBody;
        Vector2 _goal;
        public PushState(ItemResourceStateMachine universalStateMachine,
            ItemResource itemResource, ItemSourceSO itemSourceSO)
            : base(universalStateMachine, itemResource, itemSourceSO)
        {
            _pushSpeed = itemSourceSO.PushSpeed;
            _pushDistance = itemSourceSO.PushDistance;
            _direction = Vector2.down;
            _rigidBody = itemResource.RigidBody;
        }

        public override void Enter()
        {
            _goal = _rigidBody.position + _direction * _pushDistance;
        }

        public override void Exit()
        {
        }

        public override void FixedPerform()
        {
            Vector2 newPos = Vector2.Lerp(
                _rigidBody.position,
                _goal,
                Time.fixedDeltaTime * _pushSpeed);
            _rigidBody.MovePosition(newPos);
        }

        public override void Perform()
        {
            if(Vector2.Distance(_rigidBody.position, _goal) <
                SourceSO.DistanceToChangeGroundState)
            {
                Debug.Log("Я дошел до точки перехода!");
                StateMachine.ChangeOnGroundState();
            }
        }
    }
}
```

```

namespace Scripts.InventoryCode
{
    public class InventoryItemData
    {
        public string SoName;
        public int Amount;
        public string InventoryItemTypeName;
        public InventoryItemData()
        {
            InventoryItemTypeName = nameof(InventoryItemData);
        }
        public void Init(InventoryItemData inventoryItemData)
        {
            SoName = inventoryItemData.SoName;
            Amount = inventoryItemData.Amount;
        }
    }
    public class BasketItemData : InventoryItemData
    {
        public BasketItemData()
        {
            InventoryItemTypeName = nameof(BasketItemData);
        }
    }
    public class HoelItemData : InventoryItemData
    {
        public HoelItemData()
        {
            InventoryItemTypeName = nameof(HoelItemData);
        }
    }
    public class ProductItemData : InventoryItemData
    {
        public ProductItemData()
        {
            InventoryItemTypeName = nameof(ProductItemData);
        }
    }
    public class SeedBagItemData : InventoryItemData
    {
        public SeedBagItemData()
        {
            InventoryItemTypeName = nameof(SeedBagItemData);
        }
    }
    public class WateringItemData : InventoryItemData
    {
        public WateringItemData()
        {
            InventoryItemTypeName = nameof(WateringItemData);
        }
    }
}

```

Листинг класса BasketItem:


```

using Scripts.PlacementCode;
using UnityEngine;

namespace Scripts.InventoryCode
{
    [CreateAssetMenu(fileName = "Basket", menuName = "SO/InventoryItems/Basket")]
    public class BasketItem : InventoryItem
    {
        public override bool Apply(Vector3Int target)
        {
            var product = PlacementService.Instance().HarvestAt(target);
            if(product != null)
            {
                if (PlayerInventory.Instance().TryAddItem(product.Produce))
                {
                    return true;
                }
                else { return false; }
            }
            return false;
        }

        public override bool ApplyCondition(Vector3Int target)
        {
            var data = PlacementService.Instance().GetCropDataAt(target);
            return data != null && data.GrowingCrop != null && Mathf.Approximately(data.GrowthRatio, 1.0f);
        }

        public override object Clone()
        {
            BasketItem basketItem =
                CreateInstance<BasketItem>();
            basketItem.InitializeCopy(this);
            return basketItem;
        }

        public override void RenderUI(InventoryCell inventoryCell)
        {
            base.RenderUI(inventoryCell);
        }

        public override InventoryItemData GetData()
        {
            BasketItemData basketItemData = new();
            basketItemData.Init(base.GetData());
            return basketItemData;
        }
    }
}

```

Листинг класса Crop:

```

using UnityEngine;

```

```

using UnityEngine.Tilemaps;
using UnityEngine.VFX;

namespace Scripts.InventoryCode
{
    [CreateAssetMenu(fileName = "Crop", menuName = "SO/InventoryItems/Crop")]
    public class Crop : ScriptableObject,
        IDatabaseItem
    {
        public string UniqueName => _name;
        [SerializeField] private string _name;

        public TileBase[] GrowthStagesTiles;

        public ProductItem Produce;

        public float GrowthTime = 1.0f;
        public int NumberOfHarvest = 1;
        public int StageAfterHarvest = 1;
        public int ProductPerHarvest = 1;
        public float DryDeathTimer = 30.0f;
        public VisualEffect PickEffect;

        public int GetGrowthStage(float growRatio)
        {
            return Mathf.FloorToInt(growRatio * (GrowthStagesTiles.Length - 1));
        }
    }
}

```

Листинг класса HoeItem:

```

using Scripts.PlacementCode;
using UnityEngine;

namespace Scripts.InventoryCode
{
    [CreateAssetMenu(fileName = "Hoe", menuName = "SO/InventoryItems/Hoe")]
    public class HoeItem : InventoryItem
    {
        public override bool Apply(Vector3Int target)
        {
            PlacementService.Instance().TillAt(target);
            return true;
        }

        public override bool ApplyCondition(Vector3Int target)
        {
            return PlacementService.Instance() != null &&
                PlacementService.Instance().IsTillable(target);
        }

        public override object Clone()

```

```

    {
        HoelItem hoelItem =
            CreateInstance<HoelItem>();
        hoelItem.InitializeCopy(this);
        return hoelItem;
    }
    public override InventoryItemData GetData()
    {
        HoelItemData hoelItemData = new();
        hoelItemData.Init(base.GetData());
        return hoelItemData;
    }
}
}

```

Листинг класса InventoryItem:

```

using System;
using UnityEngine;

namespace Scripts.InventoryCode
{
    public abstract class InventoryItem : ScriptableObject, IDataBaselItem, ICloneable
    {
        public bool IsSelected { get; set; }

        public string DisplayName = "";

        public Sprite Icon => _icon;

        public Color Color => _color;

        public string UniqueName => _name;

        public int MaxStackSize = 10;
        public int Count
        {
            get
            {
                return _startCount;
            }
            set
            {
                _startCount = value;

                if (_startCount < 0)
                    _startCount = 0;
            }
        }

        [SerializeField] private int _startCount;
        public bool Consumable = true;
        public int BuyPrice = -1;
    }
}

```

```

[SerializeField] private string _name;

[SerializeField] private Sprite _icon;

[ColorUsage(true)]
[SerializeField] private Color _color;

[SerializeField] protected bool IsCountTextActive;

[Tooltip("Prefab that will be instantiated in the player hand when this is equipped")]
public GameObject VisualPrefab;

[Tooltip("Sound triggered when using the item")]
public AudioClip[] UseSound;

public string PlayerAnimatorTriggerUse = "GenericToolSwing";

public virtual void InitializeCopy(InventoryItem inventoryItem)
{
    DisplayName = inventoryItem.DisplayName;
    _icon = inventoryItem.Icon;
    _color = inventoryItem.Color;
    _name = inventoryItem.UniqueName;
    _startCount = inventoryItem.Count;
    MaxStackSize = inventoryItem.MaxStackSize;
    BuyPrice = inventoryItem.BuyPrice;
    Consumable = inventoryItem.Consumable;
    IsCountTextActive = inventoryItem.IsCountTextActive;
    VisualPrefab = inventoryItem.VisualPrefab;
    UseSound = inventoryItem.UseSound;
    PlayerAnimatorTriggerUse = inventoryItem.PlayerAnimatorTriggerUse;
}
public abstract bool ApplyCondition(Vector3Int target);

public abstract bool Apply(Vector3Int target);

public virtual void RenderUI(InventoryCell inventoryCell)
{
    inventoryCell.Icon.sprite = _icon;
    inventoryCell.NameDisplayText.text = DisplayName;
    inventoryCell.NameDisplayText.color = _color;
    inventoryCell.CountText.color = _color;
    inventoryCell.CountText.text = Count.ToString();
    inventoryCell.CountText.gameObject.SetActive(IsCountTextActive);
    inventoryCell.SelectIcon.gameObject.SetActive(IsSelected);
}

public virtual bool NeedTarget()
{
    return true;
}

public abstract object Clone();
public virtual InventoryItemData GetData()
{

```

```

InventoryItemData inventoryItemData = new InventoryItemData()
{
    SoName = UniqueName,
    Amount = Count
};
return inventoryItemData;
}

}
}

```

Листинг класса ProductItem:

```

using UnityEngine;

namespace Scripts.InventoryCode
{
    [CreateAssetMenu(fileName = "Product", menuName = "SO/InventoryItems/Product")]
    public class ProductItem : InventoryItem
    {
        public int SellPrice = 1;

        public override void InitializeCopy(InventoryItem inventoryItem)
        {
            base.InitializeCopy(inventoryItem);
            if(inventoryItem is ProductItem product)
            {
                SellPrice = product.SellPrice;
            }
        }
        public override bool Apply(Vector3Int target)
        {
            return true;
        }

        public override void RenderUI(InventoryCell inventoryCell)
        {
            base.RenderUI(inventoryCell);
        }

        public override bool ApplyCondition(Vector3Int target)
        {
            return Count > 0;
        }

        public override bool NeedTarget()
        {
            return false;
        }

        public override object Clone()
        {
            ProductItem productItem =
                CreateInstance<ProductItem>();

```

```

        productItem.InitializeCopy(this);
        return productItem;
    }
    public override InventoryItemData GetData()
    {
        ProductItemData productItemData = new();
        productItemData.Init(base.GetData());
        return productItemData;
    }
}

```

Листинг класса SeedBagItem:

```

using Scripts.PlacementCode;
using UnityEngine;

namespace Scripts.InventoryCode
{
    [CreateAssetMenu(fileName = "SeedBag", menuName = "SO/InventoryItems/SeedBag")]
    public class SeedBagItem : InventoryItem
    {
        public Crop CropToPlant;

        public override void InitializeCopy(InventoryItem inventoryItem)
        {
            base.InitializeCopy(inventoryItem);
            if(inventoryItem is SeedBagItem bag)
            {
                CropToPlant = bag.CropToPlant;
            }
        }
        public override bool Apply(Vector3Int target)
        {
            PlacementService.Instance().PlantAt(target, CropToPlant);
            return true;
        }

        public override bool ApplyCondition(Vector3Int target)
        {
            {
                return PlacementService.Instance().IsPlantable(target) &&
                    Count > 0;
            }
        }
        public override void RenderUI(InventoryCell inventoryCell)
        {
            base.RenderUI(inventoryCell);
        }

        public override object Clone()
        {
            {
                SeedBagItem bagItem =
                    CreateInstance<SeedBagItem>();
                bagItem.InitializeCopy(this);
                return bagItem;
            }
        }
    }
}

```

```

public override InventoryItemData GetData()
{
    SeedBagItemData seedBagItemData = new();
    seedBagItemData.Init(base.GetData());
    return seedBagItemData;
}
}
}

```

Листинг класса **WateringItem**:

```

using Scripts.PlacementCode;
using UnityEngine;

namespace Scripts.InventoryCode
{
    [CreateAssetMenu(fileName = "Watering", menuName = "SO/InventoryItems/Watering")]
    public class WateringItem : InventoryItem
    {
        public override bool Apply(Vector3Int target)
        {
            PlacementService.Instance().WaterAt(target);
            return true;
        }

        public override bool ApplyCondition(Vector3Int target)
        {
            return PlacementService.Instance().IsTilled(target);
        }

        public override object Clone()
        {
            WateringItem wateringItem =
                CreateInstance<WateringItem>();
            wateringItem.InitializeCopy(this);
            return wateringItem;
        }
        public override InventoryItemData GetData()
        {
            WateringItemData wateringItemData = new();
            wateringItemData.Init(base.GetData());
            return wateringItemData;
        }
    }
}

```

Листинг класса **InventoryPanelFactory**:

```

using System.Collections.Generic;

namespace Scripts.InventoryCode
{
    public interface IInventoryPanelFactory

```

```

{
    public InventoryBase Create(List<InventoryItem> inventoryItems);
}
}

```

Листинг класса InventoryChestPanelFactory:

```

using System.Collections.Generic;
using Zenject;
using UnityEngine;

namespace Scripts.InventoryCode
{
    public class InventoryChestPanelFactory : IInventoryPanelFactory
    {
        DiContainer _container;
        ChestInventory _chestInventoryTemplate;
        [Inject(Id = "DragParent")] Transform _globalVisualContext;
        public InventoryChestPanelFactory(DiContainer diContainer,
            /*[Inject(Id = "ChestPanelTemplate")]*/ ChestInventory chestInventoryTemplate
        )
        {
            _container = diContainer;
            _chestInventoryTemplate = chestInventoryTemplate;
        }
        public InventoryBase Create(List<InventoryItem> inventoryItems)
        {
            var storage = _container.
                InstantiatePrefabForComponent<ChestInventory>(_chestInventoryTemplate,
                    _globalVisualContext);
            storage.Initialize(inventoryItems);
            return storage;
        }
    }
}

```

Листинг класса InventoryStoragePanelFactory:

```

using System;
using System.Collections.Generic;
using UnityEngine;
using Zenject;

namespace Scripts.InventoryCode
{
    public class InventoryStoragePanelFactory : IInventoryPanelFactory
    {
        DiContainer _container;
        InventoryStorage _storageTemplate;
        [Inject(Id = "DragParent")] Transform _globalVisualContext;
        public InventoryStoragePanelFactory(DiContainer diContainer,
            [Inject(Id = "StoragePanelTemplate")] InventoryStorage storageTemplate
        )
        {

```



```

        _container = diContainer;
        _storageTemplate = storageTemplate;
    }
    public InventoryBase Create(List<InventoryItem> inventoryItems)
    {
        var storage = _container.
            InstantiatePrefabForComponent<InventoryStorage>(_storageTemplate,
                _globalVisualContext);
        storage.Initialize(inventoryItems);
        return storage;
    }
}

```

Листинг класса DragExtension:

```

using UnityEngine;
using System.Collections.Generic;
using UnityEngine.EventSystems;
using System.Threading.Tasks;

namespace Scripts.InventoryCode
{
    internal static class DragExtension
    {
        static List<RectTransform> _containersRectTransform;
        static readonly int c_MoveCellSpeed;
        static DragExtension()
        {
            if (_containersRectTransform == null)
            {
                _containersRectTransform = new List<RectTransform>();
            }
            c_MoveCellSpeed = 000000001;
        }
        public static void RegisterInventoryRectTransform(
            RectTransform containerRectTransform)
        {
            _containersRectTransform.Add(containerRectTransform);
        }
        public static void UnregisterInventoryRectTransform(
            RectTransform containerRectTransform)
        {
            _containersRectTransform.Remove(containerRectTransform);
        }

        public static bool CheckMouseIntersectionWithContainers(
            PointerEventData eventData, out InventoryBase inventory)
        {
            foreach (var rect in _containersRectTransform)
            {
                Vector2 mousePosition;
                RectTransformUtility.ScreenPointToLocalPointInRectangle(
                    rect,

```

```

        eventData.position,
        eventData.pressEventCamera,
        out mousePosition);

    if(rect.rect.Contains(mousePosition))
    {
        inventory = rect.gameObject.GetComponent<InventoryBase>();
        return true;
    }
}
inventory = null;
return false;
}
public static async Task PlaceInTheNearestCellLocal(Transform visualContext,
InventoryCell inventoryCell, int currentIndex)
{
    int closetIndex = GetNearestIndex(visualContext, inventoryCell);

    inventoryCell.transform.SetParent(visualContext);
    await MoveCellTo(inventoryCell.transform, closetIndex, currentIndex);
}
public static void PlaceInTheNearestCellGlobal(Transform visualContext,
InventoryCell inventoryCell)
{
    int closetIndex = GetNearestIndex(visualContext, inventoryCell);

    inventoryCell.transform.SetParent(visualContext);

    inventoryCell.transform.SetSiblingIndex(closetIndex);
}

static int GetNearestIndex(Transform visualContext,
InventoryCell inventoryCell)
{
    int closetIndex = 0;
    for (int i = 0; i < visualContext.transform.childCount; i++)
    {
        if (Vector3.Distance(visualContext.GetChild(i).position,
            inventoryCell.transform.position) <
            Vector3.Distance(visualContext.GetChild(closetIndex).position,
            inventoryCell.transform.position))
        {
            closetIndex = i;
        }
    }
    return closetIndex;
}
public static Transform GetNearestCellTransform(Transform visualContext,
InventoryCell inventoryCell)
{
    int closetIndex = GetNearestIndex(visualContext, inventoryCell);

    return visualContext.GetChild(closetIndex);
}

static async Task MoveCellTo(Transform cellTransform, int closetIndex, int currentIndex)

```

```

    {
        if(closetIndex > currentIndex)
        {
            for (int i = currentIndex; i <= closetIndex; i++)
            {
                cellTransform.SetSiblingIndex(i);
                await Task.Delay(c_MoveCellSpeed);
            }
        }
        else
        {
            for(int i = currentIndex; i >= closetIndex; i--)
            {
                cellTransform.SetSiblingIndex(i);
                await Task.Delay(c_MoveCellSpeed);
            }
        }
    }

    static async void Replace(InventoryCell inventoryCell1, InventoryCell inventoryCell2)
    {
        inventoryCell1.transform.SetParent(inventoryCell2.OriginVisualContext);
        inventoryCell2.transform.SetParent(inventoryCell1.OriginVisualContext);
        await MoveCellTo(inventoryCell1.transform, inventoryCell2.BeginDragSiblingIndex,
inventoryCell1.transform.GetSiblingIndex());
        await MoveCellTo(inventoryCell2.transform, inventoryCell1.BeginDragSiblingIndex,
inventoryCell2.transform.GetSiblingIndex());
    }

    public static void ShowNearestCellFor(InventoryCell inventoryCell, Transform visualContext)
    {
        int closetIndex = 0;
        Debug.Log("Count " + visualContext.transform.childCount);
        for (int i = 0; i < visualContext.transform.childCount; i++)
        {
            if (Vector3.Distance(visualContext.GetChild(i).position,
inventoryCell.transform.position) <
Vector3.Distance(visualContext.GetChild(closetIndex).position,
inventoryCell.transform.position))
            {
                closetIndex = i;
            }
        }
        var child = visualContext.GetChild(closetIndex);
        Debug.Log(closetIndex);
    }
}
}

```

Листинг класса InventoryCell:

```

using PimDeWitte.UnityMainThreadDispatcher;
using Scripts.InventoryCode.ItemResources;
using Scripts.MouseHandle;
using System;
using System.Threading.Tasks;

```

```

using TMPro;
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.UI;
using Zenject;

namespace Scripts.InventoryCode
{
    public class InventoryCell : MonoBehaviour, IDragHandler, IBeginDragHandler, IEndDragHandler
    {
        [SerializeField] Image IconElement;
        [SerializeField] TextMeshProUGUI TextElement;
        [SerializeField] TextMeshProUGUI CountTextElement;
        [SerializeField] Image SelectIconElement;
        Action _endDragEvent;
        Action<InventoryCell> _beginDragEvent;
        public InventoryItem InventoryItem { get; private set; }

        public Transform GlobalVisualContext { get; private set; }
        public Transform OriginVisualContext { get; private set; }

        public Image Icon => IconElement;
        public TextMeshProUGUI NameDisplayText => TextElement;
        public TextMeshProUGUI CountText => CountTextElement;
        public Image SelectIcon => SelectIconElement;

        public int BeginDragSiblingIndex { get; private set; }

        ItemResourceDroper _itemResourceDroper;
        MouseCursor _mouseCursor;

        [Inject]
        public void Construct(ItemResourceDroper itemResourceDroper,
            MouseCursor mouseCursor)
        {
            _itemResourceDroper = itemResourceDroper;
            _mouseCursor = mouseCursor;
        }

        private void OnDisable()
        {
        }

        private void Start()
        {
        }

        public void Initialize(Transform globalVisualContext,
            InventoryItem inventoryItem)
        {
            GlobalVisualContext = globalVisualContext;
            OriginVisualContext = transform.parent;
            InventoryItem = inventoryItem;
        }

        public void RegisterEvents(Action endDragEvent, Action<InventoryCell> beginDragEvent)
    }
}

```

```

{
    _endDragEvent = endDragEvent;
    _beginDragEvent = beginDragEvent;
}
private async void Update()
{
    InventoryItem?.RenderUI(this);
    if(InventoryItem.Consumable &&
        InventoryItem.Count < 1)
    {
        Destroy(gameObject);
        await InvokeEndDragOnItemOver();
    }
}

async Task InvokeEndDragOnItemOver()
{
    await Task.Delay(100);
    UnityMainThreadDispatcher.Instance().Enqueue(() =>
    {
        Debug.Log("InvokeEndDragOnItemOver");
        _endDragEvent?.Invoke();
    });
}

public void OnDrag(PointerEventData eventData)
{
    transform.position = Input.mousePosition;
}
public void OnBeginDrag(PointerEventData eventData)
{
    BeginDragSiblingIndex = transform.GetSiblingIndex();
    _beginDragEvent?.Invoke(this);
    transform.SetParent(GlobalVisualContext);
    _mouseCursor.ChangeCursor(CursorType.Drag);
}

public async void OnEndDrag(PointerEventData eventData)
{
    _mouseCursor.ChangeCursor(CursorType.Default);
    InventoryBase inventory;
    if (DragExtension.CheckMouseIntersectionWithContainers(eventData,
        out inventory))// если есть пересечение с инвентарем
    {
        // если это тот же инвентарь
        if (OriginVisualContext == inventory.Container)
        {
            await DragExtension.PlaceInTheNearestCellLocal(OriginVisualContext,
                this, BeginDragSiblingIndex).ContinueWith((i) =>
            {
                UnityMainThreadDispatcher.Instance().Enqueue(() =>
                {
                    _endDragEvent?.Invoke();
                });
            });
        }
    }
}

```

```

else//другой инвентарь
{
    if (inventory.IsFull())// если полон, то не перекладывать
    {
        await DragExtension.PlaceInTheNearestCellLocal(OriginVisualContext,
            this, BeginDragSiblingIndex).ContinueWith((i) =>
        {
            UnityMainThreadDispatcher.Instance().Enqueue(() =>
            {
                _endDragEvent?.Invoke();
            });
        });
    }
}
else// переложить и переподписать ячейку на события другого инвентаря
{
    DragExtension.PlaceInTheNearestCellGlobal(inventory.Container, this);
    OriginVisualContext = inventory.Container;
    _endDragEvent?.Invoke();
    inventory.RegisterDragEvents(this);
    _endDragEvent?.Invoke();
    inventory.OnDragInto(this);
    return;
}
}
}
else// нет пересечений с другим инвентарем
{
    _endDragEvent?.Invoke();
    Destroy(this.gameObject);
    _itemResourceDroper.DropByPlayer(InventoryItem);
}
_endDragEvent?.Invoke();
}
}
}
}

```

Листинг класса ItemContextData:

```

using Scripts.InventoryCode;

namespace Scripts.Inventory
{
    public class ItemContextData
    {
        public InventoryItem Item { get; private set; }
        public int Index { get; private set; }
        public string ContextName { get; private set; }
        public ItemContextData(InventoryItem inventoryItem,
            int index, string Name)
        {
            Item = inventoryItem;
            Index = index;
            ContextName = Name;
        }
    }
}

```

```

    }
}

using UnityEngine;
using Zenject;

namespace Scripts.InventoryCode
{
    public class Pack : MonoBehaviour
    {
        [SerializeField] GameObject InventoryObject;

        InputService _inputService;

        [Inject]
        public void Construct(InputService inputService)
        {
            _inputService = inputService;
        }

        public void Update()
        {
            if (_inputService.IsOpenCloseBackPack())
            {
                if (InventoryObject.activeSelf)
                {
                    InventoryObject.SetActive(false);
                }
                else
                {
                    InventoryObject.SetActive(true);
                }
            }
        }
    }
}

```

Листинг класса PlayerInventory:

```

using Scripts.FarmGameEvents;
using Scripts.Inventory;
using Scripts.InventoryCode.ItemResources;
using Scripts.SaveLoader;
using System.Collections.Generic;
using UnityEngine;
using Zenject;

namespace Scripts.InventoryCode
{
    public class PlayerInventory : MonoBehaviour
    {
        static PlayerInventory s_instance;
        [SerializeField] InventoryBase _activePackInventory;
    }
}

```

```

[SerializeField] InventoryBase _backPackInventory;
[SerializeField] private CanvasGroup _playerGroup;
GameDataState _gameDataState;

private void Awake()
{
    if (s_instance == null)
    {
        s_instance = this;
    }
}

[Inject]
public void Construct(GameDataState gameDataState)
{
    _gameDataState = gameDataState;
}

private void OnEnable()
{
    GameEvents.OnTradePanelOpenClose += OnTradePanelAction;
    GameEvents.OnExitTheGameEvent += OnExitTheGame;
}

private void OnDisable()
{
    GameEvents.OnTradePanelOpenClose -= OnTradePanelAction;
    GameEvents.OnExitTheGameEvent -= OnExitTheGame;
}

public bool TryAddItem(InventoryItem inventoryItem)
{
    if (_backPackInventory.IsFull() == false)
    {
        if (_backPackInventory.AddItem(inventoryItem))
        {
            return true;
        }
    }
    else
    {
        return false;
    }
}

else if (_activePackInventory.IsFull() == false)
{
    if (_activePackInventory.AddItem(inventoryItem))
    {
        return true;
    }
    else
    {
        return false;
    }
}

return false;
}

public bool TryPickupResource(ItemResource itemResource)
{

```



```

        if (_backPackInventory.IsFull() == false)
        {
            _backPackInventory.AddItem(itemResource.InventoryItem);
            return true;
        }
        else if (_activePackInventory.IsFull() == false)
        {
            _activePackInventory.AddItem(itemResource.InventoryItem);
            return true;
        }
        return false;
    }
    public bool IsFull()
    {
        if (!_backPackInventory.IsFull() ||
            !_activePackInventory.IsFull())
        {
            return false;
        }
        else
            return true;
    }
    public static PlayerInventory Instance()
    {
        return s_instance;
    }

    public List<InventoryItem> GetAllItems()
    {
        List<InventoryItem> inventoryItems = new List<InventoryItem>();

        inventoryItems.AddRange(_activePackInventory.GetItems());
        inventoryItems.AddRange(_backPackInventory.GetItems());

        return inventoryItems;
    }

    public List<ItemContextData> GetAllItemsContextData()
    {
        List<ItemContextData> itemContextDatas = new List<ItemContextData>();

        itemContextDatas.AddRange(_activePackInventory.GetItemsContextData());
        itemContextDatas.AddRange(_backPackInventory.GetItemsContextData());

        return itemContextDatas;
    }

    public void RemoveItem(ItemContextData itemContextData)
    {
        if (itemContextData.ContextName == _activePackInventory.Container.name)
        {
            _activePackInventory.RemoveItem(itemContextData.Index);
        }
        if (itemContextData.ContextName == _backPackInventory.Container.name)
        {
            _backPackInventory.RemoveItem(itemContextData.Index);
        }
    }

```

```

    }
    void OnExitTheGame()
    {
        var activePackItems = _activePackInventory.GetItems();
        var backPackItems = _backPackInventory.GetItems();
        _gameDataState.UpdateActivePackInventory(activePackItems);
        _gameDataState.UpdateBackPackInventory(backPackItems);
    }
    void OnTradePanelAction(bool isIgnore)
    {
        _playerGroup.blocksRaycasts = isIgnore;
    }
}
}

```

Листинг класса LevelNameValidator:

```

using System.Collections.Generic;
using System.Text.RegularExpressions;

namespace Scripts.MainMenuScripts
{
    public class LevelNameValidator
    {
        private const string c_validatePattern = @"^[^0-9][A-яA-z0-9]+$";
        Regex _validateRegex;
        public LevelNameValidator()
        {
            _validateRegex = new Regex(c_validatePattern);
        }

        public bool ValidateInputName(string name)
        {
            if(_validateRegex.IsMatch(name))
            {
                return true;
            }
            else
            {
                return false;
            }
        }
        /// <summary>
        /// true если имя существует, false - нет
        /// </summary>
        /// <param name="name"></param>
        /// <param name="names"></param>
        /// <returns></returns>
        public bool CheckForExist(string name, List<string> names)
        {
            foreach(string savedName in names)
            {
                if(savedName == name)
                {
                    return true;
                }
            }
        }
    }
}

```

```

    }
    return false;
  }
}
}

```

Листинг класса NewGameMenu:

```

using PimDeWitte.UnityMainThreadDispatcher;
using Scripts.MainMenuScripts;
using Scripts.SaveLoader;
using System.Collections.Generic;
using System.Threading.Tasks;
using UnityEngine;
using Zenject;
using UnityEngine.SceneManagement;
using Assets.Scripts;
using Scripts.FarmGameEvents;

namespace Scripts.MainMenuCode
{
    public class NewGameMenu
    {
        GameDataSaveLoader _gameDataSaveLoader;
        NewGamePanel _newGamePanel;
        List<string> _lvlNames;
        LevelNameValidator _lvlNameValidator;
        public NewGameMenu([Inject(Id = "NewGamePanel")] NewGamePanel newGamePanel)
        {
            _gameDataSaveLoader = new GameDataSaveLoader();
            _newGamePanel = newGamePanel;
            _newGamePanel.gameObject.SetActive(false);
            _lvlNameValidator = new LevelNameValidator();
        }

        public void Activate()
        {
            if (_newGamePanel.gameObject.activeSelf == false)
            {
                _newGamePanel.gameObject.SetActive(true);
            }
            _lvlNames = _gameDataSaveLoader.LoadWorldNamesJson();
        }

        public async void OnCreate()
        {
            string input = _newGamePanel.InputText.text;
            _lvlNames = _gameDataSaveLoader.LoadWorldNamesJson();
            if (_lvlNames != null &&
                _lvlNames.Count > 0)
            {
                if (_lvlNameValidator.ValidateInputName(input) &&
                    !_lvlNameValidator.CheckForExist(input, _lvlNames))
                {
                    StartLevel(input);
                }
            }
        }
    }
}

```

```

        return;
    }
    else
    {
        await FadeText();
    }
}
else if (_lvlNameValidator.ValidateInputName(input))
{
    StartLevel(input);
}
else
{
    await FadeText();
}
}
}
async Task FadeText()
{
    _newGamePanel.InfoText.text = "Название не должно повторяться и " +
        "начинаться с цифры!";
    _newGamePanel.InfoText.color = Color.red;
    await Task.Run(() =>
    {
        UnityMainThreadDispatcher.Instance().Enqueue(async () =>
        {
            Color startColor = _newGamePanel.InfoText.color;
            Color targetColor = new Color(startColor.r, startColor.g, startColor.b,
                0);
            float fadeDuration = 10;
            float elapsedTime = 0f;

            while (elapsedTime < fadeDuration)
            {
                float t = elapsedTime / fadeDuration;
                _newGamePanel.InfoText.color = Color.Lerp(startColor, targetColor, t);
                elapsedTime += Time.deltaTime;
                await Task.Yield();
            }

            _newGamePanel.InfoText.color = targetColor;
        });
    });
}
}
void StartLevel(string name)
{
    GameDataState gameDataState = new(name);
    LoadedData.Instance().InitializeGameStateData(gameDataState, true);
    List<string> names = _gameDataSaveLoader.LoadWorldNamesJson();
    if(names == null)
    {
        names = new List<string>();
    }
    names.Add(name);
    _gameDataSaveLoader.SaveWorldNamesJson(names);
}

```

```

        GameEvents.InvokeOnSaveSettingsEvent();
        SceneManager.LoadScene(GameConfiguration.FarmSceneName);
    }
    public void OnBack()
    {
        _newGamePanel.gameObject.SetActive(false);
    }
}
}

```

Листинг класса NewGamePanel:

```

using TMPPro;
using UnityEngine;

namespace Scripts.MainMenuScripts
{
    public class NewGamePanel : MonoBehaviour
    {
        [SerializeField] private TMP_InputField TMP_InputField;
        public TMP_InputField InputText => TMP_InputField;
        [SerializeField] private TextMeshProUGUI InfoTextField;
        public TextMeshProUGUI InfoText => InfoTextField;
    }
}

```

Листинг класса SettingsPanel:

```

using Scripts.FarmGameEvents;
using Scripts.InteractableObjects;
using Scripts.SaveLoader;
using System;
using System.Collections.Generic;
using TMPPro;
using UnityEngine;
using UnityEngine.InputSystem;
using UnityEngine.UI;
using Zenject;
using DropDownOption = TMPPro.TMP_Dropdown.OptionData;

namespace Scripts.MainMenuCode
{
    public class SettingsPanel : MonoBehaviour
    {
        [SerializeField] private TMP_Dropdown _dropDownResolution;
        [SerializeField] private Toggle _toggleFullScreen;
        [SerializeField] private Slider _sliderMusic;
        [SerializeField] private Slider _sliderMain;
        [SerializeField] private Slider _sliderSFX;
        [SerializeField] private Transform _keyBoardBindingsContent;
        [SerializeField] private TextMeshProUGUI _interactableRebindText;
        [SerializeField] private CanvasGroup _settingsCanvasGroup;

        public TMP_Dropdown DropDownResolution => _dropDownResolution;
    }
}

```

```

public Toggle ToggleFullScreen => _toggleFullScreen;
public Slider SliderMusic => _sliderMusic;
public Slider SliderMain => _sliderMain;
public Slider SliderSFX => _sliderSFX;
public Transform KeyboardBindingsContent => _keyboardBindingsContent;
public TextMeshProUGUI InteractableRebindText => _interactableRebindText;

List<Resolution> _availableResolutions;

InputService _inputService;

KeyBindingPanelFactory _keyBindingPanelFactory;

public InputActionAsset InputActionsAsset => _inputService.GetInputActionAsset();

[Inject]
public void Construct(FactoriesProvider factoriesProvider)
{
    _keyBindingPanelFactory = (KeyBindingPanelFactory)factoriesProvider
        .GetFactory<KeyBindingPanelFactory>();
}
private void OnEnable()
{
    GameEvents.OnPerformInteractiveRebindEvent += OnRebind;
}
private void OnDisable()
{
    GameEvents.OnPerformInteractiveRebindEvent -= OnRebind;
}
public void InitializeDropDownResolution()
{
    _dropDownResolution.options.Clear();

    _availableResolutions = new List<Resolution>();

    int currentResolutionIndex = 0;
    int i = 0;
    Resolution currentResolution = Screen.currentResolution;
    foreach (var resolution in Screen.resolutions)
    {
        _availableResolutions.Add(resolution);

        DropDownOption optionData = new();
        string resolutionText = $"{resolution.width}x{resolution.height}";
        optionData.text = resolutionText;
        _dropDownResolution.options.Add(optionData);

        if (currentResolution.width == resolution.width &&
            currentResolution.height == resolution.height)
        {
            currentResolutionIndex = i;
        }
        i++;
    }

    _dropDownResolution.value = currentResolutionIndex;

```

```

TMP_Dropdown.DropdownEvent evnt = new TMP_Dropdown.DropdownEvent();
evnt.AddListener((call) =>
{
    var res = _availableResolutions[call];
    Screen.SetResolution(res.width, res.height, _toggleFullScreen.isOn);
});
_dropDownResolution.onValueChanged = evnt;

}
public void InitializeToggleFullScreen()
{
    _toggleFullScreen.isOn = Screen.fullScreen;
    Toggle.ToggleEvent ev = new Toggle.ToggleEvent();
    ev.AddListener((call) =>
    {
        Screen.fullScreen = call;
    });
    _toggleFullScreen.onValueChanged = ev;
}
public void InitializeBindingsKeys(InputService inputService)
{
    InputActionAsset asset = inputService.GetInputActionAsset();

    foreach (var map in asset.actionMaps)
    {
        foreach (InputAction action in map.actions)
        {
            if(action.expectedControlType == "Vector2")
            {
                for (int i = 0; i < action.controls.Count; i++)
                {
                    var control = action.controls[i];

                    var binding = action.GetBindingForControl(control);

                    KeyBindingPanel keyBindingPanel = _keyBindingPanelFactory.Create(KeyBoardBindingsContent);

                    keyBindingPanel.Initialize(action, control);

                    keyBindingPanel.ActionNameText.text = binding.Value.name;
                    keyBindingPanel.KeyButtonText.text = control.displayName;
                }
            }
            else
            {
                KeyBindingPanel keyBindingPanel = _keyBindingPanelFactory.Create(KeyBoardBindingsContent);

                keyBindingPanel.ActionNameText.text = action.name;
                keyBindingPanel.KeyButtonText.text = action.controls[0].displayName;

                keyBindingPanel.Initialize(action, action.controls[0]);
            }
        }
    }
}
void OnRebind(string text, bool activeSelf)

```

```

    {
        InteractableRebindText.text = text;
        _settingsCanvasGroup.blocksRaycasts = !activeSelf;
        InteractableRebindText.gameObject.transform.parent.
            transform.gameObject.SetActive(activeSelf);
    }
}
}

```

Листинг класса LoadMenu:

```

using Scripts.SaveLoader;
using System.Collections.Generic;
using UnityEngine;
using Zenject;

namespace Scripts.MainMenuCode
{
    public class LoadMenu
    {
        IGameStatePanelFactory _gameStatePanelFactory;
        GameObject _loadLevelsPanel;
        GameDataSaveLoader _gameDataSaveLoader;
        Transform _content;
        public LoadMenu(IGameStatePanelFactory gameStatePanelFactory,
            [Inject(Id = "LoadMenuPanel")] GameObject loadLevelsPanel,
            [Inject(Id = "LoadMenuContent")] Transform content)

        {
            _gameStatePanelFactory = gameStatePanelFactory;
            _loadLevelsPanel = loadLevelsPanel;
            _gameDataSaveLoader = new GameDataSaveLoader();
            _content = content;
            _loadLevelsPanel.gameObject.SetActive(false);
        }

        public void Activate()
        {
            if(_loadLevelsPanel.activeSelf == false)
            {
                _loadLevelsPanel.SetActive(true);
            }

            List<string> lvlNames = _gameDataSaveLoader.LoadWorldNamesJson();

            // _gameStatePanelFactory.Create(GameConfiguration.SaveEditorGameStateName,
            //     _content);

            if (lvlNames != null &&
                lvlNames.Count > 0)
            {
                CreateStatePanels(lvlNames);
            }
        }
        void CreateStatePanels(List<string> names)
    }
}

```



```

{
    foreach(string name in names)
    {

        GameStatePanel panel = _gameStatePanelFactory.Create(name, _content);
        panel.MoneyText.text
            = _gameDataSaveLoader.LoadGameState(name).PlayerData.Money.ToString();
    }
}

public void OnBack()
{
    foreach (Transform state in _content)
    {
        GameObject.Destroy(state.gameObject);
    }
    _loadLevelsPanel.gameObject.SetActive(false);
}
}
}

```

Листинг класса GameStatePanel:

```

using Assets.Scripts;
using Scripts.FarmGameEvents;
using Scripts.SaveLoader;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.SceneManagement;

namespace Scripts.MainMenuCode
{
    public class GameStatePanel : MonoBehaviour
    {
        [SerializeField] private TextMeshProUGUI _nameTextField;
        [SerializeField] private TextMeshProUGUI _moneyTextField;

        public TextMeshProUGUI MoneyText => _moneyTextField;
        public TextMeshProUGUI NameText => _nameTextField;

        GameDataSaveLoader _gameDataSaveLoader;

        private void Start()
        {
            _gameDataSaveLoader = new GameDataSaveLoader();
        }

        public void Load()
        {
            GameDataState gameDataState =
                _gameDataSaveLoader.LoadGameState(NameText.text);
            LoadedData.Instance().InitializeGameStateData(gameDataState, false);
            GameEvents.InvokeOnSaveSettingsEvent();
            SceneManager.LoadScene(GameConfiguration.FarmSceneName);
        }
    }
}

```

```

    }
    public void Delete()
    {
        if(PlayerPrefs.HasKey(GameConfiguration.SaveLevelNamesKeyName))
        {
            List<string> names = _gameDataSaveLoader.LoadWorldNamesJson();
            foreach(string name in names)
            {
                if(name == NameText.text)
                {
                    names.Remove(name);
                    _gameDataSaveLoader.SaveWorldNamesJson(names);
                    if(PlayerPrefs.HasKey(name))
                    {
                        PlayerPrefs.DeleteKey(name);
                    }
                    Destroy(gameObject);
                    return;
                }
            }
        }
    }
}

```

Листинг класса IGameStatePanelFactory:

```

using UnityEngine;

namespace Scripts.MainMenuCode
{
    public interface IGameStatePanelFactory
    {
        GameStatePanel Create(string stateName, Transform content);
    }
}

```

Листинг класса StatePanelFactory:

```

using UnityEngine;
using Zenject;

namespace Scripts.MainMenuCode
{
    public class StatePanelFactory : IGameStatePanelFactory
    {
        GameStatePanel _template;
        public StatePanelFactory(
            [Inject(Id = "StatePanelTemplate")] GameStatePanel gameStatePanel)
        {
            _template = gameStatePanel;
        }
        public GameStatePanel Create(string stateName, Transform content)

```

```

    {
        var panel = GameObject.Instantiate(_template,
            content);
        panel.NameText.text = stateName;
        return panel;
    }
}

```

Листинг класса BGMove:

```

using System;
using UnityEngine;

public class BGMove : MonoBehaviour
{
    [Range(0f, 5f)]
    [SerializeField] private float BgSpeed;

    Vector2 _meshOffset;
    MeshRenderer _meshRenderer;

    void Start()
    {
        var height = Camera.main.orthographicSize * 2f;
        var width = height * Screen.width / Screen.height;

        transform.localScale = new Vector3 (width, height, 0);

        _meshRenderer = GetComponent<MeshRenderer>();
        _meshOffset = _meshRenderer.sharedMaterial.mainTextureOffset;
    }

    void Update()
    {
        _meshOffset.x += Time.deltaTime * BgSpeed;
        _meshRenderer.sharedMaterial.SetTextureOffset("_MainTex", _meshOffset);
    }
}

```

Листинг класса LoadedData:

```

using Scripts.SaveLoader;
using UnityEngine;

namespace Scripts.MainMenuCode
{
    public class LoadedData : MonoBehaviour
    {
        static LoadedData _dataInstance;

        static LoadedData()
        {
            IsGameStateDefault = true;
        }
    }
}

```

```

    }

    private void Awake()
    {
        if (_dataInstance == null)
        {
            _dataInstance = this;
        }
        DontDestroyOnLoad(this);
    }
    public static bool IsGameStateDefault { get; private set; }
    public GameState GameState { get; private set; }
    public static bool IsSettingsDataDefault { get; private set; }
    public SettingsData SettingsData { get; private set; }
    public void InitializeGameStateData(GameState gameState, bool isDefault)
    {
        GameState = gameState;
        IsGameStateDefault = isDefault;
    }
    public void InitializeSettingsData(SettingsData settingsData, bool isDefault)
    {
        SettingsData = settingsData;
        IsSettingsDataDefault = isDefault;
    }
    public static LoadedData Instance()
    {
        return _dataInstance;
    }
}
}

```

Листинг класса MainMenu:

```

using Scripts.FarmGameEvents;
using Scripts.MouseHandle;
using Scripts.SaveLoader;
using UnityEngine;
using Zenject;

namespace Scripts.MainMenuCode
{
    public class MainMenu : MonoBehaviour
    {
        private const string GameSceneName = "FarmScene";
        private LoadMenu _loadMenu;
        private NewGameMenu _newGameMenu;
        private GameObject _menuObject;
        private SettingsMenu _settingsMenu;
        private MouseCursor _mouseCursor;
        [Inject]
        public void Construct(LoadMenu loadMenu,
            NewGameMenu newGameMenu,
            SettingsMenu settingsMenu,
            [Inject(Id = "MainMenuObject")] GameObject menuObject,
            MouseCursor mouseCursor)
        {
        }
    }
}

```

```

{
    _mouseCursor = mouseCursor;
    _loadMenu = loadMenu;
    _newGameMenu = newGameMenu;
    _settingsMenu = settingsMenu;
    _menuObject = menuObject;
}
private void OnEnable()
{
    GameEvents.OnSaveSettingsEvent += _settingsMenu.Save;
}
private void OnDisable()
{
    GameEvents.OnSaveSettingsEvent -= _settingsMenu.Save;
}
private void Start()
{
    SettingsDataSaveLoader settingsDataSaveLoader = new();
    SettingsData settingsData = null;
    if (settingsDataSaveLoader.IsDefault())
    {
        settingsData = new SettingsData();
        LoadedData.Instance().InitializeSettingsData(settingsData, true);
    }
    else
    {
        settingsData = settingsDataSaveLoader.Load();
        LoadedData.Instance().InitializeSettingsData(settingsData, false);
    }
    _menuObject.SetActive(true);
    if (LoadedData.IsSettingsDataDefault == false)
    {
        _settingsMenu.Load(settingsData);
    }
    _settingsMenu.InitializeKeyBindings();
}
private void Update()
{
    if (Input.GetMouseButtonDown(0))
    {
        _mouseCursor.ImitateClick();
    }
}
public void NewGame()
{
    _menuObject.SetActive(false);
    _newGameMenu.Activate();
}
public void LoadGame()
{
    _menuObject.SetActive(false);
    _loadMenu.Activate();
}
public void Settings()
{
    _settingsMenu.Activate();
}

```

```

        _menuObject.SetActive(false);
    }

    public void WorkSpace()
    {
        _settingsMenu.Save();
        Application.Quit();
    }

    public void OnBack()
    {
        _loadMenu.OnBack();
        _menuObject.SetActive(true);
    }
    public void Create()
    {
        _newGameMenu.OnCreate();
    }
    public void BackFromCreate()
    {
        _newGameMenu.OnBack();
        _menuObject.SetActive(true);
    }
    public void BackFromSettings()
    {
        _settingsMenu.Back();
        _menuObject.SetActive(true);
    }
}
}
}

```

Листинг класса MenuSound:

```

using Scripts.MainMenuCode;
using UnityEngine;
using UnityEngine.Audio;
using Zenject;

namespace Scripts.MainMenuScripts
{
    public class MenuSound : MonoBehaviour
    {
        [SerializeField] private AudioSource _menuAudioSource;
        [SerializeField] private AudioMixer _mixer;

        SettingsMenu _settingsMenu;

        [Inject]
        public void Construct(SettingsMenu settingsMenu)
        {
            _settingsMenu = settingsMenu;
        }
        private void OnEnable()
        {
            _settingsMenu.AddListenerAtMusicVolume(OnMusicValueChange);
        }
    }
}

```

```

        _settingsMenu.AddListenerAtMainVolume(OnMainValueChange);
        _settingsMenu.AddListenerAtSFXVolume(OnSFXValueChange);
    }
    private void OnDisable()
    {
        _settingsMenu.RemoveListenerAtMusicVolume(OnMusicValueChange);
        _settingsMenu.RemoveListenerAtMainVolume(OnMainValueChange);
        _settingsMenu.RemoveListenerAtSFXVolume(OnSFXValueChange);
    }

    void OnMusicValueChange(float value)
    {
        _mixer.SetFloat("BGMVolume", Mathf.Log10(Mathf.Max(0.0001f, value)) * 30f);
    }
    void OnMainValueChange(float value)
    {
        _mixer.SetFloat("MainVolume", Mathf.Log10(Mathf.Max(0.0001f, value)) * 30f);
    }

    void OnSFXValueChange(float value)
    {
        _mixer.SetFloat("SFXVolume", Mathf.Log10(Mathf.Max(0.0001f, value)) * 30f);
    }
}

```

Листинг класса **MouseCursor**:

```

using System.Threading.Tasks;
using UnityEngine;

namespace Scripts.MouseHandle
{
    public enum CursorType
    {
        Default,
        Drag,
        Menu
    }
    public class MouseCursor
    {
        MouseIconDatabase _mouseIconDatabase;
        CursorType _lastType;
        CursorType _currentType;
        public MouseCursor(MouseIconDatabase mouseIconDatabase)
        {
            _mouseIconDatabase = mouseIconDatabase;
            _mouseIconDatabase.Init();
        }

        public void ChangeCursor(CursorType cursorType)
        {
            _lastType = _currentType;

```

```

        Texture2D texture2D = _mouseIconDatabase.GetIconByType(cursorType);
        Cursor.SetCursor(texture2D, Vector2.zero, CursorMode.Auto);
        _currentType = cursorType;
    }

    public async void ImitateClick()
    {
        ChangeCursor(CursorType.Drag);
        await Task.Delay(100);
        ToDefault();
    }
    void ToDefault()
    {
        ChangeCursor(_lastType);
    }
}
}

```

Листинг класса MouseIconDatabase:

```

using Scripts.SO.MouseSO;
using System.Collections.Generic;
using UnityEngine;

namespace Scripts.MouseHandle
{
    [CreateAssetMenu(fileName = "MouseIconsData", menuName = "SO/Mouse/MouseIconsDatabase")]
    public class MouseIconDatabase : ScriptableObject
    {
        public List<MouseIconSO> Data;

        public Dictionary<CursorType, MouseIconSO> _icons;

        public void Init()
        {
            _icons = new Dictionary<CursorType, MouseIconSO>();

            if (Data.Count > 0)
            {
                foreach (MouseIconSO icon in Data)
                {
                    if (_icons.ContainsKey(icon.CursorType) == false)
                    {
                        _icons.Add(icon.CursorType, icon);
                    }
                }
            }
        }

        public Texture2D GetIconByType(CursorType type)
        {
            if (_icons.ContainsKey(type))
            {
                return _icons[type].CursorIcon;
            }
        }
    }
}

```



```

    }
    else
    {
        return default;
    }
}
}
}

```

Листинг класса NextLevel:

```

using UnityEngine;
using UnityEngine.SceneManagement;
using Assets.Scripts;
using Scripts.MainMenuCode;
using Scripts.SaveLoader;
using Zenject;
using Scripts.FarmGameEvents;

namespace Scripts.NextLevel
{
    public class NextLevel : MonoBehaviour
    {
        SettingsMenu _settingsMenu;
        GameState _gameDataState;

        [Inject]
        public void Construct(SettingsMenu settingsMenu,
            GameState gameDataState)
        {
            _settingsMenu = settingsMenu;
            _gameDataState = gameDataState;
        }

        public void OnView()
        {
            GameEvents.InvokeExitTheGameEvent();
            LoadedData.Instance().InitializeGameStateData(_gameDataState, false);
            _settingsMenu.Save();
            SceneManager.LoadScene(GameConfiguration.NextLevelName);
        }
    }
}

```

Листинг класса NextLevelTrigger:

```

using UnityEngine;

namespace Scripts.NextLevel
{
    public class NextLevelTrigger : MonoBehaviour
    {
        [SerializeField] private GameObject _nextLevelPanel;
    }
}

```

```

private void OnTriggerEnter2D(Collider2D collision)
{
    if(collision.gameObject.tag == "Player")
    {
        _nextLevelPanel.SetActive(true);
    }
}
private void OnTriggerExit2D(Collider2D collision)
{
    if (collision.gameObject.tag == "Player")
    {
        _nextLevelPanel.SetActive(false);
    }
}
}
}

```

Листинг класса PreviousLevel:

```

using UnityEngine;
using UnityEngine.SceneManagement;
using Assets.Scripts;

namespace Scripts.NextLevel
{
    public class PreviousLevel : MonoBehaviour
    {
        public void OnAction()
        {
            SceneManager.LoadScene(GameConfiguration.FarmSceneName);
        }
    }
}

```

Листинг класса PlacementItem:

```

using UnityEngine;
using Zenject;
using Assets.Scripts.Placement;
using Scripts.SaveLoader;
using Scripts.FarmGameEvents;

namespace Scripts.PlacementCode
{
    public abstract class PlacementItem : MonoBehaviour, ISaveLoadPlacementItem
    {
        [SerializeField] private bool _isDefaultStateObject;
        protected ItemPlacementMap PlacementMap;
        protected GameState GameDataState;
        public Vector3Int PlacePosition { get; private set; }
        protected bool _isInitialized;
        protected PlacementItemData _data;

        [Inject]
    }
}

```

```

public virtual void ConstructItem(
    PlacementMapsContainer placementMapsContainer,
    GameDataState gameDataState)
{
    PlacementMap = placementMapsContainer.ItemPlacementMap;
    GameDataState = gameDataState;
}

public void OnEnable()
{
    GameEvents.OnExitTheGameEvent += OnExitTheGame;
}
private void OnDisable()
{
    GameEvents.OnExitTheGameEvent -= OnExitTheGame;
}
protected virtual void Start()
{
    PlacePosition = PlacementMap
        .Vector3ConvertToVector3Int(gameObject.transform.position);
    PlacementMap.PlaceObjectOnCell(gameObject, PlacePosition);
    OccupyCells();
}
private void OccupyCells()
{
    if (this is IOccupyingSeveralCells)
    {
        IOccupyingSeveralCells obj =
            (this as IOccupyingSeveralCells);
        PlacementMap.AddPositions(obj);
        return;
    }
    if (this is IOccupyingOneCell)
    {
        IOccupyingOneCell obj =
            (this as IOccupyingOneCell);
        PlacementMap.AddPosition(obj);
    }
}
public virtual void OnExitTheGame()
{
    SaveData();
}

public virtual void SaveData()
{
    {
        _data = GetData();
        GameDataState.AddItemData(_data);
    }
}
data.SetPosition(PlacePosition);
return data;

```

```

    }
}
}

```

Листинг класса ItemPlacementMap:

```

using Assets.Scripts.Placement;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Tilemaps;

namespace Scripts.PlacementCode
{
    public class ItemPlacementMap
    {
        public List<Vector3Int> OccupiedPositions { get; private set; }
        protected Tilemap TileMap;
        public string Name;

        public ItemPlacementMap(Tilemap tileMap)
        {
            OccupiedPositions = new List<Vector3Int>();
            TileMap = tileMap;
        }

        public virtual bool IsOccupied(Vector3Int position)
        {
            return OccupiedPositions.Contains(position);
        }

        public virtual bool IsOccupied(List<Vector3Int> positions)
        {
            foreach (var position in positions)
            {
                if (OccupiedPositions.Contains(position))
                {
                    return true;
                }
            }
            return false;
        }

        public void AddPosition(IOccupyingOneCell obj)
        {
            OccupiedPositions.Add(obj.GetOccupyingCell());
        }

        public void AddPosition(Vector3Int position)
        {
            OccupiedPositions.Add(position);
        }

        public void AddPositions(IOccupyingSeveralCells obj)
        {
            OccupiedPositions.AddRange(obj.GetOccupyingCells());
        }

        public void RemovePosition(IOccupyingOneCell obj)
        {

```

```

        OccupiedPositions.Remove(obj.GetOccupyingCell());
    }
    public void RemovePosition(Vector3Int position)
    {
        OccupiedPositions.Remove(position);
    }
    public void RemovePositions(IOccupyingSeveralCells obj)
    {
        List<Vector3Int> positions = obj.GetOccupyingCells();
        OccupiedPositions.RemoveAll(x => positions.Contains(x));
    }
    public virtual void PlaceObjectOnCell(GameObject gameObject, Vector3Int position)
    {
        Vector3 objPosition = TileMap.GetCellCenterWorld(position);
        //Vector3 objPosition = TileMap.CellToWorld(position);
        gameObject.transform.position = objPosition;
    }
    public Vector3 GetCellCenterWorld(Vector3Int position)
    {
        return TileMap.GetCellCenterWorld(position);
    }
    public Vector3Int Vector3ConvertToVector3Int(Vector3 position)
    {
        Vector3Int pos3 = TileMap.WorldToCell(position);
        return pos3;
    }
    public Vector3 Vector3IntConvertToVector3(Vector3Int position)
    {
        Vector3 pos3 = TileMap.CellToWorld(position);
        return pos3;
    }
}
}

```

Листинг класса PlacedCropData:

```

namespace Scripts.PlacementCode
{
    public class PlacedCropData
    {
        public string CropId;
        public int Stage;
        public float GrowthRatio;
        public float GrowthTimer;
        public int HarvestCount;
        public float DyingTimer;
    }
}

```

Листинг класса PlacementData:

```

using System.Collections.Generic;

namespace Scripts.PlacementCode

```

```

{
    public class PlacementData
    {
        public List<TilePlacementData> CropsTilePlacementDatas;
        public List<PlacedCropData> PlacedCropDatas;
        public List<TilePlacementData> GroundTilePlacementDatas;
        public List<PlacementService.GroundData> GroupDatas;

        public PlacementData()
        {
            CropsTilePlacementDatas = new List<TilePlacementData>();
            PlacedCropDatas = new List<PlacedCropData>();
            GroundTilePlacementDatas = new List<TilePlacementData>();
            GroupDatas = new List<PlacementService.GroundData>();
        }
    }
}

```

Листинг класса TilePlacementData:

```

using UnityEngine;

namespace Scripts.PlacementCode
{
    public class TilePlacementData
    {
        public int X;
        public int Y;
        public int Z;
        public void SetPosition(Vector3Int position)
        {
            X = position.x;
            Y = position.y;
            Z = position.z;
        }
        public Vector3Int GetPosition()
        {
            return new Vector3Int(X, Y, Z);
        }
    }
}

```

Листинг класса InteractMarker:

```

using UnityEngine;

namespace Scripts.PlacementCode
{
    public class InteractMarker : MonoBehaviour
    {
        [SerializeField]
        private Color _activeColor = Color.white;
        [SerializeField]
        private Color _inactiveColor = Color.gray;
    }
}

```

```

private SpriteRenderer _renderer;

private void Awake()
{
    _renderer = GetComponent<SpriteRenderer>();
}

// Start is called before the first frame update
public void Activate()
{
    Show();
    _renderer.color = _activeColor;
}

public void Deactivate()
{
    Show();
    _renderer.color = _inactiveColor;
}

public void Hide()
{
    gameObject.SetActive(false);
}

public void Show()
{
    gameObject.SetActive(true);
}
}

```

Листинг класса IOccupyingOneCell:

```

using UnityEngine;

namespace Assets.Scripts.Placement
{
    public interface IOccupyingOneCell
    {
        Vector3Int GetOccupyingCell();
    }
}

```

Листинг класса IOccupyingSeveralCells:

```

using System.Collections.Generic;
using UnityEngine;

namespace Assets.Scripts.Placement
{
    public interface IOccupyingSeveralCells : IOccupyingCells
    {

```

```

        List<Vector3Int> GetOccupyingCells();
    }
}

```

Листинг класса MarkerController:

```

using Scripts.PlayerCode;
using UnityEngine;
using Zenject;

namespace Scripts.PlacementCode
{
    [RequireComponent(typeof(Player))]
    public class MarkerController : MonoBehaviour
    {
        [SerializeField] private SpriteRenderer _interactMarketSprite;
        private Grid _grid;

        Player _player;
        public InteractMarker InteractMarker { get; private set; }
        public Vector3Int CurrentTarget { get; private set; }

        [Inject]
        public void Construct(Grid grid)
        {
            _grid = grid;
        }

        private void Start()
        {
            InteractMarker = _interactMarketSprite.GetComponent<InteractMarker>();
            _player = GetComponent<Player>();
        }

        private void Update()
        {
            var currentCell = _grid.WorldToCell(transform.position);
            var pointedCell = _grid.WorldToCell(_player.CurrentWorldMousePos);

            currentCell.z = 0;
            pointedCell.z = 0;

            var toTarget = pointedCell - currentCell;

            if (Mathf.Abs(toTarget.x) > 1)
            {
                toTarget.x = (int)Mathf.Sign(toTarget.x);
            }

            if (Mathf.Abs(toTarget.y) > 1)
            {
                toTarget.y = (int)Mathf.Sign(toTarget.y);
            }

            CurrentTarget = currentCell + toTarget;
        }
    }
}

```



```

        InteractMarker.transform.position =
            _grid.GetCellCenterWorld(CurrentTarget);
    }
}
}

```

Листинг класса PlacementService:

```

using Scripts.InventoryCode;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Tilemaps;
using UnityEngine.VFX;
using Zenject;
using Scripts.SaveLoader;
using Crop = Scripts.InventoryCode.Crop;
using Scripts.FarmGameEvents;

namespace Scripts.PlacementCode
{
    public class PlacementService : MonoBehaviour
    {
        static PlacementService s_instance;
        Grid _grid;
        CropDataBase _cropDatabase;
        GameState _gameDataState;
        [Inject]
        public void Construct(Grid grid,
            CropDataBase cropDataBase,
            GameState gameDataState
        )
        {
            _grid = grid;
            _gameDataState = gameDataState;
            _cropDatabase = cropDataBase;
        }

        public Tilemap GroundTilemap;
        public Tilemap CropTilemap;

        [Header("Watering")]
        public Tilemap WaterTilemap;
        public TileBase WateredTile;

        [Header("Tilling")]
        public TileBase TilleableTile;
        public TileBase TilledTile;
        public VisualEffect TillingEffectPrefab;

        private Dictionary<Crop, List<VisualEffect>> harvestEffectPool = new();
        private List<VisualEffect> _tillingEffectPool = new();
    }
}

```

```

public class GroundData
{
    public const float WaterDuration = 60 * 1.0f;

    public float WaterTimer;
}
public class CropData
{

    public Crop GrowingCrop = null;
    public int CurrentGrowthStage = 0;

    public float GrowthRatio = 0.0f;
    public float GrowthTimer = 0.0f;

    public int HarvestCount = 0;

    public float DyingTimer { get; set; }
    public bool HarvestDone => HarvestCount == GrowingCrop.NumberOfHarvest;

    public Crop Harvest()
    {
        var crop = GrowingCrop;

        HarvestCount += 1;

        CurrentGrowthStage = GrowingCrop.StageAfterHarvest;
        GrowthRatio = CurrentGrowthStage / (float)GrowingCrop.GrowthStagesTiles.Length;
        GrowthTimer = GrowingCrop.GrowthTime * GrowthRatio;

        return crop;
    }

    public void Save(PlacedCropData placedCropData)
    {
        placedCropData.Stage = CurrentGrowthStage;
        placedCropData.CropId = GrowingCrop.UniqueName;
        placedCropData.DyingTimer = DyingTimer;
        placedCropData.GrowthRatio = GrowthRatio;
        placedCropData.GrowthTimer = GrowthTimer;
        placedCropData.HarvestCount = HarvestCount;
    }

    public void Load(PlacedCropData placedCropData,
        CropDataBase cropDataBase)
    {
        CurrentGrowthStage = placedCropData.Stage;
        GrowingCrop = cropDataBase.GetItemByName(placedCropData.CropId);
        DyingTimer = placedCropData.DyingTimer;
        GrowthRatio = placedCropData.GrowthRatio;
        GrowthTimer = placedCropData.GrowthTimer;
        HarvestCount = placedCropData.HarvestCount;
    }
}

```

```

private Dictionary<Vector3Int, GroundData> _groundData = new();
private Dictionary<Vector3Int, CropData> _cropData = new();

private void Awake()
{
    if(s_instance == null)
    {
        s_instance = this;
    }
}
private void OnEnable()
{
    GameEvents.OnExitTheGameEvent += OnExitTheGame;
}
private void OnDisable()
{
    GameEvents.OnExitTheGameEvent -= OnExitTheGame;
}

private void Start()
{
    for (int i = 0; i < 4; ++i)
    {
        var effect = Instantiate(TillingEffectPrefab);
        effect.gameObject.SetActive(true);
        effect.Stop();
        _tillingEffectPool.Add(effect);
    }
}

private void Update()
{
    foreach (var (cell, groundData) in _groundData)
    {
        if (groundData.WaterTimer > 0.0f)
        {
            groundData.WaterTimer -= Time.deltaTime;

            if (groundData.WaterTimer <= 0.0f)
            {
                WaterTilemap.SetTile(cell, null);
                //GroundTilemap.SetColor(cell, Color.white);
            }
        }

        if (_cropData.TryGetValue(cell, out var cropData))
        {
            if (groundData.WaterTimer <= 0.0f)
            {
                cropData.DyingTimer += Time.deltaTime;
                if (cropData.DyingTimer > cropData.GrowingCrop.DryDeathTimer)
                {
                    _cropData.Remove(cell);
                    UpdateCropVisual(cell);
                }
            }
        }
    }
}

```

```

        else
        {
            cropData.DyingTimer = 0.0f;
            cropData.GrowthTimer = Mathf.Clamp(cropData.GrowthTimer + Time.deltaTime, 0.0f,
            cropData.GrowingCrop.GrowthTime);
            cropData.GrowthRatio = cropData.GrowthTimer / cropData.GrowingCrop.GrowthTime;
            int growthStage = cropData.GrowingCrop.GetGrowthStage(cropData.GrowthRatio);

            if (growthStage != cropData.CurrentGrowthStage)
            {
                cropData.CurrentGrowthStage = growthStage;
                UpdateCropVisual(cell);
            }
        }
    }
}

public bool IsTillable(Vector3Int target)
{
    return GroundTilemap.GetTile(target) == TilleableTile;
}

public bool IsPlantable(Vector3Int target)
{
    return IsTilled(target) && !_cropData.ContainsKey(target);
}

public bool IsTilled(Vector3Int target)
{
    return _groundData.ContainsKey(target);
}

public void TillAt(Vector3Int target)
{
    if (IsTilled(target))
        return;

    GroundTilemap.SetTile(target, TilledTile);
    _groundData.Add(target, new GroundData());

    var inst = _tillingEffectPool[0];
    _tillingEffectPool.RemoveAt(0);
    _tillingEffectPool.Add(inst);

    inst.gameObject.transform.position = _grid.GetCellCenterWorld(target);

    inst.Stop();
    inst.Play();
}

public void PlantAt(Vector3Int target, Crop cropToPlant)
{
    var cropData = new CropData();

    cropData.GrowingCrop = cropToPlant;
    cropData.GrowthTimer = 0.0f;

```

```

cropData.CurrentGrowthStage = 0;

_cropData.Add(target, cropData);

UpdateCropVisual(target);

if (!harvestEffectPool.ContainsKey(cropToPlant))
{
    InitHarvestEffect(cropToPlant);
}
}
public Crop HarvestAt(Vector3Int target)
{
    _cropData.TryGetValue(target, out var data);

    if (data == null || !Mathf.Approximately(data.GrowthRatio, 1.0f)) return null;

    var produce = data.Harvest();

    if (data.HarvestDone)
    {
        _cropData.Remove(target);
    }

    UpdateCropVisual(target);

    var effect = harvestEffectPool[data.GrowingCrop][0];
    effect.transform.position = _grid.GetCellCenterWorld(target);
    harvestEffectPool[data.GrowingCrop].RemoveAt(0);
    harvestEffectPool[data.GrowingCrop].Add(effect);
    effect.Play();

    return produce;
}

public void WaterAt(Vector3Int target)
{
    var groundData = _groundData[target];

    groundData.WaterTimer = GroundData.WaterDuration;

    WaterTilemap.SetTile(target, WateredTile);
    //GroundTilemap.SetColor(target, WateredTiledColorTint);
}

void UpdateCropVisual(Vector3Int target)
{
    if (!_cropData.TryGetValue(target, out var data))
    {
        CropTilemap.SetTile(target, null);
    }
    else
    {
        CropTilemap.SetTile(target, data.GrowingCrop.GrowthStagesTiles[data.CurrentGrowthStage]);
    }
}

public CropData GetCropDataAt(Vector3Int target)

```

```

{
    _cropData.TryGetValue(target, out var data);
    return data;
}
public void InitHarvestEffect(Crop crop)
{
    harvestEffectPool[crop] = new List<VisualEffect>();
    for (int i = 0; i < 4; ++i)
    {
        var inst = Instantiate(crop.PickEffect);
        inst.Stop();
        harvestEffectPool[crop].Add(inst);
    }
}

void OnExitTheGame()
{
    Save();
}
void Save()
{
    PlacementData placementData = new PlacementData();

    foreach (var cropData in _cropData)
    {
        TilePlacementData tilePlacementData = new TilePlacementData();
        tilePlacementData.SetPosition(cropData.Key);

        placementData.CropsTilePlacementDatas.Add(tilePlacementData);

        PlacedCropData placedCropData = new PlacedCropData();
        cropData.Value.Save(placedCropData);

        placementData.PlacedCropDatas.Add(placedCropData);
    }

    foreach (var ground in _groundData)
    {
        TilePlacementData tilePlacementData = new TilePlacementData();
        tilePlacementData.SetPosition(ground.Key);

        placementData.GroundTilePlacementDatas.Add(tilePlacementData);

        placementData.GroupDatas.Add(ground.Value);
    }

    _gameDataState.UpdatePlacementData(placementData);
}
public void Load()
{
    PlacementData placementData = _gameDataState.PlacementData;

    _groundData = new Dictionary<Vector3Int, GroundData>();
    for (int i = 0; i < placementData.GroupDatas.Count; ++i)
    {
        var pos = placementData.GroundTilePlacementDatas[i].GetPosition();

```

```

        _groundData.Add(pos, placementData.GroupDatas[i]);

        GroundTilemap.SetTile(pos, TilledTile);

        WaterTilemap.SetTile(pos, placementData.GroupDatas[i].WaterTimer > 0.0f ? WateredTile : null);
        //GroundTilemap.SetColor(data.GroundDataPositions[i], data.GroundDatas[i].WaterTimer > 0.0f ?
        WateredTiledColorTint : Color.white);
    }

    //clear all existing effect as we will reload new one
    foreach (var pool in harvestEffectPool)
    {
        if (pool.Value != null)
        {
            foreach (var effect in pool.Value)
            {
                Destroy(effect.gameObject);
            }
        }
    }

    _cropData = new Dictionary<Vector3Int, CropData>();
    for (int i = 0; i < placementData.PlacedCropDatas.Count; ++i)
    {
        CropData newData = new CropData();
        newData.Load(placementData.PlacedCropDatas[i], _cropDatabase);

        var pos = placementData.CropsTilePlacementDatas[i].GetPosition();
        _cropData.Add(pos, newData);

        UpdateCropVisual(pos);

        if (!harvestEffectPool.ContainsKey(newData.GrowingCrop))
        {
            InitHarvestEffect(newData.GrowingCrop);
        }
    }
}

public static PlacementService Instance()
{
    return s_instance;
}
}

```

Листинг класса Movement:

```

using Scripts.SO.Player;
using UnityEngine;

public class Movement
{

```

```

InputService _inputService;
Rigidbody2D _rb;
Animator _animator;
Vector2 _delta;
PlayerSO _playerSO;
int _aniX;
int _aniY;

public Movement(InputService inputService, Rigidbody2D rb,
    Animator animator,
    PlayerSO playerSO
)
{
    _inputService = inputService;
    _rb = rb;
    _animator = animator;
    _playerSO = playerSO;

    _aniX = Animator.StringToHash("X");
    _aniY = Animator.StringToHash("Y");
}
public void Update()
{
    _delta = _inputService.GetMovement();
    Animate(_delta);
}
public void FixedUpdate()
{
    _rb.MovePosition(_rb.position + _delta * Time.fixedDeltaTime * _playerSO.Speed);
}

void Animate(Vector2 delta)
{
    _animator.SetFloat(_aniX, delta.x);
    _animator.SetFloat(_aniY, delta.y);
}
}

```

Листинг класса Player:

```

using HappyHarvest;
using Scripts.FarmGameEvents;
using Scripts.InventoryCode;
using Scripts.SaveLoader;
using Scripts.SO.Player;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Zenject;

namespace Scripts.PlayerCode
{
    [RequireComponent(typeof(Rigidbody2D))]
    public class Player : MonoBehaviour

```



```

{
    [SerializeField] private Animator _animator;
    private float _speed;
    InputService _inputService;
    GameState _gameDataState;
    Rigidbody2D _rb;
    PlayerSO _playerSO;
    private Vector2 _currentLookDirection;
    public Vector3 CurrentWorldMousePos { get; private set; }

    public Transform ItemAttachBone;
    private Dictionary<InventoryItem, ItemInstance> m_ItemVisualInstance { get; set; } = new();
    InventoryItemDataBase _itemDatabase;
    PlayerMoney _playerMoney { get; set; }

    private int _dirXHash = Animator.StringToHash("DirX");
    private int _dirYHash = Animator.StringToHash("DirY");
    private int _speedHash = Animator.StringToHash("Speed");
    private int _useHash = Animator.StringToHash("Use");

    [Inject]
    public void Construct(InputService inputService,
        GameState gameDataState,
        InventoryItemDataBase inventoryItemDataBase,
        PlayerMoney playerMoney,
        PlayerSO playerSO)
    {
        _inputService = inputService;
        _gameDataState = gameDataState;
        _playerSO = playerSO;
        _playerMoney = playerMoney;
        _itemDatabase = inventoryItemDataBase;
    }
    private void Awake()
    {
        _rb = GetComponent<Rigidbody2D>();
    }
    public void Start()
    {
        _speed = _playerSO.Speed;
    }
    private void Update()
    {
        CurrentWorldMousePos = GetMousePosition();
    }
    private void FixedUpdate()
    {
        Movement();
    }
    void Movement()
    {
        var move = _inputService.GetMovement();
        if (move != Vector2.zero)
        {
            move = FourDirection(move);
            SetLookDirectionFrom(move);
        }
    }

```

```

else
{
    if (IsMouseOverGameWindow())
    {
        Vector3 posToMouse = CurrentWorldMousePos - transform.position;
        SetLookDirectionFrom(posToMouse);
    }
}

var movement = move * _speed;
var speed = movement.sqrMagnitude;

_animator.SetFloat(_dirXHash, _currentLookDirection.x);
_animator.SetFloat(_dirYHash, _currentLookDirection.y);
_animator.SetFloat(_speedHash, speed);

_rb.MovePosition(_rb.position + movement * Time.fixedDeltaTime);
}
void SetLookDirectionFrom(Vector2 direction)
{
    if (Mathf.Abs(direction.x) > Mathf.Abs(direction.y))
    {
        _currentLookDirection = direction.x > 0 ? Vector2.right : Vector2.left;
    }
    else
    {
        _currentLookDirection = direction.y > 0 ? Vector2.up : Vector2.down;
    }
}
bool IsMouseOverGameWindow()
{
    return !(0 > Input.mousePosition.x || 0 > Input.mousePosition.y || Screen.width < Input.mousePosition.x ||
Screen.height < Input.mousePosition.y);
}
Vector2 FourDirection(Vector2 move)
{
    if (Mathf.Abs(move.x) > Mathf.Abs(move.y))
    {
        move = move.x > 0 ? Vector2.right : Vector2.left;
    }
    else
    {
        move = move.y > 0 ? Vector2.up : Vector2.down;
    }
    return move;
}
public Vector3 GetMousePosition()
{
    Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
    Plane plane = new Plane(Vector3.forward, Vector3.zero);
    float distance;

    if (plane.Raycast(ray, out distance))
    {
        return ray.GetPoint(distance);
    }
}

```

```

        else { return Vector3.zero; }
    }

    public void UseItemVisual(InventoryItem inventoryItem)
    {
        var previousEquipped = inventoryItem;

        if (m_ItemVisualInstance.ContainsKey(previousEquipped))
        {
            var visual = m_ItemVisualInstance[previousEquipped];

            _animator.SetTrigger(visual.AnimatorHash);

            if (visual.Animator != null)
            {
                if (!visual.Instance.activeInHierarchy)
                {
                    //enable all parent as if it's disabled, value cannot be set
                    var current = visual.Instance.transform;
                    while (current != null)
                    {
                        current.gameObject.SetActive(true);
                        current = current.parent;
                    }
                }

                visual.Animator.SetFloat(_dirXHash, _currentLookDirection.x);
                visual.Animator.SetFloat(_dirYHash, _currentLookDirection.y);
                visual.Animator.SetTrigger(_useHash);

                //this mean we finished using an item, the entry is now empty, so we need to disable the visual if any
                if (previousEquipped != null)
                {
                    //This is a bit of a quick fix, this will let any animation to finish playing before we disable the visual.
                    StartCoroutine(DelayedObjectDisable(previousEquipped));
                }
            }
        }
        else
        {
            CreateItemVisual(previousEquipped);
            UseItemVisual(inventoryItem);
        }
    }

    void CreateItemVisual(InventoryItem item)
    {
        if (item.VisualPrefab != null && !m_ItemVisualInstance.ContainsKey(item))
        {
            var newVisual = Instantiate(item.VisualPrefab, ItemAttachBone, false);
            newVisual.SetActive(false);

            m_ItemVisualInstance[item] = new ItemInstance()
            {
                Instance = newVisual,
                Animator = newVisual.GetComponentInChildren<Animator>(),
                AnimatorHash = Animator.StringToHash(item.PlayerAnimatorTriggerUse)
            };
        }
    }

```

```

        };
    }
}
IEnumerator DelayedObjectDisable(InventoryItem item)
{
    yield return new WaitForSeconds(1.0f);
    ToggleVisualExplicit(false, item);
}
void ToggleVisualExplicit(bool enable, InventoryItem item)
{
    if (item != null && m_ItemVisualInstance.TryGetValue(item, out var itemVisual))
    {
        itemVisual.Instance.SetActive(enable);
    }
}
private void OnEnable()
{
    GameEvents.OnExitTheGameEvent +=
        OnExitTheGame;
}
private void OnDisable()
{
    GameEvents.OnExitTheGameEvent -=
        OnExitTheGame;
}
protected void OnExitTheGame()
{
    {
        PlayerData playerData = new PlayerData();
        playerData.SetPosition(_rb.position);
        playerData.Money = _playerMoney.Money;
        _gameDataState.PlayerData = playerData;
    }
}
public void Load(PlayerData playerData)
{
    {
        _rb.position = playerData.GetPosition();
        _playerMoney.Money = playerData.Money;
    }
}
}
}

```

Листинг класса PlayerMoney:

```

using TMPro;

namespace Scripts.PlayerCode
{
    public class PlayerMoney
    {
        TextMeshProUGUI _moneyText;
        public PlayerMoney(int startMoney,
            TextMeshProUGUI moneyText)
        {
            _moneyText = moneyText;
            Money = startMoney;
        }
    }
}

```

```

const int c_maxMoneyValue = 9999;
const int c_minMoneyValue = 0;
public int Money
{
    get
    {
        return _money;
    }
    set
    {
        _money = value;
        if(_money < 0)
            _money = c_minMoneyValue;
        if(_money > c_maxMoneyValue)
            _money = c_maxMoneyValue;
        _moneyText.text = _money.ToString();
    }
}

int _money;
}
}

```

Листинг класса ChestData:

```

using Scripts.InventoryCode;
using System.Collections.Generic;

namespace Scripts.SaveLoader
{
    public class ChestData : PlacementItemData
    {
        public List<InventoryItemData> Items;
        public ChestData()
        {
            Items = new List<InventoryItemData>();
            ItemTypeName = nameof(ChestData);
        }

        public void UpdateItems(List<InventoryItem> inventoryItems)
        {
            foreach (var item in inventoryItems)
            {
                Items.Add(item.GetData());
            }
        }
    }
}

```

Листинг класса PlacementItemData:

```

using UnityEngine;

namespace Scripts.SaveLoader
{
    public class PlacementItemData
    {
        public int X;
        public int Y;
        public string ItemTypeName;

        public PlacementItemData()
        {
            ItemTypeName = nameof(PlacementItemData);
        }
        public void SetPosition(Vector3Int position)
        {
            X = position.x;
            Y = position.y;
        }
        public Vector3Int GetPosition()
        {
            return new Vector3Int(X, Y);
        }
        public override bool Equals(object obj)
        {
            if(obj is PlacementItemData)
            {
                PlacementItemData other = (PlacementItemData)obj;
                if (GetType() == other.GetType())
                {
                    if (other.GetPosition() == GetPosition())
                    {
                        return true;
                    }
                }
                else
                {
                    return false;
                }
            }
            else return false;
        }
        else { return false; }
    }
}

```

Листинг класса ISaveLoadPlacementItem:

```

namespace Scripts.SaveLoader
{
    public interface ISaveLoadPlacementItem
    {
        PlacementItemData GetData();
        void SaveData();
    }
}

```

Листинг класса **PlayerData**:

```
using UnityEngine;

namespace Scripts.SaveLoader
{
    public class PlayerData
    {
        public int Money;
        public float X;
        public float Y;

        public void SetPosition(Vector2 pos)
        {
            X = pos.x;
            Y = pos.y;
        }
        public Vector2 GetPosition()
        {
            return new Vector2(X, Y);
        }
    }
}
```

Листинг класса **SettingsData**:

```
using UnityEngine;

namespace Scripts.SaveLoader
{
    public class SettingsData
    {
        public int ResolutionWidth;
        public int ResolutionHeight;
        public bool IsFoolScreen;
        public float MainVolume;
        public float MusicVolume;
        public float SFXVolume;
        public string InputOverrides;

        public void SetResolution(Resolution resolution)
        {
            ResolutionWidth = resolution.width;
            ResolutionHeight = resolution.height;
        }
        public Resolution GetResolution()
        {
            Resolution resolution = new Resolution();
            resolution.width = ResolutionWidth;
            resolution.height = ResolutionHeight;
            return resolution;
        }
    }
}
```

```
}
}
```

Листинг класса **GameDataSaveLoader**:

```
using System;
using System.Collections.Generic;
using UnityEngine;
using Newtonsoft.Json;
using Assets.Scripts;
using Newtonsoft.Json.Linq;
using Scripts.InventoryCode;

namespace Scripts.SaveLoader
{
    public class GameDataSaveLoader
    {

        public GameDataSaveLoader() { }

        public void SaveGameState(GameDataState gameDataState)
        {
            string json = JsonConvert.SerializeObject(gameDataState, Formatting.Indented);
            PlayerPrefs.SetString(gameDataState.GameDataStateName,
                json);
        }

        public GameDataState LoadGameState(string keyName)
        {
            string json = PlayerPrefs.GetString(keyName);
            GameDataState gameDataState =
                JsonConvert.DeserializeObject<GameDataState>(json,
                    new JsonSerializerSettings
                    {
                        Converters = new List<JsonConverter> {
                            new ItemPlacementDataConverter(),
                            new InventoryItemsDataConverter(),
                        }
                    });
            return gameDataState;
        }

        public void SaveWorldNamesJson(List<string> worldNames)
        {
            string json = JsonConvert.SerializeObject(worldNames);
            PlayerPrefs.SetString(GameConfiguration.SaveLevelNamesKeyName, json);
        }

        public List<string> LoadWorldNamesJson()
        {
            string json =
                PlayerPrefs.GetString(GameConfiguration.SaveLevelNamesKeyName);
            var names =
                JsonConvert.DeserializeObject<List<string>>(json);
            return names;
        }

        public class ItemPlacementDataConverter : JsonConverter
```



```

{
    public override bool CanConvert(Type objectType)
    {
        return objectType == typeof(PlacementItemData);
    }

    public override object ReadJson(JsonReader reader, Type objectType, object existingValue, JsonSerializer
serializer)
    {
        JObject item = JObject.Load(reader);
        var typeName = item["ItemTypeName"].ToString();
        switch (typeName)
        {
            case nameof(ChestData):
                return item.ToObject<ChestData>(serializer);
            case nameof(SandData):
                return item.ToObject<SandData>(serializer);
            case nameof(TreeData):
                return item.ToObject<TreeData>(serializer);
            case nameof(SeedData):
                return item.ToObject<SeedData>(serializer);
            default:
                return item.ToObject<PlacementItemData>(serializer);
        }
    }

    public override void WriteJson(JsonWriter writer, object value, JsonSerializer serializer)
    {
        throw new NotImplementedException();
    }
}

public class InventoryItemsDataConverter : JsonConverter
{
    public override bool CanConvert(Type objectType)
    {
        return objectType == typeof(InventoryItemData);
    }

    public override object ReadJson(JsonReader reader, Type objectType, object existingValue, JsonSerializer
serializer)
    {
        var item = JObject.Load(reader);
        var inventoryItemTypeName = item["InventoryItemTypeName"].ToString();
        switch (inventoryItemTypeName)
        {
            case nameof(HoeItemData):
                return item.ToObject<HoeItemData>(serializer);
            case nameof(BasketItemData):
                return item.ToObject<BasketItemData>(serializer);
            case nameof(ProductItemData):
                return item.ToObject<ProductItemData>(serializer);
            case nameof(WateringItemData):
                return item.ToObject<WateringItemData>(serializer);
            case nameof(SeedBagItemData):
                return item.ToObject<SeedBagItemData>(serializer);
            default:
                throw new Exception("Error with json converter");
        }
    }
}

```

```

    }
}

public override void WriteJson(JsonWriter writer, object value, JsonSerializer serializer)
{
    throw new NotImplementedException();
}
}
}
}
}

```

Листинг класса **GameDataState**:

```

using Scripts.InventoryCode;
using Scripts.PlacementCode;
using System.Collections.Generic;

namespace Scripts.SaveLoader
{
    public class GameDataState
    {
        public string GameDataStateName;
        public PlayerData PlayerData;
        public List<PlacementItemData> PlacementObjectsDataList;
        public List<InventoryItemData> ActivePackInventory;
        public List<InventoryItemData> BackPackInventory;
        public PlacementData PlacementData;

        public GameDataState(string gameDataStateName)
        {
            ActivePackInventory = new List<InventoryItemData>();
            BackPackInventory = new List<InventoryItemData>();
            GameDataStateName = gameDataStateName;
            PlacementObjectsDataList = new List<PlacementItemData>();
            PlacementData = new PlacementData();
        }

        public void UpdateActivePackInventory(List<InventoryItem> inventoryItems)
        {
            List<InventoryItemData> itemDataList =
                new List<InventoryItemData>();
            foreach (var item in inventoryItems)
            {
                itemDataList.Add(item.GetData());
            }
            ActivePackInventory = itemDataList;
        }

        public void UpdateBackPackInventory(List<InventoryItem> inventoryItems)
        {
            List<InventoryItemData> itemDataList =
                new List<InventoryItemData>();
            foreach (var item in inventoryItems)
            {
                itemDataList.Add(item.GetData());
            }
        }
    }
}

```

```

    }
    BackPackInventory = itemDataList;
}
public void UpdatePlacementData(PlacementData placementData)
{
    PlacementData = placementData;
}
public void AddItemData(PlacementItemData itemData)
{
    if (PlacementObjectsDataList.Contains(itemData) == false)
    {
        PlacementObjectsDataList.Add(itemData);
    }
}
public void RemoveItemData(PlacementItemData itemData)
{
    PlacementObjectsDataList.Remove(itemData);
}
}
}
}

```

Листинг класса GameStateGenerator:

```

using Scripts.InteractableObjects;
using Scripts.InventoryCode;
using Scripts.MainMenuCode;
using Scripts.PlacementCode;
using Scripts.PlayerCode;
using Scripts.SaveLoader;
using Scripts.SO.InteractableObjects;
using System.Collections.Generic;
using UnityEngine;
using Zenject;

namespace AScripts.SaveLoader
{
    public class GameStateGenerator : MonoBehaviour
    {
        [Header("Back Pack")]
        [SerializeField] List<InventoryItem> _backPackStartItemKit;
        [SerializeField] InventoryBase _backPackInventory;
        [Space]
        [Space]
        [Header("Active Pack")]
        [SerializeField] List<InventoryItem> _activeStartItemKit;
        [SerializeField] InventoryBase _activeInventory;
        [Space]
        [Space]
        [Header("Seller Pack")]
        [SerializeField] List<InventoryItem> _sellerStartItemKit;
        [SerializeField] InventoryBase _sellerInventory;

        GameState _gameDataState;
        Dictionary<string, InventoryItem> _inventoryItemsDictionary;
        Dictionary<string, SeedSO> _seedSODictionary;
    }
}

```

```

FactoriesProvider _interactableObjectsFactoryProvider;
IChestFactory _chestFactory;
ISeedFactory _seedFactory;
ISandFactory _sandFactory;
ISeedFactory _treeFactory;
ItemPlacementMap _placementMap;
ItemPlacementMap _seedPlacementMap;
SandTilePlacementMap _sandTilePlacementMap;
Player _player;
InventoryItemDataBase _inventoryItemDataBase;
PlacementService _placementService;

[Inject]
public void Construct(GameDataState gameDataState,
    //Dictionary<string, IInventoryItem> inventoryItemsDictionary,
    Dictionary<string, SeedSO> seedSODictionary,
    PlacementMapsContainer placementMapsContainer,
    FactoriesProvider interactableObjectsFactoryProvider,
    InventoryItemDataBase inventoryItemDataBase,
    PlacementService placementService,
    Player player)
{
    _interactableObjectsFactoryProvider = interactableObjectsFactoryProvider;
    _seedSODictionary = seedSODictionary;
    _gameDataState = gameDataState;
    _inventoryItemDataBase = inventoryItemDataBase;
    // _inventoryItemsDictionary = inventoryItemsDictionary;
    _placementMap = placementMapsContainer.ItemPlacementMap;
    _seedPlacementMap = placementMapsContainer.SeedPlacementMap;
    _sandTilePlacementMap = placementMapsContainer.SandTilePlacementMap;
    _player = player;
    _placementService = placementService;
}

private void Start()
{
    _chestFactory =
        (ChestFactory) _interactableObjectsFactoryProvider.GetFactory<ChestFactory>();
    _seedFactory =
        (SeedFactory) _interactableObjectsFactoryProvider.GetFactory<SeedFactory>();
    _sandFactory =
        (SandFactory) _interactableObjectsFactoryProvider.GetFactory<SandFactory>();
    _treeFactory =
        (OakSeedFactory) _interactableObjectsFactoryProvider.GetFactory<OakSeedFactory>();
    LoadPlayerInventories();
    LoadPlacementItems();
    LoadPlayerData();
}
void LoadPlayerData()
{
    PlayerData playerData = _gameDataState.PlayerData;
    if (LoadedData.IsGameStateDefault == false)
    {
        _player.Load(playerData);
    }
}
void LoadPlacementItems()

```

```

{
    if (LoadedData.IsGameStateDefault == true)
        return;

    _placementService.Load();

    List<PlacementItemData> placementItems =
        _gameDataState.PlacementObjectsDataList;

    foreach (PlacementItemData itemData in placementItems)
    {
        switch (itemData)
        {
            case SandData data:
            {
                Vector3Int sandPos = data.GetPosition();
                _sandFactory.Create(sandPos);
                _sandTilePlacementMap.AddPosition(sandPos);
                break;
            }
            case ChestData data:
            {
                List<InventoryItem> inventoryItems = ProcessInventoryItemsData(data.Items);
                Chest chest = _chestFactory.Create(inventoryItems);
                Vector3Int pos3 = data.GetPosition();
                _placementMap.PlaceObjectOnCell(chest.gameObject, pos3);
                _placementMap.AddPosition(pos3);
                break;
            }
            case TreeData data:
            {
                Seed tree = _treeFactory.Create(_seedSODictionary[data.SeedSOName]);
                _placementMap.AddPosition(data.GetPosition());
                _placementMap.PlaceObjectOnCell(tree.gameObject, data.GetPosition());
                tree.LoadSeed(data);
                break;
            }
            case SeedData data:
            {
                Seed seed = _seedFactory.Create(_seedSODictionary[data.SeedSOName]);
                _seedPlacementMap.PlaceObjectOnCell(seed.gameObject, data.GetPosition());
                _seedPlacementMap.AddPosition(data.GetPosition());
                seed.LoadSeed(data);
                break;
            }
        }
    }
}

void LoadPlayerInventories()
{
    if (LoadedData.IsGameStateDefault)
    {
        _backPackInventory.Initialize(CopyItemList(_backPackStartItemKit));
        _activeInventory.Initialize(CopyItemList(_activeStartItemKit));
    }
}

```

```

    }
    else
    {
        var backPackItems = ProcessInventoryItemsData(_gameDataState.BackPackInventory);
        var activePackItems = ProcessInventoryItemsData(_gameDataState.ActivePackInventory);
        _backPackInventory.Initialize(backPackItems);
        _activeInventory.Initialize(activePackItems);
    }
}
List<InventoryItem> CopyItemList(List<InventoryItem> items)
{
    List<InventoryItem> inventoryItems = new List<InventoryItem>();

    foreach (InventoryItem item in items)
    {
        InventoryItem copy;
        copy = (InventoryItem)item.Clone();
        inventoryItems.Add(copy);
    }

    return inventoryItems;
}
private InventoryItem ProcessItemData(InventoryItemData itemData)
{
    string itemName = itemData.SoName;
    InventoryItem item = (InventoryItem)_inventoryItemDataBase.GetItemByName(itemName)
        .Clone();
    item.Count = itemData.Amount;
    switch(item)
    {
        case HoeItem hoeItem:
            return hoeItem;
        case WateringItem wateringItem:
            return wateringItem;
        case SeedBagItem seedBagItem:
            return seedBagItem;
        case ProductItem productItem:
            return productItem;
        case BasketItem basketItem:
            return basketItem;
        default: return item;
    }
}
private List<InventoryItem> ProcessInventoryItemsData(
    List<InventoryItemData> inventoryItemsData)
{
    List<InventoryItem> inventoryItems
        = new List<InventoryItem>();
    foreach (var itemData in inventoryItemsData)
    {
        inventoryItems.Add(ProcessItemData(itemData));
    }
    return inventoryItems;
}
}

```

```
}
```

Листинг класса SettingsDataSaveLoader:

```
using Assets.Scripts;
using Newtonsoft.Json;
using UnityEngine;

namespace Scripts.SaveLoader
{
    public class SettingsDataSaveLoader
    {
        public void Save(SettingsData data)
        {
            var json = JsonConvert.SerializeObject(data);
            PlayerPrefs.SetString(GameConfiguration.SaveSettingsKeyName, json);
        }
        public SettingsData Load()
        {
            var json = PlayerPrefs.GetString(GameConfiguration.SaveSettingsKeyName);
            return JsonConvert.DeserializeObject<SettingsData>(json);
        }
        /// <summary>
        /// возвращает true, если нет сохранения
        /// </summary>
        /// <returns></returns>
        public bool IsDefault()
        {
            return !PlayerPrefs.HasKey(GameConfiguration.SaveSettingsKeyName);
        }
    }
}
```

Листинг класса TradeElementFactory:

```
using Scripts.InteractableObjects;
using UnityEngine;

namespace Scripts.SellBuy
{
    public interface ITradeElementFactory : IGameObjectFactory<TradeElement, Transform>
    {
    }
}
```

Листинг класса TradeElementFactory:

```
using UnityEngine;
using Zenject;

namespace Scripts.SellBuy
{
```

```

public class TradeElementFactory : ITradeElementFactory
{
    TradeElement _tradeElementTemplate;
    IInstantiator _instantiator;
    public TradeElementFactory(TradeElement tradeElementTemplate, IInstantiator instantiator)
    {
        _tradeElementTemplate = tradeElementTemplate;
        _instantiator = instantiator;
    }

    public TradeElement Create(Transform contentArea)
    {
        var element = _instantiator.InstantiatePrefabForComponent<TradeElement>(_tradeElementTemplate,
            contentArea);
        return element;
    }
}

public class SellTradeElementFactory : TradeElementFactory
{
    public SellTradeElementFactory(TradeElement tradeElementTemplate,
        IInstantiator diContainer)
        : base(tradeElementTemplate, diContainer)
    {
    }
}

public class BuyTradeElementFactory : TradeElementFactory
{
    public BuyTradeElementFactory(TradeElement tradeElementTemplate,
        IInstantiator diContainer)
        : base(tradeElementTemplate, diContainer)
    {
    }
}

```

Листинг класса BuyTradeElement:

```

namespace Scripts.SellBuy
{
    public class BuyTradeElement : TradeElement
    {
        public override bool Trade()
        {
            TradeService.Buy(Item, 1);
            return false;
        }
        public void Update()
        {
            ButtonElement.interactable = TradeService.BuyCondition(Item);
        }
    }
}

```

Листинг класса SellTradeElement:


```

using Scripts.FarmGameEvents;
using Scripts.Inventory;
using System.Threading.Tasks;
using UnityEngine;

namespace Scripts.SellBuy
{
    public class SellTradeElement : TradeElement
    {
        ItemContextData _itemData;
        public override async void OnButtonAction()
        {
            if (Trade())
                Destroy(gameObject);
            //StartCoroutine(OverrideSellList(5f));
            await OverrideSellList();
        }
        public override bool Trade()
        {
            TradeService.Sell(_itemData);
            return true;
        }
        public void InitializeItemContext(ItemContextData contextData)
        {
            _itemData = contextData;
        }

        async Task OverrideSellList()
        {
            await Task.Delay(50);
            Debug.Log(500);
            GameEvents.InvokeSellItemEvent();
        }
    }
}

```

Листинг класса TradeElement:

```

using Scripts.InventoryCode;
using TMPro;
using UnityEngine;
using UnityEngine.UI;
using Zenject;

namespace Scripts.SellBuy
{
    public abstract class TradeElement : MonoBehaviour
    {
        public Image IconElement => _iconElementField;
        public TextMeshProUGUI IconName => _iconNameField;
        public TextMeshProUGUI ButtonText => _buttonTextField;
    }
}

```

```

protected Button ButtonElement => _buttonElementField;

[SerializeField] private Image _iconElementField;
[SerializeField] private TextMeshProUGUI _iconNameField;
[SerializeField] private TextMeshProUGUI _buttonTextField;
[SerializeField] private Button _buttonElementField;

protected TradeService TradeService;

protected InventoryItem Item;

[Inject]
public void Construct(TradeService tradeService)
{
    TradeService = tradeService;
}
public void Init(InventoryItem item)
{
    Item = item;
}
public virtual void OnButtonAction()
{
    if(Trade())
        Destroy(gameObject);
}

public abstract bool Trade();
}
}

```

Листинг класса Seller:

```

using Scripts.FarmGameEvents;
using UnityEngine;
using Zenject;

namespace Scripts.SellBuy
{
    public class Seller : MonoBehaviour
    {
        [SerializeField] private GameObject _panelObject;
        [SerializeField] private GameObject TradePanel;
        [SerializeField] private GameObject ChatObject;
        InputService _inputService;

        [Inject]
        public void Construct(InputService inputService)
        {
            _inputService = inputService;
        }

        private void OnMouseDown()

```

```

    {
        if(_panelObject.gameObject.activeSelf == false)
        {
            _panelObject.gameObject.SetActive(true);
        }
        else
        {
            _panelObject.gameObject.SetActive(false);
        }
    }

    public void OnTrade()
    {
        _inputService?.LockGamePlayControls();
        TradePanel.SetActive(true);
        _panelObject.gameObject.SetActive(false);
        GameEvents.InvokeTradePanelActionEvent(!_panelObject.gameObject.activeSelf);
    }

    public void OnTalk()
    {
        _inputService?.LockGamePlayControls();
        ChatObject.SetActive(true);
        _panelObject.gameObject.SetActive(false);
    }
}

```

Листинг класса TradePanel:

```

using Scripts.InventoryCode;
using System.Collections.Generic;
using System.Text;
using UnityEngine;
using Zenject;
using Scripts.InteractableObjects;
using Scripts.Inventory;
using Scripts.FarmGameEvents;
using UnityEngine.EventSystems;
using Scripts.MouseHandle;

namespace Scripts.SellBuy
{
    public class TradePanel : MonoBehaviour, IDragHandler, IBeginDragHandler, IEndDragHandler
    {
        [SerializeField] Transform ContentArea;

        TradeService _tradeService;
        PlayerInventory _playerInventory;
        StringBuilder _buttonTextBuilder = new StringBuilder();
        FactoriesProvider _factoriesProvider;
        ITradeElementFactory _sellElementFactory;
        ITradeElementFactory _buyElementFactory;
        BuyItemsDatabase _buyItemsDatabase;
        MouseCursor _mouseCursor;
    }
}

```

```

InputService _inputService;

[Inject]
public void Construct(TradeService tradeService,
    PlayerInventory playerInventory,
    FactoriesProvider factoriesProvider,
    BuyItemsDatabase buyItemsDatabase,
    MouseCursor mouseCursor,
    InputService inputService)
{
    _inputService = inputService;
    _mouseCursor = mouseCursor;
    _tradeService = tradeService;
    _playerInventory = playerInventory;
    _factoriesProvider = factoriesProvider;
    _buyItemsDatabase = buyItemsDatabase;
}
private void OnEnable()
{
    GameEvents.OnSellItemEvent += OnSellItem;
}
private void OnDisable()
{
    GameEvents.OnSellItemEvent -= OnSellItem;
}
private void Start()
{
    _sellElementFactory =
        (SellTradeElementFactory)_factoriesProvider.GetFactory<SellTradeElementFactory>();
    _buyElementFactory =
        (BuyTradeElementFactory)_factoriesProvider.GetFactory<BuyTradeElementFactory>();
    gameObject.SetActive(false);
}

public void OnBuy()
{
    ClearElements();
    List<InventoryItem> inventoryItems =
        _playerInventory.GetAllItems();
    foreach (var item in _buyItemsDatabase.Items)
    {
        var UIElement = _buyElementFactory.Create(ContentArea);
        var itemClone = item.Clone() as InventoryItem;
        UIElement.Init(itemClone);
        UIElement.IconElement.sprite = item.Icon;
        UIElement.IconName.text = item.DisplayName;

        int amount = _tradeService.ItemAmount(item);
        _buttonTextBuilder.Append($"Spend {item.BuyPrice} " +
            $"for {itemClone.Count}");

        UIElement.ButtonText.text = _buttonTextBuilder.ToString();

        _buttonTextBuilder.Clear();
    }
}

```

```

public void OnSell()
{
    ClearElements();

    var itemsContext = _playerInventory.GetAllItemsContextData();

    //DebugItemContextData(itemsContext);

    foreach (ItemContextData contextData in itemsContext)
    {
        if (_tradeService.SellCondition(contextData.Item))
        {
            var UIElement = _sellElementFactory.Create(ContentArea);

            if (UIElement is SellTradeElement tradeElement)
            {
                tradeElement.InitializeItemContext(contextData);
                tradeElement.Init(contextData.Item.Clone() as InventoryItem);
            }

            UIElement.IconElement.sprite = contextData.Item.Icon;
            UIElement.IconName.text = contextData.Item.DisplayName;

            Debug.Log("Ставлю :" + contextData.Item.DisplayName);

            _buttonTextBuilder.Append($"Take {contextData.Item.Count * contextData.Item.BuyPrice} " +
                $"for {contextData.Item.Count}");

            UIElement.ButtonText.text = _buttonTextBuilder.ToString();

            _buttonTextBuilder.Clear();
        }
    }
}

void DebugItemContextData(List<ItemContextData> itemContextDatas)
{
    StringBuilder stringBuilder = new StringBuilder();

    foreach (var item in itemContextDatas)
    {
        stringBuilder.Append("Get " + item.Item.DisplayName + " ");
    }
    Debug.Log(stringBuilder.ToString());
}

public void OnExit()
{
    ClearElements();
    gameObject.SetActive(false);
    GameEvents.InvokeTradePanelActionEvent(true);
    _inputService.UnlockGamePlayControls();
}

void ClearElements()
{
    for (int i = 0; i < ContentArea.childCount; i++)
    {
        Destroy(ContentArea.GetChild(i).gameObject);
    }
}

```

```

    }
}

void OnSellItem()
{
    OnSell();
}

public void OnDrag(PointerEventData eventData)
{
    Vector3 mousePos = Input.mousePosition;

    gameObject.transform.position = mousePos;
}

public void OnBeginDrag(PointerEventData eventData)
{
    _mouseCursor.ChangeCursor(CursorType.Drag);
}

public void OnEndDrag(PointerEventData eventData)
{
    _mouseCursor.ChangeCursor(CursorType.Default);
}
}
}

```

Листинг класса TradeService:

```

using Scripts.InventoryCode;
using Scripts.PlayerCode;
using Zenject;
using UnityEngine;
using Scripts.Inventory;

namespace Scripts.SellBuy
{
    public class TradeService
    {
        PlayerInventory _playerInventory;
        PlayerMoney _playerMoney;
        [Inject]
        public void Construct(PlayerInventory playerInventory,
            PlayerMoney playerMoney)
        {
            _playerInventory = playerInventory;
            _playerMoney = playerMoney;
        }
        public bool BuyCondition(InventoryItem item)
        {
            if(_playerInventory.IsFull() && item.BuyPrice > 0)
            {
                return false;
            }
            if(item.Consumable)

```

```

        {
            return item.BuyPrice <= _playerMoney.Money;
        }
        else
        {
            return false;
        }
    }

    public void Buy(InventoryItem item, int amount)
    {
        for (int i = 0; i < amount; i++)
        {
            InventoryItem inventoryItem =
                (InventoryItem)item.Clone();
            _playerInventory.TryAddItem(inventoryItem);
        }
        _playerMoney.Money -= item.BuyPrice;
    }
    public bool SellCondition(InventoryItem item)
    {
        return item is ProductItem && item.BuyPrice > 0;
    }
    public void Sell(ItemContextData contextData)
    {
        var item = contextData.Item;
        Debug.Log("Удаляю :" + item.DisplayName);
        _playerInventory.RemoveItem(contextData);

        _playerMoney.Money += item.BuyPrice * item.Count;
    }
    public int ItemAmount(InventoryItem item)
    {
        return Mathf.RoundToInt(_playerMoney.Money / item.BuyPrice);
    }
}
}

```

Листинг класса SoundService:

```

using System;
using System.Collections.Generic;
using UnityEngine.Audio;
using UnityEngine;
using Zenject;
using Scripts.MainMenuCode;

namespace Scripts.Sounds
{
    public class SoundService : MonoBehaviour
    {
        [Serializable]

```

```

public class SoundData
{
    public float MainVolume = 1.0f;
    public float BGMVolume = 1.0f;
    public float SFXVolume = 1.0f;
}

[SerializeField] private AudioMixer _mixer;
[SerializeField] private AudioSource _UISource;
[SerializeField] private AudioSource _sfxReferenceSource;

public SoundData Sound { get; protected set; } = new();

private Queue<AudioSource> m_SFXPool;
SettingsMenu _settingsMenu;

[Inject]
public void Construct(SettingsMenu settingsMenu)
{
    _settingsMenu = settingsMenu;
}

private void Awake()
{
    const int PoolLength = 16;

    m_SFXPool = new Queue<AudioSource>();

    for (int i = 0; i < PoolLength; ++i)
    {
        GameObject obj = new GameObject("SFXPool");
        obj.transform.SetParent(transform);

        var source = Instantiate(_sfxReferenceSource);

        m_SFXPool.Enqueue(source);
    }
}

private void OnEnable()
{
    if (!_settingsMenu.Controls) return;
    _settingsMenu?.AddListenerAtMainVolume(OnMainVolumeValueChange);
    _settingsMenu?.AddListenerAtMusicVolume(OnMusicVolumeValueChange);
    _settingsMenu?.AddListenerAtSFXVolume(OnSFXVolumeValueChange);
}

private void OnDisable()
{
    if (!_settingsMenu.Controls) return;
    _settingsMenu?.RemoveListenerAtMainVolume(OnMainVolumeValueChange);
    _settingsMenu?.RemoveListenerAtMusicVolume(OnMusicVolumeValueChange);
    _settingsMenu?.RemoveListenerAtSFXVolume(OnSFXVolumeValueChange);
}

private void Start()
{
    //Load();
}

```



```

    }

    public void UpdateVolume()
    {
        _mixer.SetFloat("MainVolume", Mathf.Log10(Mathf.Max(0.0001f, Sound.MainVolume)) * 30.0f);
        _mixer.SetFloat("SFXVolume", Mathf.Log10(Mathf.Max(0.0001f, Sound.SFXVolume)) * 30.0f);
        _mixer.SetFloat("BGMVolume", Mathf.Log10(Mathf.Max(0.0001f, Sound.BGMVolume)) * 30.0f);
    }

    public void PlaySFXAt(Vector3 position, AudioClip clip, bool spatialized)
    {
        var source = m_SFXPool.Dequeue();

        source.clip = clip;
        source.transform.position = position;

        source.spatialBlend = spatialized ? 1.0f : 0.0f;

        source.Play();

        m_SFXPool.Enqueue(source);
    }

    public void PlayUISound()
    {
        _UISource.Play();
    }

    void OnMainVolumeValueChanged(float value)
    {
        _mixer.SetFloat("MainVolume", Mathf.Log10(Mathf.Max(0.0001f, value)) * 30.0f);
    }
    void OnSFXVolumeValueChanged(float value)
    {
        _mixer.SetFloat("SFXVolume", Mathf.Log10(Mathf.Max(0.0001f, value)) * 30.0f);
    }

    void OnMusicVolumeValueChanged(float value)
    {
        _mixer.SetFloat("BGMVolume", Mathf.Log10(Mathf.Max(0.0001f, value)) * 30.0f);
    }
}
}

```

Листинг класса IUniversalState:

```

namespace Scripts.StateMachine
{
    public interface IUniversalState
    {
        void Enter();
        void Exit();
        void Perform();
    }
}

```

Листинг класса UniversalStateMachine:

```
namespace Scripts.StateMachine
{
    public abstract class UniversalStateMachine<T> where T : IUniversalState
    {
        protected T CurrentState;
        protected T LastState;

        public void Initialize(T state)
        {
            CurrentState = state;
            ChangeState(state);
        }
        public void ChangeState(T newState)
        {
            LastState = CurrentState;
            CurrentState.Exit();
            CurrentState = newState;
            newState.Enter();
        }
        public void ChangeLastState()
        {
            CurrentState.Exit();
            var state = CurrentState;
            CurrentState = LastState;
            LastState = state;
            CurrentState.Enter();
        }
    }
}
```

Листинг класса UniversalStateMachine:

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace Scripts.InteractableObjects
{
    /// <summary>
    /// Сервис локации
    /// </summary>
    public class FactoriesProvider
    {
        Dictionary<Type, object> _factories;

        public FactoriesProvider()
        {
            _factories = new Dictionary<Type, object>();
        }
        public void RegisterFabric<T, D>
            (IGameObjectFactory<T, D> interactableObjectFactory)
```

```

{
    Type key = interactableObjectFactory.GetType();
    if(!_factories.ContainsKey(key))
    {
        _factories[key] = interactableObjectFactory;
    }
}
public object GetFactory<F>()
{
    Type t = typeof(F);
    return (F)_factories[t];
}
}
}

```

Листинг класса GameConfiguration:

```

namespace Assets.Scripts
{
    public static class GameConfiguration
    {
        public static readonly string FarmSceneName = "FarmScene";
        public static readonly string MainMenuSceneName = "MainMenuScene";
        public static readonly string SaveLevelNamesKeyName = "LevelsKey";
        public static readonly string SaveEditorGameStateName = "Editor mode";
        public static readonly string SaveSettingsKeyName = "SettingsKey";
        public static readonly string NextLevelName = "NextLevel";
    }
}

```

ПРИЛОЖЕНИЕ Б **(обязательное)**

Руководство системного программиста

1. Общие сведения о программе.

Разработанное игровое приложение предназначено для игры между двумя игроками на одном компьютере или на разных устройствах по сети. Разработанное игровое приложение предназначено для развития внимания и реакции, а также для развлечения. Также развивает концентрацию и внимание и значительно улучшает память, позволяя запоминать всё большие объёмы информации.

Для корректной работы приложения необходима следующая конфигурация технических средств аппаратного обеспечения:

- центральный процессор *Intel Core 2 Duo* с тактовой частотой 2.30 МГц или более;
- наличие клавиатуры, мыши и монитора *SVGA* с разрешением не менее 1640 на 750 пикселей;
- операционная система *Windows 7* и выше;
- 100 Мб оперативной памяти;
- скорость интернет-соединения не ниже 100 килобит в секунду.

2. Структура программы.

Игровое приложение логически можно разбить на несколько составляющих: игровой движок, содержащий средства работы с графикой, непосредственно логика игровых объектов и игрового процесса, проект сетевого взаимодействия и графический интерфейс пользователя.

3. Настройка программы.

Приложение необходимо запускать от имени администратора. Для запуска решения необходима среда разработки *Visual Studio* с установленными фреймворком *.NET* и *Unity 2021.3.16f1*.

4. Дополнительные возможности.

Приложение является узконаправленным и не имеет дополнительных возможностей.

ПРИЛОЖЕНИЕ В **(обязательное)**

Руководство программиста

1. Назначения и условия применения программы.

Игровое приложение является платформой для однопользовательской игры на одном устройстве. Основное применение – получение досуга. Доступен лишь один режим – игра на двоих по сети.

Программа показала свою максимальную производительность при взаимодействии с:

- операционной системой *Windows 7*;
- компьютерными средствами: клавиатура, мышь;
- устройствами вывода: монитор.

2. Характеристика программы.

Сразу же после запуска программы загружается главное меню. В главном меню необходимо нажать соответствующую клавишу, сразу после этого начинается игровой процесс, который длится до тех пор, пока не будет нажата клавиша перехода на главную сцену и клавиша закрытия приложения после.

3. Обращение к программе.

Для старта работы приложения необходимо запустить файл с расширением *Farmer.exe*.

4. Входные и выходные данные.

В качестве средств разработки для приложения используется язык программирования *C#*, игровой движок *Unity 2021.3.16f1*.

5. Сообщение программисту.

При возникновении непредвиденных ошибок или остановке работы приложения рекомендуется провести перезапуск всего приложения.

ПРИЛОЖЕНИЕ Г (обязательное)

Руководство пользователя

1. Введение.

Игровое приложение «*Farmer's Valley*» является аркадной платформой для однопользовательской игры на одном устройстве. Основное применение – получение досуга.

Приложение имеет два этапа работы: главное меню, непосредственный игровой процесс.

2. Назначение и условия применения.

Игровое приложение «*Farmer's Valley*» является аркадной платформой для многопользовательской игры по сети.

Программа показала свою максимальную производительность при взаимодействии с:

- операционной системой *Windows 7*;
- компьютерными средствами: клавиатура, мышь;
- устройствами вывода: компьютерный монитор.

3. Подготовка к работе.

Для старта работы приложения необходимо запустить файл с расширением *Farmer.exe*.

4. Описание операций.

Основные манипуляции со стороны игрока:

- старт игры;
- выбор игрового состояния;
- перемещение игрового персонажа по игровому полю влево и вправо;
- перемещение игрового персонажа по игровому полю вверх;
- взаимодействие с интерфейсом;
- окончание игры.

5. Непредвиденные ситуации.

Программа грамотно верифицирована, но в ситуации возникновения ошибки рекомендуется полный перезапуск игрового приложения.

6. Рекомендации.

Данная игра не требует высокого склада ума и хорошей интуиции, каждый желающий игрок может насладиться этой замечательной и увлекательной игрой.

ПРИЛОЖЕНИЕ Д (справочное)

Формулы расчета экономической эффективности

Для расчёта экономической эффективности разработанного программного продукта, используются следующие формулы:

$$K_{\text{эк}} = \frac{K_{\text{т.н}}}{K_{\text{т.б}}}, \quad (\text{Д.1})$$

где $K_{\text{т.н}}$, $K_{\text{т.б}}$ – коэффициенты технического уровня нового и базисного программного продукта, которые можно рассчитать по формуле (Д.2):

$$K_m = \sum_{i=1}^n \beta \frac{P_i}{P_3}, \quad (\text{Д.2})$$

где β – коэффициенты весовости i -го технического параметра;

n – число параметров;

P_i – численное значение i -го технического параметра сравниваемого программного продукта;

P_3 – численное значение i -го технического параметра эталона.

$$K_{\text{ф.в}} = \frac{K_{\text{ф.в.н}}}{K_{\text{ф.в.б}}}, \quad (\text{Д.3})$$

где $K_{\text{ф.в.н}}$, $K_{\text{ф.в.б}}$ – балльная оценка неизмеримых показателей нового и базового изделия соответственно.

$$K_n = \frac{K_{\text{эк}} \cdot K_{\text{ф.в}} \cdot K_n}{K_{\text{ц}}}, \quad (\text{Д.4})$$

где K_n – коэффициент соответствия нового программного продукта нормативам ($K_n = 1$);

$K_{\text{ц}}$ – коэффициент цены потребления.

$$V_o = \sum_{i=1}^n V_i, \quad (\text{Д.5})$$

где V_i – объем отдельной функции ПО;
 n – общее число функций.

$$V_y = \sum_{i=1}^n V_{yi}, \quad (\text{Д.6})$$

где V_{yi} – уточненный объем отдельной функции ПО в строках исходного кода.

$$K_c = 1 + \sum_{i=1}^n K_i, \quad (\text{Д.7})$$

где K_i – коэффициент, соответствующий степени повышения сложности;
 n – количество учитываемых характеристик.

$$T_{y.t.3} = T_n \cdot K_{t.3} \cdot K_c \cdot K_n \cdot K_{y.p}, \quad (\text{Д.8})$$

$$T_{y.э.п} = T_n \cdot K_{э.п} \cdot K_c \cdot K_n \cdot K_{y.p}, \quad (\text{Д.9})$$

$$T_{y.t.п} = T_n \cdot K_{t.п} \cdot K_c \cdot K_n \cdot K_{y.p}, \quad (\text{Д.10})$$

$$T_{y.p.п} = T_n \cdot K_{p.п} \cdot K_c \cdot K_n \cdot K_t \cdot K_{y.p}, \quad (\text{Д.11})$$

$$T_{y.в.н} = T_n \cdot K_{в.н} \cdot K_c \cdot K_n \cdot K_{y.p}, \quad (\text{Д.12})$$

где $K_{t.3}$, $K_{э.п}$, $K_{t.п}$, $K_{p.п}$ и $K_{в.н}$ – значения коэффициентов удельных весов трудоемкости стадий разработки ПО в общей трудоемкости ПО.

$$T_o = \sum_{i=1}^n T_{yi}, \quad (\text{Д.13})$$

где T_{yi} – нормативная (скорректированная) трудоемкость разработки ПО на i -й стадии, чел.-дн.;
 n – количество стадий разработки.

$$Z_p = Z_{tr} + Z_{эт} + Z_{тех} + Z_{м.в} + Z_{мат} + Z_{общ.пр} + Z_{непр}, \quad (\text{Д.14})$$

$$Z_{tr} = Z_{П_{осн}} + Z_{П_{доп}} + ОГЧ_{зп}, \quad (\text{Д.15})$$

где $ЗП_{осн}$ – основная заработная плата разработчиков, руб.;
 $ЗП_{доп}$ – дополнительная заработная плата разработчиков, руб.;
 $ОТЧ_{зп}$ – сумма отчислений от заработной платы (социальные нужды, страхование от несчастных случаев), руб.

$$ЗП_{осн} = C_{ср.час} \cdot T_o \cdot K_{ув}, \quad (Д.16)$$

где $C_{ср.час}$ – средняя часовая тарифная ставка, руб./час;
 T_o – общая трудоемкость разработки, чел.-час;
 $K_{ув}$ – коэффициент доплаты стимулирующего характера, $K_{ув} = 1,8$.

$$C_{ср.час} = \frac{\sum_i C_{чi} \cdot n_i}{\sum_i n_i}, \quad (Д.17)$$

где $C_{чi}$ – часовая тарифная ставка разработчика i -й категории, руб./час;
 n_i – количество разработчиков i -й категории.

$$C_{ч} = \frac{C_{м1} \cdot T_{к1}}{F_{мес}}, \quad (Д.18)$$

где $C_{м1}$ – тарифная ставка 1-го разряда;
 $T_{к1}$ – тарифный коэффициент.

$$ЗП_{доп} = \frac{ЗП_{осн} \cdot H_{доп}}{100}, \quad (Д.19)$$

где $H_{доп}$ – норматив на дополнительную заработную плату разработчиков.

$$ОТЧ_{с.н} = \frac{(ЗП_{осн} + ЗП_{доп}) \cdot H_{з.п}}{100}, \quad (Д.20)$$

где $H_{з.п}$ – процент отчислений на социальные нужды и обязательное страхование от суммы основной и дополнительной заработной платы ($H_{з.п} = 34\%$).

$$З_{М.В} = C_{\text{ч}} \cdot K_{\text{т}} \cdot t_{\text{ЭВМ}}, \quad (\text{Д.21})$$

где $C_{\text{ч}}$ – стоимость 1 часа машинного времени, руб./ч;

$K_{\text{т}}$ – коэффициент мультипрограммности, показывающий распределение времени работы ЭВМ в зависимости от количества пользователей ЭВМ, $K_{\text{т}} = 1$;

$t_{\text{ЭВМ}}$ – машинное время ЭВМ, необходимое для разработки и отладки проекта, ч.

$$C_{\text{ч}} = \frac{З_{\text{П.об}} + З_{\text{ар}} + З_{\text{ам}} + З_{\text{э.п}} + З_{\text{в.м}} + З_{\text{т.р}} + З_{\text{пр}}}{F_{\text{ЭВМ}}}, \quad (\text{Д.22})$$

где $З_{\text{П.об}}$ – затраты на заработную плату обслуживающего персонала с учетом всех отчислений, руб./год;

$З_{\text{ар}}$ – стоимость аренды помещения под размещение вычислительной техники, руб./год;

$З_{\text{ам}}$ – амортизационные отчисления за год, руб./год;

$З_{\text{э.п}}$ – затраты на электроэнергию, руб./год;

$З_{\text{в.м}}$ – затраты на материалы, необходимые для обеспечения нормальной работы ПЭВМ (вспомогательные), руб./год;

$З_{\text{т.р}}$ – затраты на текущий и профилактический ремонт ЭВМ, руб./год;

$З_{\text{пр}}$ – прочие затраты, связанные с эксплуатацией ПЭВМ, руб./год;

$F_{\text{ЭВМ}}$ – действительный фонд времени работы ЭВМ, час/год.

$$\frac{З_{\text{П.об}}}{100} = \frac{З_{\text{осн.об}} + З_{\text{доп.об}} + \text{ОТЧ}_{\text{эп.об}}}{100}, \quad (\text{Д.23})$$

$$З_{\text{осн.об}} = 12 \cdot \sum_i (C_{\text{м.об}i} \cdot n_i), \quad (\text{Д.24})$$

$$\frac{З_{\text{осн.об}} \cdot H_{\text{доп}}}{100}, \quad (\text{Д.25})$$

$$\frac{\text{ОТЧ}_{\text{зп.об}}}{100} = \frac{(З_{\text{осн.об}} + З_{\text{доп.об}}) \cdot H_{\text{зп}}}{100}, \quad (\text{Д.26})$$

где $З_{\text{осн.об}}$ – основная заработная плата обслуживающего персонала, руб.;

$ЗП_{\text{доп.об}}$ – дополнительная заработная плата обслуживающего персонала, руб.;

$ОТЧ_{\text{зп.об}}$ – сумма отчислений от заработной платы (социальные нужды, страхование от несчастных случаев), руб.;

$Q_{\text{эвм}}$ – количество обслуживаемых ПЭВМ, шт.;

$С_{\text{м.об}i}$ – месячная тарифная ставка i -го работника, руб.;

n – численность обслуживающего персонала, чел.;

$Н_{\text{доп}}$ – процент дополнительной заработной платы обслуживающего персонала от основной;

$Н_{\text{зп}}$ – процент отчислений на социальные нужды и обязательное страхование от суммы основной и дополнительной заработной платы.

$$\frac{С_{\text{ар}} \cdot S}{Q_{\text{эвм}}}, \quad (Д.27) \quad З_{\text{ар}} =$$

где $С_{\text{ар}}$ – средняя годовая ставка арендных платежей, руб./м²;

S – площадь помещения, м².

$$\frac{\sum_i З_{\text{при}}(1 + K_{\text{доп}}) m_i \cdot Н_{\text{ам}i}}{100}, \quad (Д.28) \quad З_{\text{ам}} =$$

где $З_{\text{при}}$ – затраты на приобретение i -го вида основных фондов, руб.;

$K_{\text{доп}}$ – коэффициент, дополнительных затраты, связанные с доставкой, монтажом и наладкой оборудования, $K_{\text{доп}} = 12\%$ от $З_{\text{пр}}$;

$З_{\text{при}}(1 + K_{\text{доп}})$ – балансовая стоимость ЭВМ, руб.;

$Н_{\text{ам}i}$ – норма амортизации, %.

$$\frac{M_{\text{сум}} \cdot F_{\text{эвм}} \cdot C_{\text{эл}} \cdot A}{100}, \quad (Д.29) \quad З_{\text{эвм}} =$$

где $M_{\text{сум}}$ – паспортная мощность ПЭВМ, кВт;

$M_{\text{сум}} = 0,44$ кВт;

$C_{\text{эл}}$ – стоимость одного кВт-часа электроэнергии, руб.;

A – коэффициент интенсивного использования мощности, $A=0,98 \dots 0,9$.

$$F_{\text{ЭВМ}} = (D_{\text{г}} - D_{\text{вых}} - D_{\text{пр}}) \cdot F_{\text{см}} \cdot K_{\text{см}} \cdot (1 - K_{\text{пот}}), \quad (\text{Д.30})$$

где $D_{\text{г}}$ – общее количество дней в году, $D_{\text{г}} = 365$ дней;

$D_{\text{вых}}$, $D_{\text{пр}}$ – число выходных и праздничных дней в году, $D_{\text{вых}} + D_{\text{пр}} = 121$ дней;

$F_{\text{см}}$ – продолжительность 1 смены, $F_{\text{см}} = 8$ часов;

$K_{\text{см}}$ – коэффициент сменности, $K_{\text{см}} = 1$;

$K_{\text{пот}}$ – коэффициент, учитывающий потери рабочего времени, связанные с профилактикой и ремонтом ЭВМ, примем $K_{\text{доп}} = 0,2$.

$$K_{\text{м.з}}, \quad Z_{\text{в.м}} = \sum_i Z_{\text{при}} (1 + K_{\text{доп}}) m_i \cdot \quad (\text{Д.31})$$

где $Z_{\text{пр}}$ – затраты на приобретение (стоимость) ЭВМ, руб.;

$K_{\text{доп}}$ – коэффициент, характеризующий дополнительные затраты, связанные с доставкой, монтажом и наладкой оборудования, $K_{\text{доп}} = 12\text{--}13\%$ от $Z_{\text{пр}}$;

$K_{\text{м.з}}$ – коэффициент, характеризующий затраты на вспомогательные материалы ($K_{\text{м.з}} = 0,01$).

$$K_{\text{т.р}}, \quad Z_{\text{т.р}} = \sum_i Z_{\text{при}} (1 + K_{\text{доп}}) m_i \cdot \quad (\text{Д.32})$$

где $K_{\text{т.р}}$ – коэффициент, характеризующий затраты на текущий и профилактический ремонт, $K_{\text{т.р}} = 0,08$.

$$K_{\text{пр}}, \quad Z_{\text{пр}} = \sum_i Z_{\text{при}} (1 + D_{\text{доп}}) m_i \cdot \quad (\text{Д.33})$$

где $K_{\text{пр}}$ – коэффициент, характеризующий размер прочих затрат, связанных с эксплуатацией ЭВМ ($K_{\text{пр}} = 0,05$).

$$t_{\text{ЭВМ}} = (t_{\text{р.п}} + t_{\text{вн}}) \cdot F_{\text{см}} \cdot K_{\text{см}}, \quad (\text{Д.34})$$

где $t_{\text{р.п}}$ – срок реализации стадии «Рабочий проект» (РП);

$t_{\text{вн}}$ – срок реализации стадии «Ввод в действие» (ВП);

$$t_{p.п} + t_{вн} = 33;$$

$F_{см}$ – продолжительность рабочей смены, ч., $F_{см} = 8$ ч.;

$K_{см}$ – количество рабочих смен, $K_{см} = 1$.

$$З_{эт} = (З_{т.р} + З_{тех} + З_{м.в}) K_{эт}, \quad (Д.35)$$

где $K_{эт}$ – коэффициент, учитывающий размер затрат на изготовление эталонного экземпляра, $K_{эт} = 0,05$.

$$З_{мат} = \sum_i Ц_i N_i (1 + K_{т.з}) - Ц_{0i} N_{0i}, \quad (Д.36)$$

где $Ц_i$ – цена i -го наименования материала полуфабриката, комплектующего, руб.;

N_i – потребность в i -м материале, полуфабрикате, комплектующем, натур. ед.;

$K_{т.з}$ – коэффициент, учитывающий сложившийся процент транспортно-заготовительных расходов в зависимости от способа доставки товаров, $K_{т.з} = 0,1$;

$Ц_{0i}$ – цена возвратных отходов i -го наименования материала, руб.;

N_{0i} – количество возвратных отходов i -го наименования, натур. ед.;

n – количество наименований материалов, полуфабрикатов, и т.д.

$$З_{общ.пр} = \frac{З_{П_{осн}} \cdot Н_{доп}}{100}, \quad (Д.37)$$

где $Н_{доп}$ – норматив общепроизводственных затрат.

$$З_{непр} = \frac{З_{П_{осн}} \cdot Н_{непр}}{100}, \quad (Д.38)$$

где $Н_{непр}$ – норматив непроизводственных затрат.

$$Ц_{отп} = З_p + П_p, \quad (Д.39)$$

$$П_p = \frac{З_p \cdot У_p}{100}, \quad (Д.40)$$

где Z_p – себестоимость ПО, руб.;

P_p – прибыль от реализации программного продукта, руб.;

Y_p – уровень рентабельности программного продукта, % ($Y_p = 30\%$).

$$C_{отп} = Z_p + P_p + P_{ндс}, \quad (Д.41)$$

$$P_{ндс} = \frac{(Z_p + P_p) \cdot H_{ндс}}{100}, \quad (Д.42)$$

где $H_{ндс}$ – ставка налога на добавленную стоимость, %, $H_{ндс} = 20\%$.

$$C_n = C_{отп} / O_a, \quad (Д.43)$$

где C_n – итоговая начальная отпускная цена;

$C_{отп}$ – цена отпуска;

O_a – объём аудитории, чел.

$$\Delta_{пр} = (Z_{пр.б} - Z_{пр.н}) \cdot A_{пр.н}, \quad (Д.44)$$

где $Z_{пр.б}$, $Z_{пр.н}$ – приведенные затраты на единицу выпуска ПО по базовому и новому вариантам, руб.;

$A_{пр.н}$ – годовой объём выпуска в расчетном году для реализуемого ПО, ед.

$$P_p = K_{е.п.} \cdot P_{с.п.} \cdot K_d \cdot K_m \quad (Д.45)$$

где $K_{е.п.}$ – количество пользователей в день, равняется 40;

$P_{с.п.}$ – средний ежедневных доход с пользователя, равняется 0,18 руб.;

K_d – количество дней, равняется 30;

K_m – количество месяцев, равняется 12.

$$P = \frac{\Pi}{B}, \quad (Д.46)$$

где Π – показатель прибыли в год;

B – показатель выручки.

ПРИЛОЖЕНИЕ Ё (обязательное)

Добавлено примечание ((П6)): Приложение Ё должно следовать за приложением Д.

Результат опытной эксплуатации

Приложение оперирует при следующей минимальной конфигурации аппаратного обеспечения:

- процессор с архитектурой *x86-64* и тактовой частотой не менее 2.0 ГГц;
- клавиатура и мышь, обеспечивающие стандартное взаимодействие с приложением;
- монитор с разрешением 1280 на 720 пикселей или выше;
- операционная система *Windows 7/8/10*, *macOS 10.12* и выше или *Linux*;
- 4 Гб оперативной памяти;
- графический процессор с поддержкой *DirectX 11* или *OpenGL 4.2*, с 2 Гб видеопамяти или более.

Опытная эксплуатация проводилась в течение двух часов на персональных компьютерах различной конфигурации, используемых разными пользователями. В ходе тестирования осуществлялась проверка следующих аспектов:

- запуск приложения;
- полное прохождение игры пользователем;
- взаимодействие с инвентарем;
- корректное отображение интерфейса;
- корректная работа торговли;
- загрузка и сохранение игрового состояния;
- закрытие приложения.

Необходимости в доработке или устранении проблем не выявлено, так как в процессе эксплуатации не было выявлено никаких ошибок или недочетов.

ПРИЛОЖЕНИЕ Ж
(рекомендуемое)

Список опубликованных работ

1. Дубовцов, И.Д. Искусственный интеллект в играх жанра *RPG*/ И. Д. Дубовцов// Новые математические методы и компьютерные технологии в проектировании, производстве и научных исследованиях: материалы XXVII Республиканской научной конференции студентов, магистрантов и аспирантов, Гомель, 18 - 20 апреля 2024 г. / ГГУ им. Франциска Скорины ; под ред: С. П. Жогаль [и др.] – Гомель, 2024. – 193 с.

2. Дубовцов, И.Д. Искусственный интеллект в играх жанра *RPG*/ И. Д. Дубовцов// Новые математические методы и компьютерные технологии в проектировании, производстве и научных исследованиях: материалы I Междунар. науч.-техн. конф, студентов, аспирантов и молодых ученых, Гомель, 29 фев. 2024 г. / Гомел. гос. техн. ун-т им. П. О. Сухого [и др.] ; под общ. ред. А. А. Бойко. – Гомель : ГГТУ им. П. О. Сухого, 2024. – 319 с.

3.