МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования
«Гомельский государственный технический университет имени П.О.Сухого»

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

направление специальности 1-40 05 01-12 Информационные системы и технологии (в игровой индустрии)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе

по дисциплине «Программирование сетевых приложений»
на тему: «Игровое приложение «Ферма» для двух игроков по сети с использование среды разработки *Unity*, стека протоколов *TCP/IP*»

Исполнитель: студент группы ИТИ-41
Дубовцов И.Д.
Руководитель: заведующий кафедрой
Курочка К.С.

Дата проверки: _____
Дата допуска к защите: _____
Дата защиты: _____

Оценка работы: _____
Подписи членов комиссии
по защите курсовой работы: _____

Гомель 2023

СОДЕРЖАНИЕ

Введение 4
1 Игровые приложения и средства их разработки 5
1.1 Многопользовательские аркадные игровые приложения 5
1.2 Игровой движок *Unity* 7
1.3 Сетевое взаимодействие в играх 11
1.4 Язык программирования *C#* 13
2 Программная реализация игрового сетевого приложения «Ферма» 15
2.1 Архитектура приложения «Ферма» 15
2.2 Структура приложения «Ферма» 17
2.3 Описание разработанных классов 20
3 Верификация и апробация игрового сетевеого приложения «Ферма» 26
3.1 Принцип работы приложения «Ферма» 26
3.2 Тестирование приложения «Ферма» 27
3.3 Верификация приложения «Ферма» 28
Заключение 32
Список использованных источников 33
Приложение А Исходный код приложения 34
Приложение Б Руководство системного программиста 96
Приложение В Руководство программиста 97
Приложение Г Руководство пользователя 98
Приложение Д Внешний вид окон интерфейса приложения 99
Приложение Ё Результат опытной эксплуатации 100
Приложение Ж Схема структуры приложения 101

ВВЕДЕНИЕ

Популяризация компьютерной игровой индустрии начала свое зарождение с обычной идеи создания игры, в которой игрок перемешает по экрану световые лучи, возникнувшей еще до начала 1970-х годов. В это время взаимодействие человека и вычислительной техники уже стало непреложным фактором.

Видео игры проникли во много сфер современной жизни. На сегодняшний день индустрия компьютерных игр является одной из самых прибыльных и скоростных по развитию отраслей компьютерных технологий и, вместе с этим, глобального сектора развлечений. Рынок компьютерных игр становится все глубже и уже к 2019 году вырос в среднем на 11% и обошел ряд конкурентных рынков, к примеру, кино-, музыкальная индустрия.

Большую долю игровой индустрии занимают сетевые игры. Сетевые игры стали неотъемлемой частью современной культуры и развлечений, оказывая значительное влияние на общество и формируя новые тренды. Они требуют стабильного и быстрого интернет-соединения. Чем выше скорость передачи данных, тем лучше качество игрового опыта.

Сетевое взаимодействие в сетевых приложениях может столкнуться с рядом проблем, которые влияют на качество связи и пользовательский опыт: задержки, потеря пакетов, сетевые атаки, безопасность. Для борьбы с этими проблемами разработчики сетевых приложений должны постоянно следить за стабильностью сетевого взаимодействия, внедрять средства безопасности и оптимизации, а также предоставлять поддержку для пользователей при возникновении проблем.

В этой курсовой работе будет разработано игровое сетевое приложение «Ферма». Будет проведен анализ существующих подходов к разработке сетевых приложений с использованием стека протоколов *TCP/IP*, а также рассмотрены современные вызовы и трудности, с которыми разработчики сталкиваются при создании таких приложений на игровом движке *Unity*.

Приложения, использующие стек протоколов *TCP/IP*, продолжают оставаться актуальными и перспективными в силу своей универсальности, надежности и широкого применения в современной информационной инфраструктуре.

1 ИГРОВЫЕ ПРИЛОЖЕНИЯ И СРЕДСТВА ИХ РАЗРАБОТКИ

1.1 Многопользовательские аркадные игровые приложения

Многопользовательские аркадные игры занимают особое место в сфере развлечений, предоставляя игрокам уникальный опыт совместного взаимодействия в виртуальных мирах. Этот подраздел посвящен рассмотрению представителей аркадного жанра, ориентированных на многопользовательский режим. Аркадные игры, как жанр, привлекают внимание своей простотой управления, динамичным геймплеем и часто выразительным дизайном. Таким образом, чтобы понять, что такое аркадные игры, стоит рассмотреть их достойных представителей.

1.1.1 WormsBattlegrounds – компьютерная игра серии *Worms* в жанре пошаговой стратегии, разработанная и изданная британской компанией *Team17*.

На рисунке 1.1 представлен кадр из игры *WormsBattlegrounds*.

Рисунок 1.1 – Кадр из игры *WormsBattlegrounds*

Игровой процесс сохраняет схожесть с предшествующими частями серии: управление командой червяков осуществляется игроком, который использует разнообразное оружие, такое как базуки, гранаты, овцы и бананы, с целью уничтожения червяков других команд. При этом каждая команда выполняет выстрелы поочередно, и побеждает та команда, в которой хотя бы один червяк остается живым, в то время как червяки противников погибают.

Основным инновационным элементом в игре стала физика объектов и жидкостей. Некоторые объекты на ландшафтах, такие как канистры, гайки и

баллоны, подвластны движению при толчке или под воздействием выстрела. Кроме того, выстрел по некоторым из этих объектов влияет на ход битвы; например, взрыв канистры с бензином создает огонь вокруг нее, а взрыв баллона отравляет червяков, находящихся поблизости. Попадание червяка в воду на случайном участке влечет за собой утрату пяти единиц здоровья каждый ход. Также присутствует оружие, позволяющее добавлять на ландшафт больше воды.

Игра включает режимы кампании для одиночного игрока с разнообразными настройками оружия и ландшафтов, а также многопользовательские варианты игры за одним компьютером или через сеть. Кроме того, предоставляется возможность настройки внешнего вида червяков, например, изменение их шляп или голосов. Версии *Deluxe* и *Extreme* для портативного компьютера и *PS Vita* с *PlayStation 3* включают в себя дополнения, представленные новыми ландшафтами, миссиями и одеждой для червяков. Эти дополнения изначально были выпущены для консолей и компьютеров.

1.1.2 Super Meat Boy – аркадная компьютерная инди-игра в жанре платформера, разработанная командой *Team Meat*.

В игре акцент делается на высокой точности управления персонажем, где игрок управляет «Мясным пацаном» – квадратным персонажем, лишенным кожи, с задачей спасти «Пластыревую девушку» от «Доктора зародыша».

На рисунке 1.2 представлен кадр из игры *Super Meat Boy*.

Рисунок 1.2 – Кадр из игры *Super Meat Boy*

Геймплей включает в себя преодоление уровней с различными ловушками, такими как циркулярные пилы, иглы и соль, с использованием способности скользить по стенам.

Количество жизней не ограничено, и после смерти персонаж мгновенно возрождается в начале уровня. После успешного прохождения уровня игроку предоставляется возможность просмотра повтора последних 40 попыток прохождения.

Структура игры включает пять основных глав, каждая из которых содержит по 20 уровней. Для открытия босса игроку необходимо успешно завершить не менее 17 уровней в любом порядке. Пройдя пятую главу, открывается шестая глава под названием «*The End*», состоящая из пяти этапов. После ее завершения игрок сталкивается с финальным боссом – «*Dr. Fetus»*. После победы над «*Dr. Fetus»* открывается седьмая глава «*The Cotton Alley*», представляющая собой 20 сложных уровней, где доступен только персонаж «Пластыревая девушка». Общее число уровней в игре достигает приблизительно 350.

Игра также предлагает дополнительные элементы, такие как синие порталы (далее *warp*-зоны) – синие порталы, ведущие игрока в мини-игры с ретро-стилизацией. Каждая глава включает в себя по четыре *warp*-зоны. Присутствует также особая *warp*-зона красного цвета в светлом мире, предоставляющая бонус в виде разблокировки дополнительных персонажей. Игрок может также встретить видоизмененную «пластыревую девушку» на случайном уровне после прохождения основной главы.

В ранних стадиях разработки структура игры включала 15 уровней в каждой главе, разделенных на наборы по 3 этапа, с 16 пластырями на главу и отсутствием темного мира и *warp*-зон. Постепенно разработчики отошли от системы наборов, сделав все уровни глав изначально доступными.

1.2 Игровой движок *Unity*

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

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, прегражденние окружающего света в экранном пространстве (*Screen Space Ambient Occlusion, SSAO*), динамические тени. 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 которой 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 компоноваться сцены будущей игры, связывая игровые ресурсы и код в интерактивные объекты. Он позволяет быстро и рационально создавать профессиональные игры, обеспечивая невиданную продуктивность разработчиков и предоставляя в их распоряжение исчерпывающий список самых современных технологий в области видеоигр. На рисунке 1.3 представлен интерфейс приложения *Unity*.

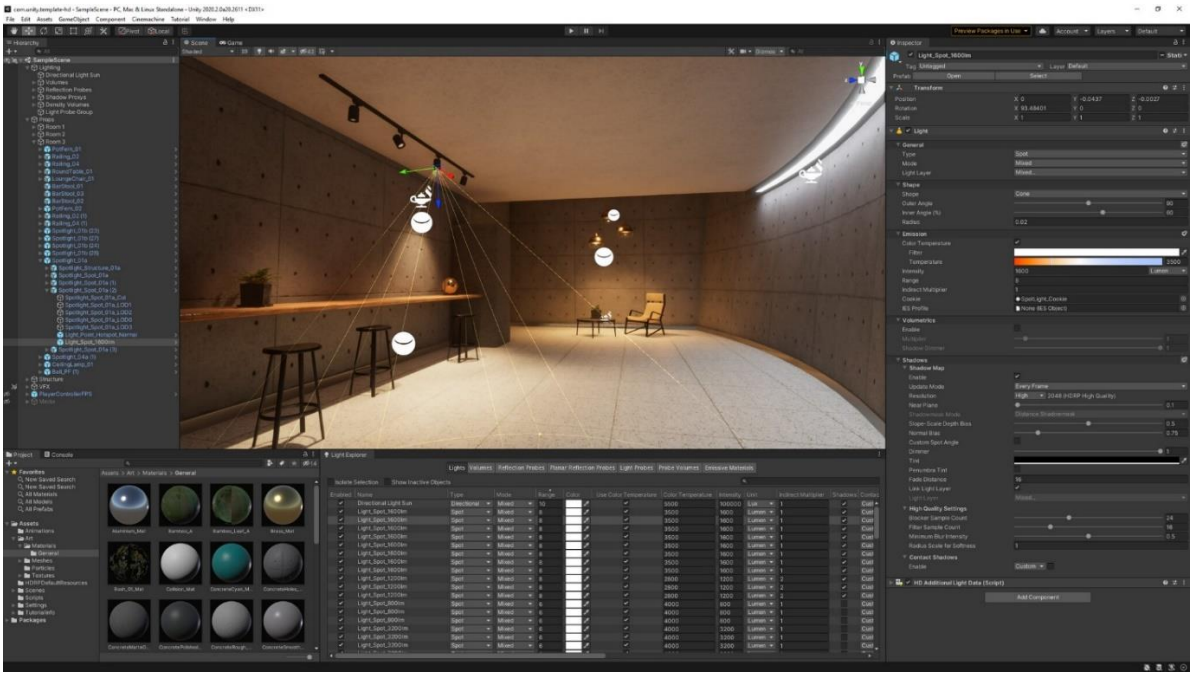

Рисунок 1.3 – Интерфейс *Unity* приложения

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.

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*). 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.

Изначально *Unity* предназначался для создания трехмерных игр, но у него есть и другие варианты применения. Начиная с версии 4.3, выпущенной в конце 2013 года, в *Unity* появилась возможность отображения двумерной графики, хотя и раньше с помощью этого инструмента разрабатывали двумерные игры (особенно мобильные, в создании которых помогла кросс-платформенная природа *Unity*). Изначально для эмуляции двумерной графики в трехмерных сценах требовался сторонний фреймворк (например, *2D Toolkit* от *Unikron Software*). В конечном счете основной редактор и игровой движок поменяли, встроив в него двумерную графику. Именно с этой функциональностью и знакомит данная глава. Рабочий процесс при создании двумерной и трехмерной графики в *Unity* примерно одинаков и включает в себя импорт графических ресурсов, перетаскивание их в сцену и написание сценариев, которые затем будут присоединены к объектам. Основной вид ресурсов, необходимых для создания двумерной графики, называется «спрайтом».

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.

Оба подхода схематично показаны на рисунке 1.4.

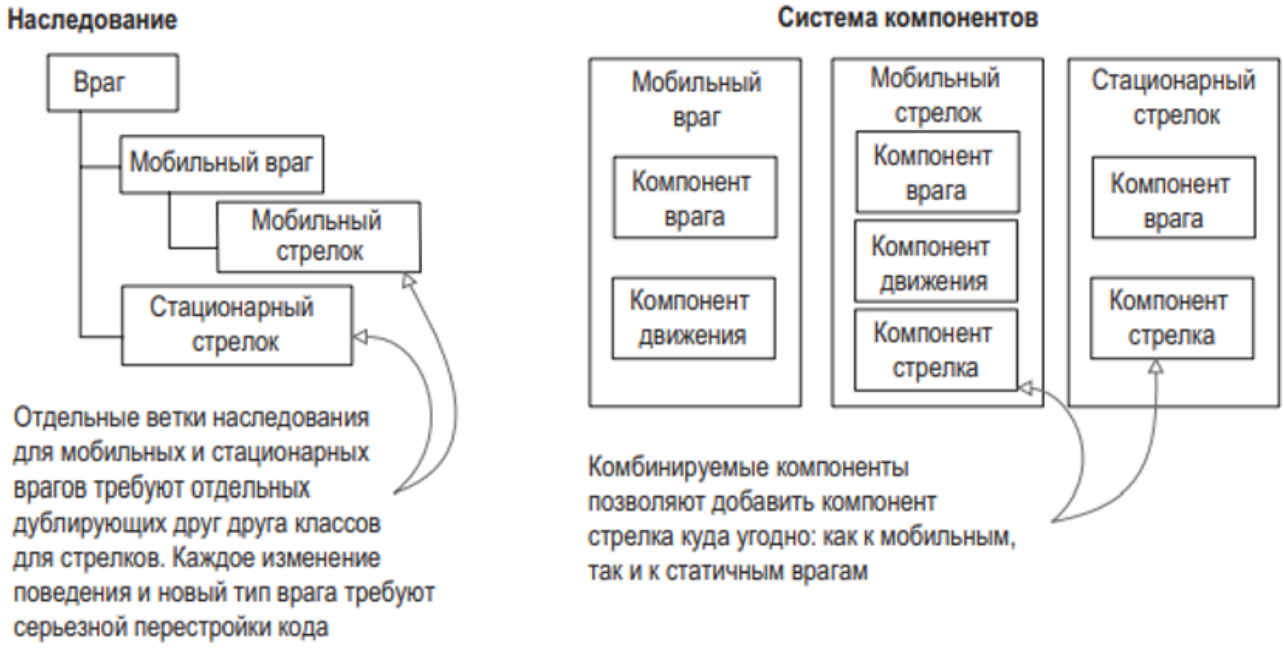

Рисунок 1.4 – Сравнение наследования с компонентной системой

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. Такая компоновка облегчает создание прототипов, потому что взять нужный набор компонентов куда быстрее и проще, чем перестраивать цепочку наследования при изменении каждого объекта. 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 [4, с.17].

1.3 Сетевое взаимодействие в играх

Сетевое взаимодействие представляет собой установление связи между двумя или более компьютерами. Основная концепция заключается в установлении взаимоотношений между клиентом (компьютером, осуществляющим запрос информации) и сервером (компьютером, ответственным за обработку запросов информации). Сервер может представлять собой выделенный хост-компьютер, принимающий запросы от всех участников, а также компьютер одного из игроков, на котором запущена игра и который действует в роли сервера для остальных участников. С момента запуска сервера и подключения клиента эти два компьютера приобретают возможность взаимообмена информацией, соответствующей игровому процессу.

Существуют две основные модели взаимодействия в сфере сетевых игр.

Первая модель, известная как *Peer-to-peer*, представляет собой архитектуру, основанную на равноправном взаимодействии программ. В этой модели каждая запущенная копия игры обладает равными правами и обязана отслеживать состояние других копий, осуществляя передачу сообщений между ними. Эта модель эффективна для игр с небольшим количеством участников, однако при увеличении числа игроков наблюдается значительный поток сообщений, необходимых для поддержания игрового процесса. Схематическое представление модели *peer-to-peer* представлено на рисунке 1.5.

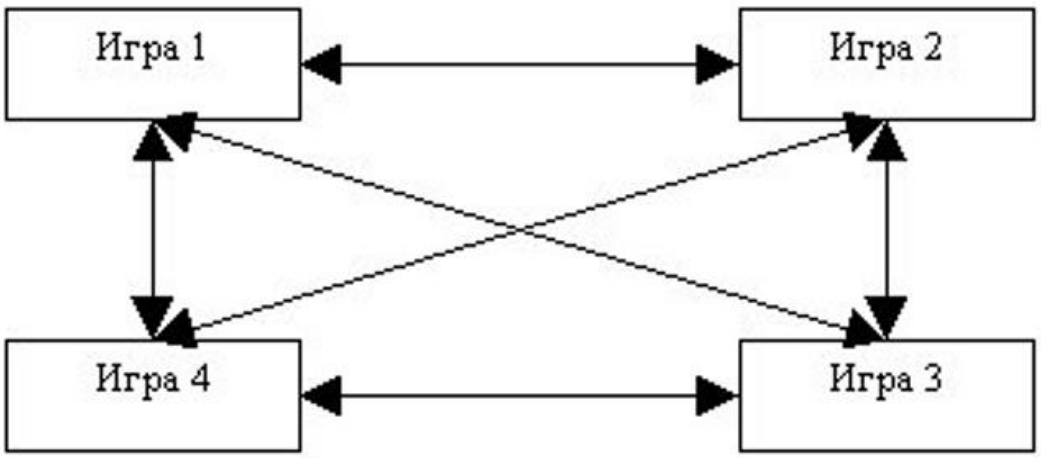

Рисунок 1.5 – *Peer-to-peer*

Часто в этой модели одна из игр назначается хостом, к которому обращаются новые игроки для участия. Однако хост не управляет обменом сообщениями между играми.

Вторая модель взаимодействия в сетевых играх – модель *Client-Server* (Клиент-Сервер). Здесь выделяются два типа программ: программа-сервер, ответственная за организацию и взаимодействие в игре, и программы-клиенты, обменивающиеся сообщениями только с сервером, не взаимодействуя между собой напрямую. Такой подход позволяет эффективно использовать пропускную способность каналов связи и поддерживать участие множества игроков в игре одновременно. Модель взаимодействия *Client-Server* схематически изображена на рисунке 1.6.

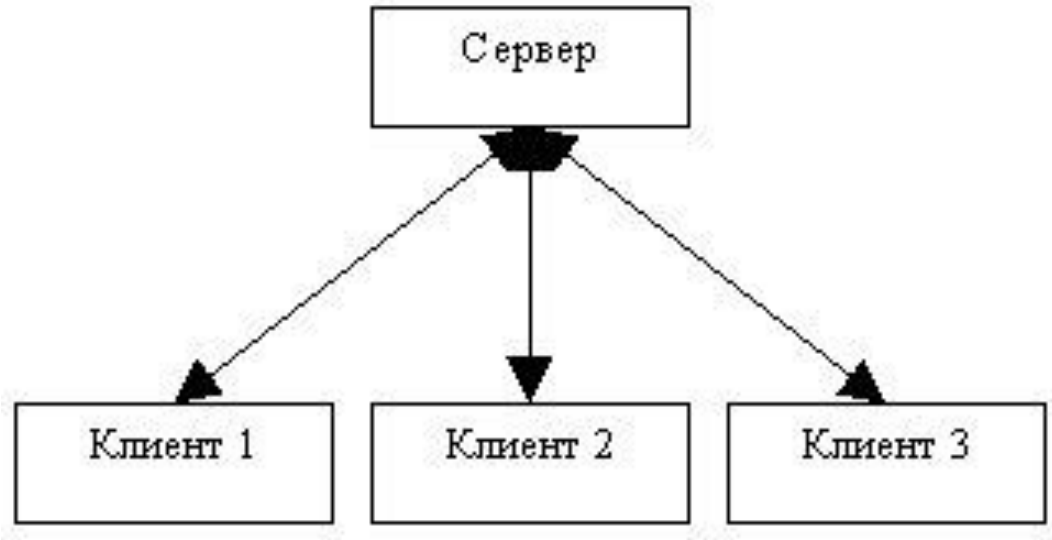

Рисунок 1.6 – *Client-Server*

Каждая игра в модели одноранговой архитектуры является самостоятельной единицей, обеспечивающей полноценную организацию игрового процесса. Каждая копия игры функционирует независимо, передавая информацию о своем состоянии другим копиям и получая данные о состоянии других игр. В клиент-серверной схеме взаимодействия существует несколько подходов, включая разделение вычислений между сервером и клиентами.

Существуют и другие модели организации сетевого взаимодействия игр, такие как кольцевая система обмена сообщениями или использование нескольких игровых серверов.

Относительно видов сетевых игр, их можно разделить на два основных типа. Первый - пошаговые игры (*Turn Based Games*), к примеру, шахматы или компьютерные версии настольных игр. Второй тип - игры реального времени (*Real Time Games*), представленные, например, игрой «*Dota 2»*. При этом пошаговые игры лучше переносят задержки в сетевой связи, тогда как игры

реального времени могут столкнуться с серьезными проблемами при даже небольших сбоях в связи.

Многие современные игры, не предназначенные для многопользовательского режима, поддерживают хранение результатов игр на специальных интернет-сервисах производителей, что позволяет игрокам соревноваться «заочно».

1.4 Язык программирования *C#*

C# – это язык с *C*-подобным синтаксисом. Здесь он близок в этом отношении к *C++* и *Java* [4, с.21].

Будучи объектно-ориентированным языком, он много перенял у *Java* и *C++*. Как и *Java*, *C#* изначально предназначался для веб-разработки, и примерно 75% его синтаксических возможностей такие же, как у *Java*. *C#* также называют «очищенной версией *Java*». Ещё 10% *C#* позаимствовал из *C++* и 5% – из *Visual Basic*. Оставшиеся 10% *C#* – это реализация собственных идей разработчиков. Объектно-ориентированный подход позволяет строить с помощью *C#* крупные, но в то же время гибкие, масштабируемые и расширяемые приложения.

C# уже давно поддерживает много полезных функций:

– инкапсуляция;

– наследование;

– полиморфизм;

– перегрузка операторов;

– статическая типизация [5, с.43].

При этом он всё ещё активно развивается, и с каждой новой версией появляется всё больше интересного – например лямбды, динамическое связывание, асинхронные методы.

По сравнению с другими языками *C#* довольно молод, но в то же время он уже прошёл большой путь. Первая версия языка вышла вместе с релизом *Microsoft Visual Studio .NET* в феврале 2002 года. Текущей версией языка является версия *C# 8.0*, которая вышла в сентябре 2019 года вместе с релизом *.NET Core 3* [6, с.28].

У «шарпа» выделяют много преимуществ:

– поддержка подавляющего большинства продуктов *Microsoft*;

– бесплатность ряда инструментов для небольших компаний и некоторых индивидуальных разработчиков (*Visual Studio*, облако *Azure*, *Windows Server*, *Parallels Desktop* для *Mac Pro*);

– типы данных имеют фиксированный размер (32-битный *int* и 64-битный *long*), что повышает «мобильность» языка и упрощает программирование, так как вы всегда знаете точно, с чем вы имеете дело;

– автоматическая «сборка мусора», это значит, что нам в большинстве случаев не придётся заботиться об освобождении памяти. Вышеупомянутая общеязыковая среда *CLR* сама вызовет сборщик мусора и очистит память;

– большое количество «синтаксического «сахара» (специальных конструкций, разработанных для понимания и написания кода);

– низкий порог вхождения;

– с помощью *Xamarin* на *C#* можно писать программы и приложения для таких операционных систем, как *iOS*, *Android*, *MacOS* и *Linux* [6, с.38].

Но есть у *C#* и некоторые недостатки:

– приоритетная ориентированность на платформу *Windows*;

– язык бесплатен только для небольших фирм, индивидуальных программистов, стартапов и учащихся;

– инструментарий *C#* позволяет решать широкий круг задач, язык действительно очень мощный и универсальный.

На нём часто разрабатывают:

– веб-приложения;

– игры;

– мобильные приложения для *Android* или *iOS*;

– программы под *Windows*.

Перечень возможностей разработки практически не имеет ограничений благодаря широчайшему набору инструментов и средств. Конечно, всё это можно реализовать при помощи других языков. Но некоторые из них узкоспециализированные, а в некоторых придётся использовать дополнительные инструменты сторонних разработчиков. В *C#* решить широкий круг задач возможно быстрее, проще и с меньшими затратами времени и ресурсов.

2 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ИГРОВОГО СЕТЕВОГО ПРИЛОЖЕНИЯ «ФЕРМА»

2.1 Архитектура приложения «Ферма»

Представление надежной и эффективной архитектуры сетевого приложения является критическим компонентом в разработке современных многопользовательских игр и приложений. Процесс пришествия к пониманию важности архитектуры и структуры приложений был пошаговым, основанным на накопленном опыте, столкновениях с проблемами и поиске эффективных решений для их решения.

Её главная цель – определить структуру, компоненты и взаимодействия между ними в приложении. Она позволяет увидеть «общую картинку» приложения и оценить его целостность, а также обеспечить гибкость и расширяемость приложения в будущем.

Некоторые из основных причин, по которым нужно разрабатывать архитектуру приложения:

– 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.

Декомпозиция часто включает в себя разделение программы на модули, классы, функции и другие элементы, что способствует более эффективной работе разработчиков и повышению читаемости исходного кода.

Таким образом, правильное разбиение задачи на отдельные компоненты приведет к оптимизированию разработки архитектуры и структуры приложения.

Результат разбиения задачи на отдельные компоненты показан на рисунке 2.1.

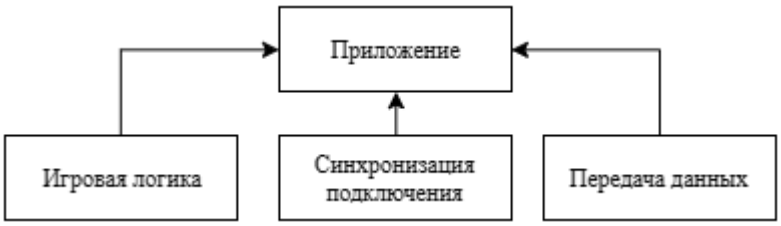

Рисунок 2.1 – Результат разбиения задачи на отдельные компоненты

Игровая логика будет разрабатываться на игровом движке *Unity*. Он будет отвечать за отображение графических элементов, также на нем будут разработаны механики и сценарии игры.

Сетевое взаимодействие между игроками будет обусловливаться их синхронизацией подключения и передачей данных между ними. Синхронизация подключения и передача данных будут разработаны средствами платформы *.Net* и протокола *TCP*.

После этого стоит выбрать шаблон сетевого взаимодействия. Здесь нужно учитывать, что в игре будет небольшое кол-во игроков. В таком случае подойдет шаблон «*P2P*», который относится к категории симметричных клиент-серверных шаблонов.

Симметричность в данном контексте означает отсутствие в сети подразделения на клиентов и серверы. В этом шаблоне одна система выступает и как клиент, и как сервер. Каждая система, также называемая пиром, отправляет запросы другим пирам сети и в то же время получает и обслуживает запросы от других пиров этой сети. Такая схема сильно отличается от традиционной клиент-серверной сети, в которой клиент должен только отправлять запрос и ожидать его обработки сервером.

Таким образом, это будет работать следующим так: пользователь будет выбирать роль сервера или клиента. Если была выбрана роль сервера, то пользователю нужно будет ожидать подключения клиента. Если была выбрана роль клиента, то пользователю нужно будет подключаться к серверу. В результате чего у каждого игрока будет запущена игровая логика и в это время будет

обрабатываться весь пользовательский ввод локально, а результаты ввода будут отправлены другому пользователю.

На рисунке 2.2 представлена архитектура «*P2P*» шаблона.

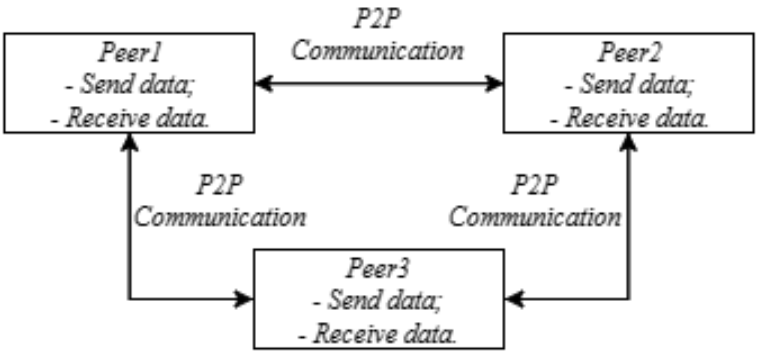

Рисунок 2.2 – «*P2P*» шаблон

Помимо того, что этот шаблон хорошо подходит к сетевым приложениям, в которых количество клиентов мало, он также ещё прост в реализации.

2.2 Структура приложения «Ферма»

После определения архитектуры приложения можно перейти к разработке структуры приложения.

Игрокам нужно будет подключаться и в ходе геймплея обмениваться данными. Таким образом, рационально было бы отделить сетевое взаимодействие от игровой логики. В таком случае, можно осуществлять подключение игроков на одной сцене, а на другой реализовать игровую логику.

После подключения игроков данные о подключении будут использоваться в сцене с игровой логикой для передачи данных по сети. В этом случае нужно как-то сохранять данные о подключении. Здесь можно столкнуться с проблемой того, что при переключении сцен в *Unity* ресурсы сцены будут очищены. В таком случае есть два варианта решения проблемы: сохранять данные в статические переменные или запускать фоновую вспомогательную сцену. Сохранение данных в статических переменных является плохой практикой, так как, в случае, когда полей данных будет много очищать их вручную будет неудобно. В случае с фоновой сценой также будут применяться статические переменные, но, так как данные будут находится в фоновой сцене, *Unity* сам позаботиться об их очистке. В конченом варианте, сохранение данных в отдельной фоновой сцене выглядит неплохо.

В *Unity* последовательностью загрузки сцен управляет сам разработчик. Первым делом будет загружаться сцена для синхронизации подключения

игроков. Чтобы сохранить данные о подключении пользователей нужно гарантировать, что фоновая сцена, с помощью которой данные о подключении будут переноситься и использоваться в другой сцене с игровой логикой, будет запущена первой. Это позволит проверить среду запуска приложения на пригодность прежде, чем запускать остальные сцены.

Схема сцен приложения представлена на рисунке 2.3.

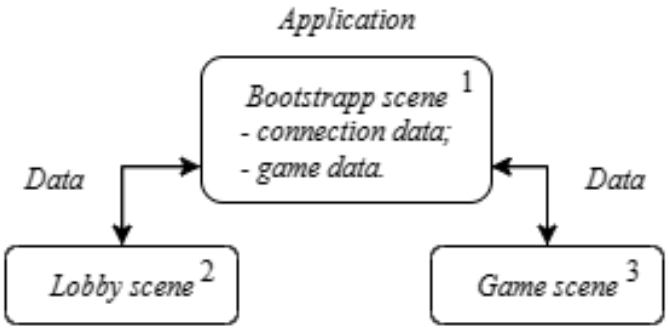

Рисунок 2.3 – Схема сцен приложения

2.2.1 Bootstrapp сцена – это место, где определяется простая сцена, которая содержит только системные объекты игры. Цель состоит в том, чтобы инициализировать основные системы игры, проверить готовность среды и систем к загрузке игры, а затем загрузить основную игру. Такая сцена загружается всего один раз в самом старте и больше никогда.

2.2.2 Lobby сцена – это сцена, которая будет отвечать за синхронизацию подключения игроков и загрузку *Gameplay* сцены. Игроку будет представлена возможность стать «хостом» или клиентом. При ожидании подключения важно сохранять отзывчивость приложения. Ведь это основа того, что оставляет пользователя в приложении во время каких-либо загрузок. Поэтому, во время подключения игрокам будет предоставлена возможность прервать операцию подключения и вернуться в начало *Lobby* сцены.

Lobby сцена будет отвечать за 2 этапа: установление соединения и загрузка уровня. Во время установления соединения со стороны клиента нужно будет ввести строку подключения. В случае с сервером, нужно будет просто ожидать подключения. Во время загрузки уровня будет отображаться условный процесс загрузки, при этом загрузку уровня прервать будет нельзя.

2.2.3 Gameplay сцена – на этой сцене будет рассчитываться вся игровая логика и общение с клиентами. Данные для общения с клиентами будут браться из *Bootstrapp* сцены.

Для того, чтобы обеспечить синхронизацию подключения и отправку данных согласно «*P2P*» шаблону с использованием стека протоколов *TCP/IP*,

необходимо разработать компоненты, отвечающие за отправку данных и подключение «пиров»:

– *TcpBase*;
– *Server*;
– *Client*.
– *User*;
– *Communicator*;
– *PlayerData*;
– *BootSingleton*.

Компонент *TcpBase* будет отвечать за отправку и получения данных с использованием протокола *TCP*. Также будет содержать необходимую логику для хранения информации подключения, отключения и очищение данных соединения пользователей.

Компонент *Client* будет отвечать за подключение к удаленному хосту.

В свою очередь, компонент *Server* будет отвечать за привязку к локальной точке и прослушивание подключений на выбранной локальной точке. Компонент *Client* и *Server* будут наследовать некоторую реализацию компонента *TcpBase*.

На рисунке 2.4 представлена схема компонентов для создания подключения.

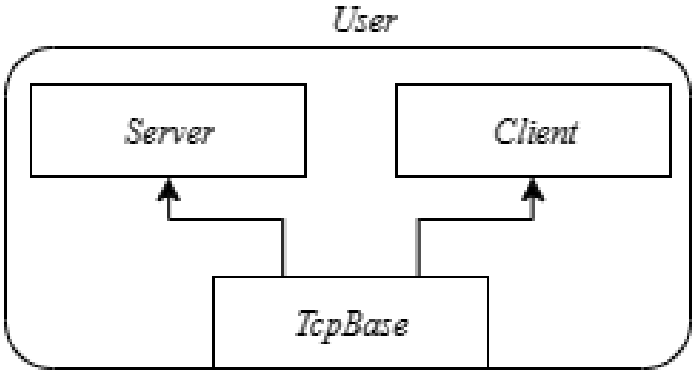

Рисунок 2.4 – Схема компонентов для создания подключения

В «*P2P*» шаблоне нет сущности, которая всегда будет обладать правом на решения любых конфликтных ситуаций. Вместо этого каждый *peer* приходит к решению самостоятельно, просчитывая все локально. Таким образом, результатом создания подключения будет инициализация компонента *User*.

Компонент *User* инициализируется в результате подключения и используется для передачи данных.

Компонент *PlayerData* будет реализовывать структуру данных для хранения данных общения между пользователями. Такая структура данных будет содержать в себе необходимые данные для игровой логики.

Компонент *Communicator* будет отвечать за отправку структуры данных между пользователями в формате *JSON* используя компонент *TcpBase*.

На рисунке 2.5 представлена схема компонентов для общения между пользователями.

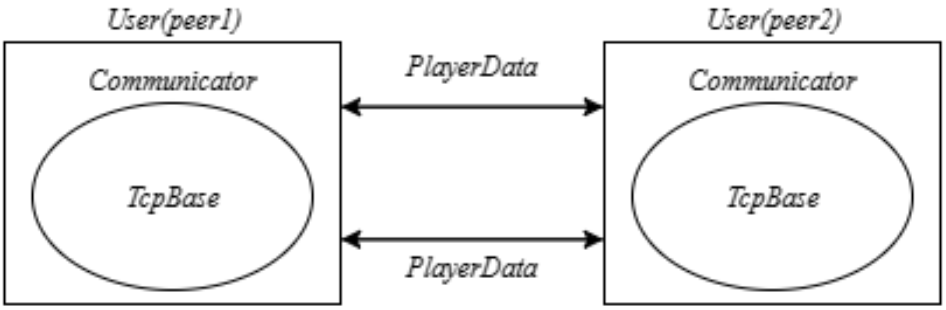

Рисунок 2.5 – Схема компонентов для общения между пользователями

Компонент *BootSingleton* будет отвечать за доступность к компонентам создания подключения и передачи данных. Реализует паттерн проектирования «*Singleton*».

Singleton – это паттерн проектирования, гарантирующий, что у класса будет только один экземпляр. К этому экземпляру будет предоставлена глобальная, то есть доступная из любой части программы, точка доступа. Если попытаться создать новый объект этого класса, то вернётся уже созданный существующий экземпляр.

После разделения задачи на отдельные компоненты можно начать разработку необходимого программного кода, который будет реализовывать все компоненты.

2.3 Описание разработанных классов

В этом подразделе будут описаны классы и скрипты, реализующие разработанные компоненты задачи. Под скриптами будут подразумеваться классы, которые наследуются от *MonoBehaviour*.

Стоит начать с классов для сетевого взаимодействия.

TCPBase(Приложение А, с. 33) – абстрактный класс, предоставляет основные методы для обмена данными между клиентом и сервером по сети. Содержит методы:

– *InitializeTCPSocket*(): создает и возвращает новый экземпляр сокета *TCP*;

– *SendAsync<T>*(): асинхронная отправка данных;

– *RecvAsync<T>*(): асинхронный прием данных;

– *SendFixAsync<T>*(): отправка данных фиксированного размера;

– *RecvFixAsync<T>*(): прием фиксированных данных;

– *Stop*(): завершает соединение с сервером;

– *CheckConnetcion*(): проверяет, подключен ли сокет;

– *StopSocket*(): останавливает сокет и освобождает ресурсы;

– *GetLastError*(): возвращает сообщение об ошибке;

– *GetLocalPoint*() и *GetRemotePoint*(): возвращают строковое представление локальной и удаленной конечных точек.

Client(Приложение А, с. 37) – класс представляет клиентскую сторону приложения и предоставляет методы для подключения и отключения от сервера с использованием базовой функциональности, предоставляемой классом *TCPBase*(Приложение А, с. 33)**.** Конструктор инициализирует объект клиента, устанавливает конечную точку и создает сокет.

Содержит методы:

– *TryConnectAsync*(): реализует асинхронную попытку подключения к серверу, в случае успеха устанавливает локальную и удаленную конечные точки;

– *TryDisconnect*(): предоставляет асинхронную возможность отключения от сервера с использованием события *DisconnectReuseSocket*, что позволяет повторно использовать сокет после отключения, обрабатывает возможные ошибки при отключении;

– *Client* (конструктор): инициализирует объект клиента, устанавливает конечную точку и создает сокет.

– *GetLastError* (унаследован от *TCPBase*): возвращает строку с описанием последней ошибки;

– *GetLocalPoint* (унаследован от *TCPBase*): возвращает строковое представление локальной конечной точки;

– *GetRemotePoint* (унаследован от *TCPBase*): возвращает строковое представление удаленной конечной точки.

Server(Приложение А, с. 38) – класс, который предоставляет функциональность для управления серверным сокетом, привязки к определенной конечной точке, прослушивания входящих подключений, асинхронного принятия подключений, а также проверки и остановки соединений. Конструктор инициализирует объект сервера, устанавливает локальный *IP*-адрес и создает серверный сокет. Содержит методы:

– *SetEndPoint*(): устанавливает конечную точку сервера;

– *TryBindPoint*(): попытка привязать серверный сокет к установленной конечной – точке с обработкой возможных ошибок;

– *Listen*(): запускает прослушивание входящих подключений;

– *TryAcceptAsync*(): асинхронно принимает входящее подключение и устанавливает соответствующие конечные точки;

– *GetLocalIpAddress*(): возвращает локальный *IP*-адрес сервера;

– *Stop*(переопределенный от *TCPBase*): останавливает сокет сервера и, если есть подключение, также останавливает клиентский сокет;

– *CheckConnection* (переопределенный от *TCPBase*): проверяет соединение серверного сокета, а также, если есть подключение, проверяет соединения и клиентского сокета.

JSONStringSpliter(Приложение А, с. 40) – класс предоставляет функциональность для разделения строки, содержащей несколько *JSON*-строк, на отдельные *JSON*-строки. Конструктор *JSONStringSpliter* инициализирует регулярные выражения для разделения *JSON*-строк. Метод *SplitJSONStrings* разделяет строку *JSON* на список отдельных *JSON*-строк. Если присутствует разделитель, используется регулярное выражение для разделения, иначе возвращается исходная строка как единственный элемент списка.

PrefixWriterReader(Приложение А, с. 41) – класс предоставляет функциональность для создания и анализа префикса, который используется для указания размера передаваемых данных. Содержит методы:

– *InitializePrefix*(): инициализирует префикс, указывая размер передаваемых данных;

– *WriteInfoPrefix*(): добавляет информацию к префиксу и возвращает результат как строку.;

– *ReadInfoPrefix*(): считывает информацию из префикса и возвращает размер передаваемых данных. Если формат префикса не соответствует ожидаемому, генерирует исключение.

Теперь стоит описать классы, которые синхронизируют подключение пользователей.

Класс *LobbyConnection*(Приложение А, с. 53) отвечает за работу с подключением в лобби игры. Реализует методы для создания серверного и клиентского соединений, обработку событий и вызов соответствующих обработчиков при изменении статуса подключения. Содержит события:

– *OnCancelServerConnectionEvent*, *OnCancelClientConnectionEvent*:
события отмены соединения на сервере и клиенте соответственно;

– *OnStartCreateConnectionEvent*: событие начала процесса создания соединения;

– *OnCreateConnectionSeccessEvent*: событие успешного создания соединения;

– *OnCreateConnectionFailedEvent*: событие неудачного создания соединения;

– *OnCreateServerEndPointEvent*: событие создания конечной точки сервера;

– *OnCreateConnectionStringFailedEvent*: событие неудачного создания строки подключения;

– *onServerConnectionCreatedEvent*: событие успешного создания серверного соединения;

– *onClientConnectionCreatedEvent*: событие успешного создания клиентского соединения.

Содержит методы:

– *Awake*(): инициализирует экземпляры классов для создания соединений при запуске сцены;

– *OnCreate*(): метод для создания серверного соединения, вызывает соответствующие события в зависимости от результата операции;

– *OnConnect*(): метод для создания клиентского соединения, вызывает соответствующие события в зависимости от результата операции;

– *OnServerBack*(), *OnClientBack*(): методы для завершения соединения на сервере и клиенте соответственно;

– *CancelConnection*()**:** метод для отмены текущего соединения.

Класс *ClientConnectionCreator*(Приложение А, с. 66) отвечает за создание и настройку клиентского соединения. Реализует методы для инициализации конечной точки, создания клиентского соединения, и возвращает результат асинхронной операции. Содержит методы:

– *ClientConnectionCreator*(): конструктор класса, инициализирует экземпляр регулярного выражения для разбора строки конечной точки;

– *InitializeEndPoint*(*string endPoint*): метод для инициализации конечной точки на основе строки *endPoint*, проверяет корректность формата строки, «парсит» *IP*-адрес и порт, и создает объект *IPEndPoint*;

– *CreateClientConnection*(*CancellationToken cancellationToken*): метод для создания клиентского соединения асинхронно, принимает токен отмены и возвращает результат операции, создает экземпляр *Client*, пытается установить соединение с сервером асинхронно с периодической проверкой, при отмене задачи завершает соединение;

– *GetClient*(): метод возвращает созданный клиентский объект *Client*.

Класс *ServerConnectionCreator*(Приложение А, с. 68) ответственен за создание и настройку серверного соединения. Реализует методы для создания сервера, привязки к определенной конечной точке, прослушивания входящих подключений, а также возвращает результат асинхронной операции. Содержит метод создания серверного соединения асинхронно. Принимает токен отмены и делегат для обработки события создания конечной точки сервера. Создает новый экземпляр *Server*, привязывает его к конечной точке, начинает прослушивание

входящих подключений и вызывает соответствующие события. Регистрирует действие по отмене операции. Возвращает результат асинхронной операции – успешное либо неудачное создание соединения.

Класс *LevelLoader*(Приложение А, с. 57) отвечает за загрузку уровня в лобби, контролируя процесс подключения и сигналов загрузки от сервера или клиента. Реализует методы для обработки создания серверного и клиентского соединений, а также проверку сигналов загрузки. Содержит методы:

– *Start*(): метод вызывается при старте объекта, инициализирует поля, подписывается на события подключения и отключения сервера или клиента;

– *OnDisable*(): метод вызывается при выключении объекта, отписывается от событий подключения и отключения сервера или клиента;

– *OnServerConnectionCreated*(): метод, обрабатывающий событие создания серверного соединения, инициализирует сервер и запускает проверку сигналов загрузки асинхронно, в случае успешной проверки сигнала и отсутствия отмены загружает уровень;

– *OnClientConnecitonCreated*(): метод, обрабатывающий событие создания клиентского соединения, инициализирует клиента и запускает проверку сигналов загрузки асинхронно, в случае успешной проверки сигнала и отсутствия отмены загружает уровень;

– *CheckSignal*(*Task checkSignalLoadingTask*): метод для проверки сигнала загрузки асинхронно, вызывает события начала и конца проверки сигнала, Ожидает завершения задачи проверки сигнала и вызывает событие завершения проверки;

– *LoadLevel*(*SceneName sceneName*): метод для асинхронной загрузки уровня по его имени, использует асинхронную операцию *SceneManager.LoadSceneAsync* и ожидает завершения загрузки;

– *OnServerCanceled*(): метод, вызываемый при отмене подключения к серверу. Вызывает общий метод отмены операции;

– *OnClientCanceled*(): метод, вызываемый при отмене подключения к клиенту. Вызывает общий метод отмены операции;

– *Cancel*(): метод для отмены операции, пытается отменить токен отмены и, в случае возникновения исключения, останавливает сервер и клиента.

Класс *LevelSignalChecker*(Приложение А, с. 63) отвечает за проверку сигналов загрузки уровня между сервером и клиентом. Реализует методы для проверки сигнала загрузки от сервера и клиента, а также внутренние методы для инициализации и ожидания сигналов. Содержит методы:

– *CheckServerLevelSignal*(): метод для проверки сигнала загрузки уровня от сервера, инициализирует задачи отправки и приема сигнала, ожидает выполнения одной из них и возвращает результат проверки;

– *CheckClientLevelSignal*(): метод для проверки сигнала загрузки уровня от клиента, инициализирует задачи отправки и приема сигнала, ожидает выполнения одной из них и возвращает результат проверки;

– *InitializeLevelLoadCompliteTask*(): метод для инициализации задач отправки и приема сигнала загрузки уровня, создает задачу отправки, задачу приема и задачу, завершающуюся при получении сигнала или по истечении времени ожидания;

– *WaitForSignalOrReciveSignal*(): метод для ожидания сигнала загрузки уровня или приема сигнала от определенного типа соединения, возвращает результат проверки сигнала загрузки уровня.

Класс *User*(Приложение А, с. 46) представляет собой основной компонент для управления пользователем в контексте соединения. Он отвечает за инициализацию и управление соединением, обработку событий загрузки уровня, а также проверку состояния соединения во время выполнения. Содержит методы:

– *Start*(): метод, вызываемый при старте компонента, регистрирует обработчики событий загрузки уровня и завершения игры;

– *OnDisable*(): метод, вызываемый при выключении компонента, вызывает завершение игры и останавливает коммуникатор;

– *Update*(): метод, вызываемый на каждом кадре. проверяет состояние соединения и вызывает завершение игры в случае разрыва соединения;

– *InitializeUserBase*(): метод для инициализации базы пользователя и установки типа соединения и типа игрока;

– *OnLevelLoaded*(): метод, вызываемый при загрузке уровня, инициализирует коммуникатор, отправляет и принимает сигналы начала коммуникации, и, при успешной проверке, запускает коммуникатор и устанавливает флаг создания соединения;

– *OnApplicationQuit*(): метод, вызываемый при завершении приложения, останавливает сервер или клиент в зависимости от типа соединения;

– *OnGameOver*(): метод, вызываемый при завершении игры, останавливает коммуникатор и сбрасывает флаг создания соединения.

Класс *PlayerData*(Приложение А, с. 49) представляет данные игрока, которые могут сохраняться и использоваться в различных контекстах игры. Он содержит информацию о командах, выполненных командах, несвободных территориях, положении, направлении, состоянии кнопки и деньгах игрока.

3 ВЕРИФИКАЦИЯ И АПРОБАЦИЯ ИГРОВОГО СЕТЕВЕОГО ПРИЛОЖЕНИЯ «ФЕРМА»

3.1 Принцип работы приложения «Ферма»

После запуска игрового приложения на экране появляется меню для создания подключения(далее «лобби»). В классе *LobbyConnetcion*(Приложение А, код *LobbyConnetcion.cs*) реализованы слушатели для обработки нажатия на кнопки, которые есть в лобби. В случае, если будет нажата кнопка «*Create»*, то будет создан экземпляр класса *Server*(Приложение А, код *Server.cs*), который будет ожидать *TCP* подключение, на доступном сетевом интерфейсе и порте. Во время ожидания подключение выведет строку подключения для клиента. Если будет нажата кнопка «*Connect»*, то будет создана экземпляр класса *Client*(Приложение А, код *Client.cs*), который будет ожидать ввода строки подключения для сервера. Когда подключение будет успешно создано, то вызовутся события создания подключения.

Во время ожидания подключения можно нажать на кнопку «*Cancel»*, слушатель которой реализован в классе *LobbyConneciton* (Приложение А, код *LobbyConnetcion.cs*). В таком случае, будет вызвано событие прерывания подключения.

В классе *LevelLoader*(Приложение А, код *LevelLoader.cs*) реализованы слушатели, которые подписаны на события успешного создания соединения между клиентом и сервером. В нем осуществляется асинхронная загрузка уровня. После успешной загрузки уровня вызывается событие успешной загрузки уровня.

Класс *User*(Приложение А, код *User.cs*) реализовывает слушатели загрузки уровня. После загрузки уровня в этом классе посылается сигнал другому пользователю. Сигнал о том, что загружен уровень представляет класс *StartCommunicationSignal*(Приложение А, код *StartCommunicationSignal.cs*), который содержит данные о времени загрузки уровня. После того, как клиенты получили сигнал о загрузке уровня и его времени загрузки, клиенты которые загрузились раньше ожидают загрузки других клиентов. После ожидания других клиентов, в классе *Communicator*(Приложение А, код *Communicator.cs*) вызывается метод *Start*(). Который посылает другому клиенту структуру данных *PlayerData*(Приложение А, код *PlayerData.cs*), а также принимает структуру данных *PlayerData*(Приложение А, код *PlayerData.cs*) от другого клиента.

После загрузки игры класс *PlayersInstaller* создает игровые объекты игроков в зависимости от того, какой тип подключения у клиента. Если тип подключения *Server*, то игрок будет управлять игровым объектом под названием

«*Cat*». Если тип подключения *Client*, то игрок будет управлять игровым объектом «*Racoon*». В случае, если кто-то из игроков закроет приложение – игра завершится и осуществится вызов события окончания игры. После чего пользователь вернется в лобби.

За выход из игры отвечает класс *LobbyConnection*(Приложение А, код *LobbyConnetcion.cs*), который обрабатывает событие нажатия на кнопку «*Exit*».

3.2 Тестирование приложения «Ферма»

Проект *Tests* позволяет протестировать разработанные игровые библиотеки. Для тестирования используются статические методы класса *Assert* и классы модульного тестирования, которые находится в пространстве имен *Microsoft.VisualStudio.Test-Tools.UnitTesting*.

CreateConnectionTests(Приложение А, код *CreateConnectionTests.cs*) – класс для тестирования методов подключения, отключения. А также методов принятия подключений и очищения данных соединения.

TestCommunication(Приложение А, код *TestCommunication.cs*) – класс, который позволяет протестировать методы отправки или получения данных.

TestMovement(Приложение А, код *TestMovement.cs*) – класс, который тестирует методы передвижения игроков. Проверяет метод передвижения подключенного игрока.

Данный проект позволяет проверить, корректно ли работают разработанные классы и их методы и свойства. На рисунке 3.1 представлены результаты тестирования.

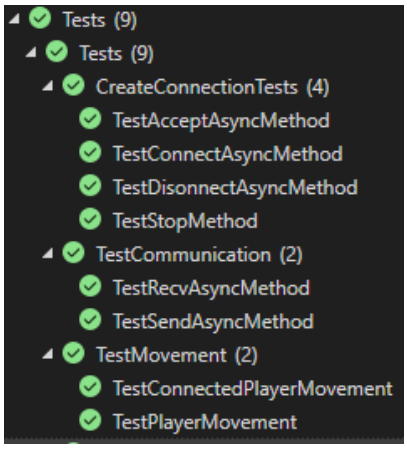

Рисунок 3.1 – Результаты тестирования

Также в процессе тестирования анализируется сетевое взаимодействие между устройствами и объем передаваемых данных. В результате определяется

минимальная пропускная способность сети, приблизительно равная 180 килобитам в секунду.

3.3 Верификация приложения «Ферма»

После запуска программы перед пользователем появляется меню лобби. Лобби содержит кнопки для подключения игроков и выхода из приложения.

На рисунке 3.2 представлено меню лобби.

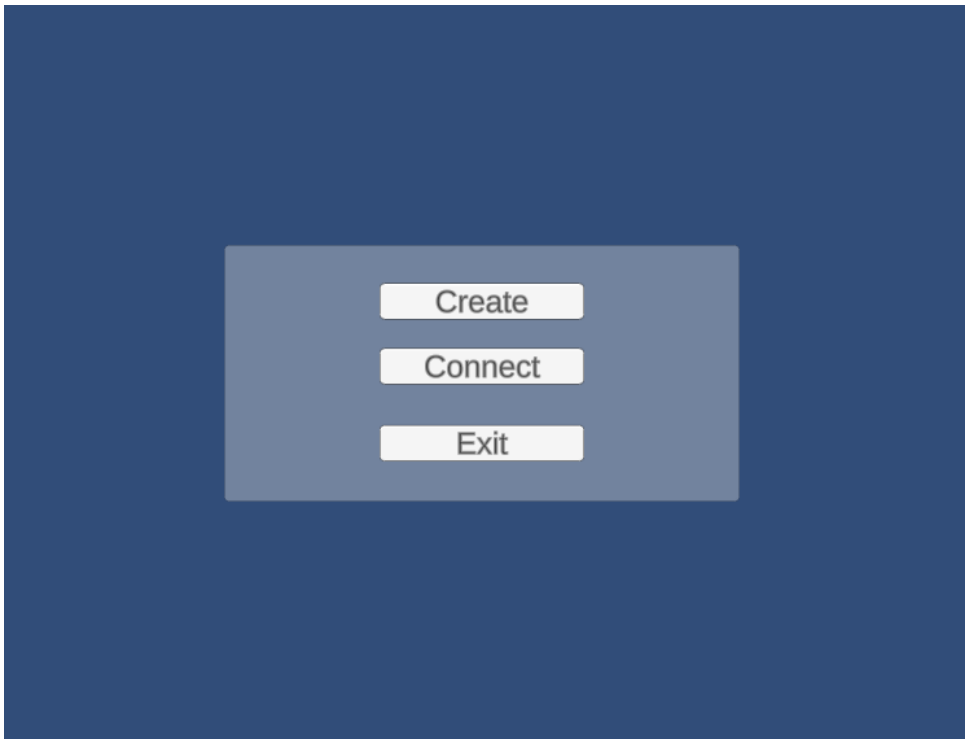

Рисунок 3.2 – Меню лобби

При нажатии на кнопку *Create* игроку придется ожидать подключение. При этом предоставить строку подключения другому игроку.

При игре по сети обоим игрокам необходимо подключение к интернету или находиться одной локальной сети. Для игрока, создающего сеанс игры, необходимо запускать приложение от имени администратора, так как класс *Server* запрашивает доступ к сетевым ресурсам компьютера. Для подключения другому игроку необходимо ввести *IP*-адрес устройства игрока, создающего сеанс игры.

При нажатии на кнопку *Connect* игроку откроется интерфейс для ввода строки подключения. В случае, если формат введенной строки будет неправильным, то подключение не будет начато. Если формат строки будет правильным, но строка подключения будет содержать неверный *IP*-адрес, то

подключение по заданной строке будет продолжаться дот тех пор, пока действие не будет прервано игроком, который нажмет на кнопку *Cancel*.

На рисунке 3.3 представлен интерфейсы создания подключения.

Рисунок 3.3 – Интерфейсы создания подключения

После успешного подключения начнется проверка сигнала передачи данных, во время которого будет возможно прерывание операции подключения.

Когда начнется загрузка игрового уровня отмена операции будет недоступна. При успешном подключении другого пользователя у обоих игроков открывается игровая сцена и начинается игровой цикл.

На рисунке 3.4 представлен вид игры после загрузки игровой сцены от лица двух игроков.

Рисунок 3.4 – Вид игры после загрузки уровня от лица двух игроков

Игроки играют за персонажа «*Cat*», при этом для друг друга они являются персонажем «*Racoon*».

Можно передвигаться только в четырех направлениях. В таблице 3.1 представлено управление игровым персонажем.

Таблица 3.1 – Управление игровым персонажем.

Действие	Персонаж
Движение вверх	*W*
Движение вниз	*S*
Движение вправо	*D*
Движение влево	*A*
Выбрать корзину	*1*
Выбрать лейку	*2*
Выбрать семя пшеницы	*3*
Выбрать семя подсолнуха	*4*
Выбрать семя тыквы	*5*
Выбрать семя кактуса	*6*
Выбрать семя адского нароста	*7*
Использование предмета инвентаря	Нажатие на левую кнопку мыши (ЛКМ)

Цель игрока – набрать определенное количество монет за отведенное количество дней. Монеты игрок зарабатывает, собирая урожай.

На рисунке 3.5 показаны культуры, с которыми может взаимодействовать игрок.

Рисунок 3.5 – Культуры, с которыми может взаимодействовать игрок

Карта разделена на секции, каждая из которых представляет из себя определенную территорию, на которой можно выращивать только определенную культуру. В самом начале лучше будет посадить семена, территория, для посадки которых находится вблизи, иначе, можно потратить время впустую. Приоритет нужно отдавать растению «адский нарост», изображенному на рисунке 3.6 красным цветом, так как за сбор такого растения дается максимально доступное количество монет за сбор культуры.

Вначале игры игроку выдается случайное количество используемых ресурсов.

На рисунке 3.6 продемонстрирован инвентарь игрока с отображением его ресурсов.

Рисунок 3.6 – Отображение инвентаря и ресурсов игрока

Для того, чтобы двигать прогресс прохождения игроку придется использовать инструменты. Игрок, используя инструмент «корзина», может собирать урожай тем самым получать монеты. За сбор тыквы игрок получит 40 монет, за пшеницу 10 монет, за подсолнух 20 монет, за кактус 30 монеты, за адский нарост 50 монет. Таким образом, при наличии семян адского нароста лучше всегда пытаться «пускать их в ход», но также стоит учитывать, что семена адского нароста растут дольше всего. Используя инструмент «лейка» на семена, растения будут расти быстрее.

На рисунке 3.7 показаны инструменты, с помощью которых игрок может взаимодействовать с культурами.

Рисунок 3.7 – Инструменты, с помощью которых игрок может взаимодействовать с культурами

Также геймплей характеризуется сменой дня и ночи.

После того, как пройдет определенное количество дней, игроки увидят результат игры в виде проигрыша или поражения. Результат выигрыша или поражения просчитывается исходя из набранных очков обоих игроков. В случае, если один из игроков набрал по окончанию игры больше очков, чем другой игрок, тот игрок, который набрал больше очков увидит надпись «*Win*», в другом случае «*Lose*». В случае, если один из игроков отключится раньше, чем пройдет определенное количество дней, то оба игрока проиграют.

ЗАКЛЮЧЕНИЕ

В ходе проделанной работы было разработано игровое приложение жанра аркада на двух игроков по сети. Каждый игрок управляет своим персонажем используя периферийные устройства ввода. Их задача заключается в том, чтобы набрать большее количество очков, чем противник.

При выполнении поставленной задачи был проведён аналитический обзор научной литературы и интернет-источников в области компьютерной графики и информационных технологий, был проведён обзор многопользовательских игр жанра «Аркада», особенностей разработки многопользовательских игр и их средств разработки. Были отобраны максимально удачные способы решения поставленной задачи, использованы новые идеи для реализации различных геймплей-механик, а также реализован понятный и удобный интерфейс.

Выбраны средства разработки программного обеспечения и программные интерфейсы для работы с графикой и моделями, которые способствуют быстрой и качественной разработке игровых приложений.

В процессе подготовки к разработке была рассмотрена среда разработки *Unity* и язык программирования *C#*.

В процессе проектирования сетевого взаимодействия с помощью декомпозиции был выбран «*P2P*» шаблон симметричной клиент-серверной модели. Затем был определен алгоритм синхронизации и сетевой коммуникации.

Была разработана структура приложения, учитывающая необходимость обеспечения стабильного взаимодействия между игроками и создания плавного взаимодействия на игровом поле. Этот аспект включал в себя создание эффективных механизмов обмена данными посредством протокола *TCP/IP*, гарантирующих надежное соединение в условиях многопользовательской игры.

Игровое приложение «Ферма» выделяется среди других многопользовательских игр по нескольким ключевым критериям, в частности, графикой и геймплеем. Разработанное в среде *Unity*, это приложение предоставляет пользователю высококачественную *2D*-графику и анимацию, а также обеспечивает поддержку различных платформ и устройств.

Приложение «Ферма» радует разнообразным и увлекательным геймплеем. Здесь каждый игрок может выбирать собственный стиль игры, будь то жадный подход или активные тактики, а также экспериментировать с различными видами культур и инструментов. Эти особенности делают «Ферму» привлекательной и уникальной в мире многопользовательских игр.

Техническая документация к проекту и сам проект проверены системой «Антиплагиат», оригинальность составила 90,52%.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *GamesBeat* [Электронный ресурс]. – 2023. – Режим доступа: *https://venturebeat.com/2019/03/24/the-truth-about-hypercasual-games* – Дата доступа: 10.12.2023;

2. *Wikipedia* [Электронный ресурс]. – 2023. Режим доступа: *https://ru.wikipedia.org/wiki/Super_Meat_Boy* – Дата доступа: 8.12.2023;

3. *Wikipedia* [Электронный ресурс]. – 2023. Режим доступа: *https://ru.wikipedia.org/wiki/Worms_Battlegrounds* – Дата доступа: 8.12.2023

4. *Hocking, J. Unity In Action* : *Multiplatform Game Development in C# / J. Hocking* . – Питер, Москва, Екатеринбург, Вороне, Нижний Новгород, Ростов-на-Дону, Самара, Минск: 2-ое международное издание, 2019.-344;

5. Приемы объектно-ориентированного программирования. Паттерны проектирования / Э. Гамма – СПб.: Питер, 2015.-368; 5. *CLR via C#*. Программирование на платформе *Microsoft .NET Framework 4*;

6. На языке *C#*. / Дж. Рихтер. – СПб: Питер, 2019;

7. Приемы объектно-ориентированного программирования. Паттерны проектирования / Э. Гамма – СПб.: Питер, 2015.

ПРИЛОЖЕНИЕ А
(обязательное)
Исходный код приложения

Класс TcpBase.cs:

```
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading.Tasks;
using UnityEngine;

namespace ClientServer
{
   public abstract class TCPBase
   {
      int _fixedPackageSize = 1024 * 1024;
      public EndPoint EndPoint { get; protected set; }
      public EndPoint LocalEndPoint { get; protected set; }
      public EndPoint RemoteEndPoint { get; protected set; }
      protected Socket ClientSocket;
      protected Exception _error { get; set; }


      PrefixWriterReader _prefix = new PrefixWriterReader(15);
      JSONStringSpliter _jsonSpliter = new JSONStringSpliter();

      protected Socket InitializeTCPSocket()
      {
          return new Socket(AddressFamily.InterNetwork,
            SocketType.Stream, ProtocolType.Tcp);
      }
      public virtual async Task<int> SendAcync<T>(T obj)
      {
         return await SendAsync(ClientSocket, obj);
      }
      public virtual async Task<T> RecvAcync<T>()
      {
         return await RecvAcync<T>(ClientSocket);
      }
      public virtual async Task<int> SendFixAcync<T>(T obj)
      {
         return await SendFixAsync(ClientSocket, obj);
      }
      public virtual async Task<T> RecvFixAcync<T>()
      {
         return await RecvFixAsync<T>(ClientSocket);
      }
      public virtual bool Stop()
      {
```

```
    return StopSocket(ClientSocket);
}
public virtual bool CheckConnection()
{
    return CheckConnectionFor(ClientSocket);
}
protected bool CheckConnectionFor(Socket socket)
{
    return socket.Connected;
}
protected async Task<int> SendAsync<T>(Socket socket, T serializeObject)
{
    try
    {
        string jsonString = JsonConvert.SerializeObject(serializeObject);
        Debug.Log($"Serialize: {jsonString}");
        byte[] jsonBytes = Encoding.UTF8.GetBytes(jsonString);

        _prefix.InitializePrefix(jsonBytes.Length);

        jsonString = _prefix.WriteInfoPrefix(jsonString);
        Debug.Log($"Send: {jsonString}");
        jsonBytes = Encoding.UTF8.GetBytes(jsonString);

        ArraySegment<byte> buffer = new ArraySegment<byte>(jsonBytes);

        int send_bytes = await socket.SendAsync(buffer, SocketFlags.None);

        return send_bytes;

    }
    catch (Exception ex)
    {
        _error = ex;
        return 0;
    }
}

protected async Task<T> RecvAcync<T>(Socket socket)
{
    int jsonSize = 0;
    try
    {
        byte[] prefixBuffer = new byte[_prefix.PrefixSize - 1];
        byte[] buffer;
        ArraySegment<byte> bytes = new ArraySegment<byte>(prefixBuffer);
        int recv_bytes = await socket.ReceiveAsync(bytes, SocketFlags.None);

        if(recv_bytes > 0)
        {
            string prefixStr = Encoding.UTF8.GetString(prefixBuffer);
            Debug.Log($"Recv prefix: {prefixStr}");

            jsonSize = _prefix.ReadInfoPrefix(prefixStr);
```

```
            buffer = new byte[jsonSize];

            int remainderDataLenght = prefixBuffer.Length - _prefix.PrefixLength;

            Array.Copy(prefixBuffer, _prefix.PrefixLength,
               buffer, 0, remainderDataLenght);

            string Str = Encoding.UTF8.GetString(buffer);


            byte[] jsonBuffer = new byte[jsonSize];
            ArraySegment<byte> jsonBytes = new ArraySegment<byte>(jsonBuffer);

            recv_bytes = await socket.ReceiveAsync(jsonBytes, SocketFlags.None);

            if (recv_bytes > 0)
            {
               Array.Copy(jsonBuffer, 0, buffer, remainderDataLenght,
                  buffer.Length - remainderDataLenght);
               string jsonString = Encoding.UTF8.GetString(buffer);
               Debug.Log($"Recv: {jsonString}");
               T deserializeObject = JsonConvert.DeserializeObject<T>(jsonString);
               return deserializeObject;
            }
         }
         return default;
      }
      catch (SocketException ex)
      {
         _error = ex;
         return default;
      }
      catch (JsonReaderException ex)
      {
         _error = ex;
         return default;
      }
      catch(JsonSerializationException ex)
      {
         _error = ex;
         return default;
      }
   }
   protected async Task<T> RecvFixAsync<T>(Socket socket)
   {
      byte[] buffer = new byte[_fixedPackageSize];

      ArraySegment<byte> segmentBuffer = new ArraySegment<byte>(buffer);

      int recvBytes = await socket.ReceiveAsync(segmentBuffer, SocketFlags.None);
      string recvStr;
      if (recvBytes > 0)
      {
         byte[] recv = new byte[recvBytes];
         Array.Copy(buffer, recv, recvBytes);
```

```
            recvStr = Encoding.UTF8.GetString(recv);

            List<string> json_data = _jsonSpliter.SplitJSONStrings(recvStr);

            T deserializeObject = JsonConvert.DeserializeObject<T>(json_data[json_data.Count - 1]);
            return deserializeObject;
        }
        else
        {
            return default;
        }
    }

    protected async Task<int> SendFixAsync<T>(Socket socket, T obj)
    {
        try
        {
            string jsonString = JsonConvert.SerializeObject(obj);
            jsonString += "\n";
            byte[] jsonBytes = Encoding.UTF8.GetBytes(jsonString);

            ArraySegment<byte> segmentBytes = new ArraySegment<byte>(jsonBytes);

            int sendBytes = await socket.SendAsync(segmentBytes, SocketFlags.None);

            return sendBytes;
        }
        catch (Exception ex)
        {
            _error = ex;
            return default;
        }

    }

    protected bool StopSocket(Socket socket)
    {
        try
        {
            socket.Shutdown(SocketShutdown.Both);
            return true;
        }
        catch (SocketException ex)
        {
            _error = ex;
            return false;
        }
        catch(ObjectDisposedException ex)
        {
            _error = ex;
            return false;
        }
        finally
        {
            socket.Close();
```

```
            socket.Dispose();
        }
    }
    public string GetLastError()
    {
        return _error.Message;
    }

    public string GetLocalPoint()
    {
        return LocalEndPoint.ToString();
    }

    public string GetRemotePoint()
    {
        return RemoteEndPoint.ToString();
    }
  }
}
```

Класс Client.cs:

```
using System;
using System.Threading.Tasks;
using System.Net;
using System.Net.Sockets;


namespace ClientServer.Client
{
  public class Client : TCPBase
  {
    public bool IsConnected => ClientSocket.Connected;
    public Client(EndPoint endPoint)
    {
      EndPoint = endPoint;

      ClientSocket = InitializeTCPSocket();
    }

    public async Task<bool> TryConnectAsync()
    {
      try
      {
        await ClientSocket.ConnectAsync(EndPoint);
        LocalEndPoint = ClientSocket.LocalEndPoint;
        RemoteEndPoint = ClientSocket.RemoteEndPoint;
        return true;
      }
      catch (SocketException ex)
      {
        _error = ex;
        return false;
      }
```

```
            catch(ObjectDisposedException ex)
            {
                _error = ex;
                return false;
            }
        }

        public bool TryDicsonnect()
        {
            bool res = false;
            try
            {
                SocketAsyncEventArgs eventArgs = new SocketAsyncEventArgs();
                eventArgs.DisconnectReuseSocket = true;
                res = ClientSocket.DisconnectAsync(eventArgs);
            }
            catch (SocketException ex)
            {
                _error = ex;
            }
            return res;
        }
    }
}
```

Класс Server.cs:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Threading.Tasks;


namespace ClientServer.Server
{
    public class Server : TCPBase
    {
        const int c_backlog = 1;
        Socket _serverSocket;
        public Server()
        {
            IPAddress ip = IPAddress.Parse(GetLocalIpAddress());
            EndPoint = new IPEndPoint(ip, 0);
            _serverSocket = base.InitializeTCPSocket();
        }
        public void SetEndPoint(EndPoint endPoint)
        {
            EndPoint = endPoint;
        }
        public bool TryBindPoint()
        {
            try
            {
                _serverSocket.Bind(EndPoint);
```

```
            LocalEndPoint = _serverSocket.LocalEndPoint;
            return true;
        }
        catch (Exception ex)
        {
            _error = ex;
            return false;
        }
    }
    public bool Listen()
    {
        try
        {
            _serverSocket.Listen(c_backlog);
            return true;

        }
        catch (Exception ex)
        {
            _error = ex;
            return false;
        }
    }

    public async Task<bool> TryAcceptAsync()
    {
        try
        {
            ClientSocket = await _serverSocket.AcceptAsync();
            RemoteEndPoint = ClientSocket.RemoteEndPoint;
            LocalEndPoint = ClientSocket.LocalEndPoint;
            return true;
        }
        catch (Exception ex)
        {
            _error = ex;
            return false;
        }
    }
    public string GetLocalIpAddress()
    {
        string sHostName = Dns.GetHostName();
        IPHostEntry ipE = Dns.GetHostByName(sHostName);
        IPAddress[] IpA = ipE.AddressList;
        return IpA[IpA.Length - 1].ToString();
    }
    public override bool Stop()
    {
        if(ClientSocket == null)
        {
            return StopSocket(_serverSocket);
        }
        else
            return StopSocket(ClientSocket) && StopSocket(_serverSocket);
    }
```

```
        public override bool CheckConnection()
        {
            if(ClientSocket == null)
            {
                return CheckConnectionFor(_serverSocket);
            }
            else
                return CheckConnectionFor(ClientSocket) || CheckConnectionFor(_serverSocket);
        }
    }
}
```

Класс JSONStringSpliter.cs:

```
using System.Collections.Generic;
using System.Text.RegularExpressions;

namespace ClientServer
{
    public class JSONStringSpliter
    {
        const string c_separetorPattern = "}{";
        const string c_jsonDataPattern = "{.*}";

        Regex _separatorRegex;
        Regex _jsonDataRagex;
        public JSONStringSpliter()
        {
            _separatorRegex = new Regex(c_separetorPattern);
            _jsonDataRagex = new Regex(c_jsonDataPattern);
        }

        public List<string> SplitJSONStrings(string local)
        {
            List<string> jsonStrings = new List<string>();
            if(_separatorRegex.IsMatch(local))
            {
                string[] json_Strings = _jsonDataRagex.Split(local);
                return new List<string>(json_Strings);
            }
            else
            {
                jsonStrings.Add(local);
                return jsonStrings;
            }
        }
    }
}
```

Класс PrefixWriteReader.cs:

```
using System;
using System.Text;
using System.Text.RegularExpressions;

namespace ClientServer
{
   public class PrefixWriterReader
   {
      public int PrefixSize { get; private set; }
      public int PrefixLength { get; private set; }
      const string c_prefixBase = "Size:[ ]";
      const string c_sizeGroupName = "Size";
      const string c_sizePattern = @"Size:\[(?'Size'\d+)\s*\]";
      string _prefix;
      StringBuilder _prefixBuilder;
      Regex _sizeRegex;
      public PrefixWriterReader(int prefixSize)
      {
         _prefixBuilder = new StringBuilder();
         _sizeRegex = new Regex(c_sizePattern);
         PrefixSize = prefixSize;
      }
      public void InitializePrefix(int byteLength)
      {
         _prefixBuilder.Clear();
         _prefixBuilder.Append(c_prefixBase);
         _prefix = byteLength.ToString();
         _prefixBuilder = _prefixBuilder.Replace(" ", _prefix);
      }

      public string WriteInfoPrefix(string local)
      {
         _prefixBuilder.Append(local);
         return _prefixBuilder.ToString();
      }
      public int ReadInfoPrefix(string local)
      {
         if (_sizeRegex.IsMatch(local))
         {
            Match match = _sizeRegex.Match(local);
            var sizeData = match.Value;
            PrefixLength = sizeData.Length;
            return int.Parse(match.Groups[c_sizeGroupName].Value);
         }
         else
         {
            throw new Exception("Prefix massage failed to read");
         }
      }
   }
}
```

Класс PerformBootstrapp.cs:

```
using UnityEngine;
using UnityEngine.SceneManagement;

namespace Assets.Code.Bootstrapp
{
   public static class PreformBootstrapp
   {
      const string c_bootstrappSceneName = "Bootstrapp";
      [RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.BeforeSceneLoad)]

[System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.Synchronized
)]
      public static void Execute()
      {
         for(int sceneIndex = 0; sceneIndex < SceneManager.sceneCount; sceneIndex++)
         {
            var loadCandidate = SceneManager.GetSceneAt(sceneIndex);

            if (loadCandidate.name == c_bootstrappSceneName)
               return;
         }
         Debug.Log("Looading bootstrapp!");

         SceneManager.LoadScene(c_bootstrappSceneName, LoadSceneMode.Additive);
      }
   }
}
```

Класс InputService.cs:

```
using Assets.Code.Scripts;
using System;
using UnityEngine;

public class InputService : IDisposable
{
   PlayerControls _playerControls;
   public InputService()
   {
      _playerControls = new PlayerControls();
      _playerControls.Enable();
      GameEvents.OnGameOverEvent += OnGameOver;
   }

   public Vector2 GetMovement()
   {
      return _playerControls.PlayerActions.Movement.ReadValue<Vector2>();
   }

   public bool IsSpawn()
   {
      return _playerControls.PlayerActions.Spawn.WasPerformedThisFrame();
```

```
    }

    public bool IsMouseLeftButton()
    {
        return _playerControls.PlayerActions.Click.WasPerformedThisFrame();
    }
    public Vector2 GetMousePosition()
    {
        return _playerControls.PlayerActions.MousePosition.ReadValue<Vector2>();
    }
    public bool IsCell_1()
    {
        return _playerControls.PlayerActions.Cell_1.WasPerformedThisFrame();
    }
     public bool IsCell_2()
    {
        return _playerControls.PlayerActions.Cell_2.WasPerformedThisFrame();
    }
    public bool IsCell_3()
    {
        return _playerControls.PlayerActions.Cell_3.WasPerformedThisFrame();
    }
    public bool IsCell_4()
    {
        return _playerControls.PlayerActions.Cell_4.WasPerformedThisFrame();
    }
    public bool IsCell_5()
    {
        return _playerControls.PlayerActions.Cell_5.WasPerformedThisFrame();
    }
    public bool IsCell_6()
    {
        return _playerControls.PlayerActions.Cell_6.WasPerformedThisFrame();
    }
    public bool IsCell_7()
    {
        return _playerControls.PlayerActions.Cell_7.WasPerformedThisFrame();
    }
    public void Dispose()
    {
        _playerControls.Disable();
        _playerControls.Dispose();
    }

    public void OnGameOver()
    {
        _playerControls.Disable();
    }
}
```

Класс Communicator.cs:

```
using System;
using System.Threading;
using System.Threading.Tasks;
```

```
using Assets.Code.Scripts.Boot.Data;
using ClientServer;
using Newtonsoft.Json;
using PimDeWitte.UnityMainThreadDispatcher;
using UnityEngine;

namespace Assets.Code.Scripts.Boot.Communication
{
   public class Communicator
   {
      TCPBase _user;
      public static PlayerData SendData { get; set; }
      public static PlayerData RecvData { get; set; }
      int _tick;

      Task _communicateTask;
      CancellationTokenSource _cancellationTokenSource;
      public Communicator(TCPBase user, int tick)
      {
         SendData = new PlayerData();
         RecvData = new PlayerData();
         _tick = tick;
         _user = user;
         _cancellationTokenSource = new CancellationTokenSource();
      }

      public void Start()
      {
         _communicateTask = Task.Factory.StartNew(CommunicateTask, _cancellationTokenSource.Token,
            _cancellationTokenSource.Token);
      }
      async void CommunicateTask(object state)
      {
         CancellationToken ct = (CancellationToken)state;

         while (!ct.IsCancellationRequested)
         {
            try
            {
               RecvData = await CommunicateFix();

               await Task.Delay(_tick);
            }
            catch (JsonReaderException ex)
            {
               Debug.Log(ex.Message);
            }
            catch(Exception ex)
            {
               UnityMainThreadDispatcher.Instance().Enqueue(() =>
               {
                  Debug.Log(ex.Message);
               });
            }

         }
```

```
        }

        public async Task<PlayerData> CommunicateFix()
        {

            int sendBytes = await _user.SendFixAcync(SendData);
            if (sendBytes < 1)
            {
                GameEvents.InvokeGameOverEvent();
                Debug.Log(_user.GetLastError());
            }

            PlayerData recv = await _user.RecvFixAcync<PlayerData>();

            if (recv != null)
            {
                return recv;
            }
            else
            {
                UnityMainThreadDispatcher.Instance().Enqueue(() =>
                {
                    Debug.Log("Recv is Null");
                });
                return default;
            }
        }

        public void Stop()
        {
            _cancellationTokenSource?.Cancel();
            SendData = null;
            RecvData = null;
        }

    }
}
```

Класс StartCommunicationSignal.cs:

```
using System;
namespace Assets.Code.Scripts.Boot.Communication
{
    public class StartCommunicationSignal
    {
        public StartCommunicationSignal() { }
        public DateTime DateTime { get; set; }
    }
}
```

Класс CommunicationEvents.cs:

```
using System;
```

```
namespace Assets.Code.Scripts.Boot.Communication
{
   public class CommunicationEvents
   {
      public static Action OnStartCommunicateEvent;
      public static Action OnDataSendedEvent;
      public static Action oncomplitedAdded;

      public static void InvokeCommunicationEvent()
      {
         OnStartCommunicateEvent?.Invoke();
      }
      public static void InvokeDataSendedIvent()
      {
         OnDataSendedEvent?.Invoke();
      }
      public static void InvokeOnComplitedAdded()
      {
         oncomplitedAdded?.Invoke();
      }
   }
}
```

Класс User.cs:

```
using System;
using System.Threading.Tasks;
using UnityEngine;
using ClientServer;
using Assets.Code.Scripts.Lobby;
using Assets.Code.Scripts.Boot.Communication;
using ClientServer.Server;
using ClientServer.Client;
namespace Assets.Code.Scripts.Boot
{
   public class User : BootSingleton<User>
   {
      const int c_tick = 00005;
      TCPBase _userBase;
      public static DateTime LoadTime;
      public static bool IsConnectionCreated { get; private set; }
      public ConnectionType ConnectionType { get; private set; }
      public PlayerType PlayerType { get; private set; }
      Communicator _communicator;

      private void Start()
      {
         IsConnectionCreated = false;
         LevelLoader.onLevelLoadedEvent += OnLevelLoaded;
         GameEvents.OnGameOverEvent += OnGameOver;
      }
      private void OnDisable()
      {
         GameEvents.InvokeGameOverEvent();
         LevelLoader.onLevelLoadedEvent -= OnLevelLoaded;
```

```
    GameEvents.OnGameOverEvent -= OnGameOver;
    _communicator?.Stop();


}
private void Update()
{
    if(User.IsConnectionCreated)
    {
        switch(ConnectionType)
        {
            case ConnectionType.Server:
                {
                    if(!(_userBase as Server).CheckConnection())
                    {
                        GameEvents.InvokeGameOverEvent();
                    }
                    break;
                }
            case ConnectionType.Client:
                {
                    if(!(_userBase as Client).CheckConnection())
                    {
                        GameEvents.InvokeGameOverEvent();
                    }
                    break;
                }
        }
    }
}
public void InitializeUserBase(TCPBase userBase, ConnectionType connectionType)
{
    _userBase = userBase;
    ConnectionType = connectionType;
    if(connectionType == ConnectionType.Server)
    {
        PlayerType = PlayerType.Player1;
    }
    else
        PlayerType = PlayerType.Player2;
}
public async void OnLevelLoaded()
{
    _communicator = new Communicator
        (_userBase, c_tick);

    bool checkSignalResult = false;
    StartCommunicationSignal signal
    = new StartCommunicationSignal();
    StartCommunicationSignal recvSignal = new StartCommunicationSignal();

    signal.DateTime = DateTime.Now;
    int send_bytes = await _userBase.SendAcync(signal);

    if (send_bytes < 1)
    {
```

```
            Debug.Log(_userBase.GetLastError());
            return;
        }

        recvSignal =
        await _userBase.RecvAcync<StartCommunicationSignal>();

        if (recvSignal != null)
        {
            checkSignalResult = true;
        }
        else
        {
            Debug.Log(_userBase.GetLastError());
            return;
        }

        if (checkSignalResult)
        {
            if (signal.DateTime < recvSignal.DateTime)
            {
                TimeSpan diff = recvSignal.DateTime - signal.DateTime;
                await Task.Delay(diff);
            }
            DateTime loadTime = DateTime.Now;
            _communicator.Start();
            IsConnectionCreated = true;
            CommunicationEvents.InvokeCommunicationEvent();
        }
    }
    private void OnApplicationQuit()
    {
        switch (ConnectionType)
        {
            case ConnectionType.Server:
                {
                    if ((_userBase as Server).Stop())
                    {
                        GameEvents.InvokeGameOverEvent();
                    }
                    break;
                }
            case ConnectionType.Client:
                {
                    if ((_userBase as Client).Stop())
                    {
                        GameEvents.InvokeGameOverEvent();
                    }
                    break;
                }
        }
    }
    void OnGameOver()
    {
        _communicator?.Stop();
        User.IsConnectionCreated = false;
```

```
        }
    }
}
```

Класс PlayerData.cs:

```
using System;
using System.Collections.Generic;
using UnityEngine;

namespace Assets.Code.Scripts.Boot.Data
{
    public class PlayerData
    {
        public List<ItemCommand> ItemCommands;
        public List<ItemCommand> CompletedCommands;
        public List<String> NotFreeTerritoryList;
        public float DirectionX;
        public float DirectionY;
        public PlayerData()
        {
            ItemCommands = new List<ItemCommand>();
            CompletedCommands = new List<ItemCommand>();
            NotFreeTerritoryList = new List<String>();
        }
        public bool IsLeftButton;
        public float Money;
        public float positionX;
        public float positionY;
        public void UpdatePosition(Vector2 position)
        {
            positionX = position.x;
            positionY = position.y;
        }
        public Vector2 GetPosition()
        {
            Vector2 pos = new Vector2(positionX, positionY);
            return pos;
        }
        public void AddItemCommand(ItemCommand itemCommand)
        {
            ItemCommands.Add(itemCommand);
        }
        public void AddComplitedCommand(ItemCommand itemCommand)
        {
          CompletedCommands.Add(itemCommand);
        }
        public void AddNotFreeTerritory(string name)
        {
            NotFreeTerritoryList.Add(name);
        }
        public void SetDirection(Vector2 direction)
        {
            DirectionX = direction.x;
            DirectionY = direction.y;
```

```
        }
        public Vector2 GetDirection()
        {
            return new Vector2(DirectionX, DirectionY);
        }
    }
}
```

Класс SpawnCommand.cs:

```
using System;

namespace Assets.Code.Scripts.Boot.Data
{
    public class ItemCommand
    {
        public Item ObjectType;
        public CommandType CommandType;
        public PlayerType PlayerType;

        public string ParentTerritoryName;
        public override bool Equals(object obj)
        {
            if (obj == null || GetType() != obj.GetType())
            {
                return false;
            }
            ItemCommand command = (ItemCommand)obj;

            return command.ObjectType == ObjectType
                && command.ParentTerritoryName == ParentTerritoryName
                && command.CommandType == CommandType
                && command.PlayerType == PlayerType;
        }

        public override int GetHashCode()
        {
            return HashCode.Combine(ObjectType, ParentTerritoryName, CommandType, PlayerType);
        }
    }
}
```

Класс BootSingleton.cs:

```
using System;
using UnityEngine;

namespace Assets.Code.Scripts.Boot
{
    public class BootSingleton<T> : MonoBehaviour
    {
        public static T Instance { get; private set; }
        public void Awake()
        {
```

```
            if (Instance != null)
            {
                Debug.Log($"Find another instance on {gameObject.name}");
                Destroy(gameObject);
                return;
            }
            Instance = (T)Convert.ChangeType(this, typeof(T));
            DontDestroyOnLoad(gameObject);
            OnAwake();
        }
        protected virtual void OnAwake()
        {

        }
    }
}
```

Класс Enums.cs:

```
namespace Assets.Code
{
    public enum ConnectionType
    {
        Server,
        Client
    }

    public enum SceneName
    {
        Farm
    }

    public enum CommandType
    {
        Spawn,
        Delete
    }
    public enum PlayerType
    {
        Player1,
        Player2
    }

    public enum Item
    {
        Basket,
        Watering,
        Wheat,
        Pumpking,
        SunFlower,
        InfernalGrowth,
        Reed
    }
    public enum GrowStatus
    {
```

```
        Growing,
        Ready
    }

    public enum PlayerMode
    {
        Single,
        Multiple
    }
}
```

Класс GameEvents.cs:

```
using System;
using UnityEngine;

namespace Assets.Code.Scripts
{
    public class GameEvents
    {
        public static event Action<float> OnPickSeedEvent;
        public static event Action OnGameOverEvent;
        public static event Action OnDayStartEvent;
        public static event Action OnNightStartEvent;

        public static void InvokePickSeedEvent(float money)
        {
            OnPickSeedEvent?.Invoke(money);
        }
        public static void InvokeOnDayStartEvent(float intensivity)
        {
            OnDayStartEvent?.Invoke();
        }
        public static void InvokeOnNightStartEvent(float intensivity)
        {
            OnNightStartEvent?.Invoke();
        }
        public static void InvokeGameOverEvent()
        {
            Debug.Log("GAME OVER!");
            OnGameOverEvent?.Invoke();
        }
    }
}
```

Класс LobbyInstaller.cs:

```
using TMPro;
using UnityEngine;
using Zenject;

namespace Assets.Code.Scripts.Lobby
{
    public class LobbyInstaller : MonoInstaller
```

```
    {
        [SerializeField] GameObject StartPanel;
        [SerializeField] GameObject CreatePanel;
        [SerializeField] GameObject ConnectPanel;
        [SerializeField] GameObject ConnectionStringPanel;
        [SerializeField] GameObject ConnectCancelButton;
        [SerializeField] GameObject CreateCancelButton;
        public TextMeshProUGUI ProcessText;
        [SerializeField] GameObject LoadImage;
        public override void InstallBindings()
        {
            Container.Bind<LobbyService>().AsSingle();
            Container.Bind<LobbyConnection>().AsSingle();
            Container.Bind<LevelLoader>().AsSingle();
            Container.Bind<ProgressHandler>().AsSingle();
            BindStartPanel();
            BindConnectPanel();
            BindCreatePanel();
            BindConnectionStringPanel();
            BindProgressUI();
            BindCancelButtons();
        }
        void BindCancelButtons()
        {
            Container.BindInstance(ConnectCancelButton).WithId("ConnectCancelButton");
            Container.BindInstance(CreateCancelButton).WithId("CreateCancelButton");
        }
        void BindStartPanel()
        {
            Container.BindInstance(StartPanel).WithId("StartPanel");
        }
        void BindConnectPanel()
        {
            Container.BindInstance(CreatePanel).WithId("CreatePanel");
        }
        void BindCreatePanel()
        {
            Container.BindInstance(ConnectPanel).WithId("ConnectPanel");
        }
        void BindConnectionStringPanel()
        {
            Container.BindInstance(ConnectionStringPanel).WithId("ConnectionStringPanel");
        }
        void BindProgressUI()
        {
            Container.BindInstance(ProcessText).WithId("ConnectionProgressText");
            Container.BindInstance(LoadImage).WithId("ConnectionLoadImage");
        }
    }
}
```

Класс LobbyConnection.cs:

```
using System;
using System.Threading.Tasks;
```

```
using ClientServer.Client;
using ClientServer.Server;
using UnityEngine;
using System.Threading;
using TMPro;
using Assets.Code.Scripts.Lobby.Connection;

namespace Assets.Code.Scripts.Lobby
{//Работа с подключением
   public class LobbyConnection : MonoBehaviour
   {
      public event Action OnCancelServerConnectionEvent;
      public event Action OnCancelClientConnectionEvent;
      public event Action OnStartCreateConnectionEvent;
      public event Action OnCreateConnectionSeccessEvent;
      public event Action OnCreateConnectionFailedEvent;
      public event Action<string> OnCreateServerEndPointEvent;
      public event Action OnCreateConnectionStringFailedEvent;

      public event Action<Server> onServerConnectionCreatedEvent;
      public event Action<Client> onClientConnectionCreatedEvent;

      [SerializeField] TMP_InputField inputField;

      Server _server;
      Client _client;
      CancellationTokenSource _cancellationTokenSource;
      ClientConnectionCreator _clientConnectionCreator;
      ServerConnectionCreator _serverConnectionCreator;
      Task<bool> _connectionTask;

      private void Awake()
      {
         _clientConnectionCreator = new ClientConnectionCreator();
         _serverConnectionCreator = new ServerConnectionCreator();
      }

      public async void OnCreate()
      {
         OnStartCreateConnectionEvent?.Invoke();

         _cancellationTokenSource
            = new CancellationTokenSource();

         _connectionTask =
            _serverConnectionCreator
            .CreateServerConnection(_cancellationTokenSource.Token,
            OnCreateServerEndPointEvent);

         bool result = await _connectionTask;

         if (result)
         {
            _server = _serverConnectionCreator.GetServer();
            OnCreateConnectionSeccessEvent?.Invoke();
            await Task.Delay(1000);
```

```
            onServerConnectionCreatedEvent?.Invoke(_server);
        }
        else
        {
            _server?.Stop();
            OnCreateConnectionFailedEvent?.Invoke();
            await Task.Delay(1000);
            return;
        }
    }

    public async void OnConnect()
    {
        _cancellationTokenSource
            = new CancellationTokenSource();

        if(!_clientConnectionCreator.InitializeEndPoint(inputField.text))
        {
            OnCreateConnectionStringFailedEvent?.Invoke();
            return;
        }

        _connectionTask =
            _clientConnectionCreator
            .CreateClientConnection(_cancellationTokenSource.Token);

        OnStartCreateConnectionEvent?.Invoke();

        await Task.Delay(1000);

        bool result = await _connectionTask;
        if (result)
        {
            _client = _clientConnectionCreator.GetClient();
            OnCreateConnectionSeccessEvent?.Invoke();
            onClientConnectionCreatedEvent?.Invoke(_client);
            await Task.Delay(1000);
            Debug.Log($"Подключился к {_client.GetRemotePoint()}");
        }
        else
        {
            _client?.Stop();
            OnCreateConnectionFailedEvent?.Invoke();
            await Task.Delay(1000);
            Debug.Log("Подключение не удалось!");
        }
    }
    public void OnServerBack()
    {
        _server?.Stop();
        CancelConnection(OnCancelServerConnectionEvent);
    }
    public void OnClientBack()
    {
        _client?.Stop();
        CancelConnection(OnCancelClientConnectionEvent);
```

```
        }

        void CancelConnection(Action cancelConnectionEvent)
        {
            try
            {
                _cancellationTokenSource?.Cancel();
            }
            catch (AggregateException)
            {
            }
            cancelConnectionEvent?.Invoke();
        }
    }
}
```

Класс LobbyService.cs:

```
using System.Threading.Tasks;
using UnityEngine;
using Zenject;


namespace Assets.Code.Scripts.Lobby
{
    public class LobbyService : MonoBehaviour
    {
        GameObject _startPanel;
        GameObject _createPanel;
        GameObject _connectPanel;
        GameObject _createConnectionStringPanel;
        LobbyConnection _connection;
        GameObject _connectCancelButton;
        [Inject]
        public void Constructor(
            [Inject(Id = "StartPanel")] GameObject startPanel,
            [Inject(Id = "CreatePanel")] GameObject createPanel,
            [Inject(Id = "ConnectPanel")] GameObject connectPanel,
            [Inject(Id = "ConnectionStringPanel")] GameObject connectionStringPanel,
            [Inject(Id = "ConnectCancelButton")] GameObject connectCancelButton,
            [Inject(Id = "CreateCancelButton")] GameObject createCancelButton
            )
        {
            _startPanel = startPanel;
            _createPanel = createPanel;
            _connectPanel = connectPanel;
            _createConnectionStringPanel = connectionStringPanel;
            _connectCancelButton = connectCancelButton;
        }

        private void Start()
        {
            _connection = GetComponent<LobbyConnection>();
            _connection.OnCancelClientConnectionEvent += ConnectBack;
            _connection.OnCancelServerConnectionEvent += CreateBack;
```

```
        }
        private void OnDisable()
        {
            _connection.OnCancelClientConnectionEvent -= ConnectBack;
            _connection.OnCancelServerConnectionEvent -= CreateBack;
        }
        public void OnConnect()
        {
            _startPanel.SetActive(false);
            _connectPanel.SetActive(true);
            _createConnectionStringPanel.SetActive(true);
        }
        public void OnCreate()
        {
            _startPanel.SetActive(false);
            _createPanel.SetActive(true);
        }
        public void OnExit()
        {
            Application.Quit();
        }
        public async void ConnectBack()
        {
            await Task.Delay(1000);
            _connectPanel?.SetActive(false);
            _startPanel?.SetActive(true);
        }
        public async void CreateBack()
        {
            _createPanel?.SetActive(false);
            await Task.Delay(1000);
            _startPanel?.SetActive(true);
        }
        public void OnBackAtCreateConnectionStringPanel()
        {
            _createConnectionStringPanel.SetActive(false);
            _connectPanel.SetActive(false);
            _startPanel.SetActive(true);
        }
        public void OnContinue()
        {
            _createConnectionStringPanel.SetActive(false);
            _connectCancelButton.SetActive(true);
        }
    }
}
```

Класс LevelLoader.cs:

```
using System;
using System.Threading.Tasks;
using UnityEngine;
using ClientServer.Client;
using ClientServer.Server;
using UnityEngine.SceneManagement;
```

```
using Assets.Code.Scripts.Boot;
using Assets.Code.Scripts.Lobby.LoadingLevel;
using System.Threading;
using PimDeWitte.UnityMainThreadDispatcher;

namespace Assets.Code.Scripts.Lobby
{
   public class LevelLoader : MonoBehaviour
   {
      public event Action OnCanceledOrFailedLoadLevelSignalEvent;
      public event Action OnStartCheckLoadLevelSignalEvent;
      public event Action OnStopCheckLoadLevelSignalEvent;
      public static Action onLevelLoadedEvent { get; set; }
      LobbyConnection _lobbyConnection;
      SceneName _sceneName;
      bool _signalResult;

      LevelSignalChecker _signalChecker;

      CancellationTokenSource _cancellationTokenSource;
      Server _server;
      Client _client;

      private void Start()
      {
         _signalResult = false;
         _lobbyConnection = GetComponent<LobbyConnection>();
         _sceneName = SceneName.Farm;
         _lobbyConnection.onClientConnectionCreatedEvent
            += OnClientConnecitonCreated;
         _lobbyConnection.onServerConnectionCreatedEvent
            += OnServerConnectionCreated;
         _lobbyConnection.OnCancelClientConnectionEvent
            += OnClientCanceled;
         _lobbyConnection.OnCancelServerConnectionEvent
            += OnServerCanceled;
         _signalChecker = new LevelSignalChecker();
      }
      private void OnDisable()
      {
         _lobbyConnection.onClientConnectionCreatedEvent
            -= OnClientConnecitonCreated;
         _lobbyConnection.onServerConnectionCreatedEvent
            -= OnServerConnectionCreated;
         _lobbyConnection.OnCancelClientConnectionEvent
            -= OnClientCanceled;
         _lobbyConnection.OnCancelServerConnectionEvent
            -= OnServerCanceled;
      }
      public async void OnServerConnectionCreated(Server server)
      {
         _cancellationTokenSource = new CancellationTokenSource();
         _server = server;
         User.Instance.InitializeUserBase(server, ConnectionType.Server);
         Task checkLevelLoadingTask =
            Task.Run(async () =>
```

```
        {
            _signalResult = await _signalChecker.CheckServerLevelSignal(server,
                _cancellationTokenSource.Token);
        }, _cancellationTokenSource.Token);

    await CheckSignal(checkLevelLoadingTask);

    if (!_cancellationTokenSource.Token.IsCancellationRequested && _signalResult)
    {
        await LoadLevel(_sceneName);
        onLevelLoadedEvent?.Invoke();
    }
    else
    {
        _server?.Stop();
        OnCanceledOrFailedLoadLevelSignalEvent?.Invoke();
    }
}

public async void OnClientConnecitonCreated(Client client)
{
    _cancellationTokenSource = new CancellationTokenSource();
    _client = client;
    User.Instance.InitializeUserBase(client, ConnectionType.Client);
    Task checkLevelLoadingTask =
        Task.Run(async () =>
        {
            _signalResult = await _signalChecker.CheckClientLevelSignal(client,
                _cancellationTokenSource.Token);
        }, _cancellationTokenSource.Token);

    await CheckSignal(checkLevelLoadingTask);
    if (!_cancellationTokenSource.Token.IsCancellationRequested && _signalResult)
    {
        await LoadLevel(_sceneName);
        onLevelLoadedEvent?.Invoke();
    }
    else
    {
        _client?.Stop();
        OnCanceledOrFailedLoadLevelSignalEvent?.Invoke();
    }
}
async Task CheckSignal(Task checkSignalLoadingTask)
{
    OnStartCheckLoadLevelSignalEvent?.Invoke();
    await Task.Delay(1000);
    await checkSignalLoadingTask.ContinueWith((compliteTask) =>
    {
        UnityMainThreadDispatcher.Instance().Enqueue(() =>
        {
            OnStopCheckLoadLevelSignalEvent?.Invoke();
        });
    });

}
```

```
        async Task LoadLevel(SceneName sceneName)
        {
            var tcs = new TaskCompletionSource<bool>();

            AsyncOperation asyncLoad =
                SceneManager.LoadSceneAsync(sceneName.ToString());

            asyncLoad.completed += (operation) =>
            {
                tcs.SetResult(true);
            };

            while (asyncLoad.isDone == false)
            {
                Debug.Log("Загрузка уровня!");
                await Task.Delay(1);
            }

            await tcs.Task;
        }
        public void OnServerCanceled()
        {
            Cancel();
        }
        public void OnClientCanceled()
        {
            Cancel();
        }
        void Cancel()
        {
            try
            {
                _cancellationTokenSource?.Cancel();
            }
            catch (AggregateException ex)
            {
                Debug.Log(ex.Message);
            }
            finally
            {
                _server?.Stop();
                _client?.Stop();
            }
        }
    }
}
```

Класс ProgressHandler.cs:

```
using System.Threading.Tasks;
using TMPro;
using UnityEngine;
using Zenject;

namespace Assets.Code.Scripts.Lobby
```

```
{
    public class ProgressHandler : MonoBehaviour
    {
        const string c_ConnectionText = "Waiting for connection...";
        const string c_SuccsessConnectionText = "Connection is established";
        const string c_ConnectionFailedText = "Connection failed";
        const string c_CanceledText = "Canceled...";
        const string c_LoadLevelText = "Loading level ...";
        const string c_CheckText = "Check signal ...";
        const string c_CheckSignalFailedText = "Check signal failed!";
        const string c_FailedCreateConnectionStringText = "Error!";

        LobbyConnection _lobbyConnection;
        LevelLoader _levelLoader;
        TextMeshProUGUI _processText;
        [SerializeField] TextMeshProUGUI _endPointText;
        GameObject _loadImage;
        GameObject _connectCancelButton;
        GameObject _createCancelButton;
        GameObject _connectionStringPanel;
        [Inject]
        public void Constructor(
            [Inject(Id = "ConnectionProgressText")] TextMeshProUGUI progressText,
            [Inject(Id = "ConnectionLoadImage")] GameObject loadImage,
            [Inject(Id = "ConnectCancelButton")] GameObject connectCancelButton,
            [Inject(Id = "CreateCancelButton")] GameObject createCancelButton,
            [Inject(Id = "ConnectionStringPanel")] GameObject connectionStringPanel)
        {
            _processText = progressText;
            _loadImage = loadImage;
            _connectCancelButton = connectCancelButton;
            _createCancelButton = createCancelButton;
            _connectionStringPanel = connectionStringPanel;
        }

        private void Start()
        {
            _levelLoader = GetComponent<LevelLoader>();
            _lobbyConnection = GetComponent<LobbyConnection>();
            _lobbyConnection.OnStartCreateConnectionEvent
                += OnStartCreateConnection;
            _lobbyConnection.OnCancelClientConnectionEvent
                += OnCancelClientConnection;
            _lobbyConnection.OnCancelServerConnectionEvent
                += OnCancelServerConnection;
            _lobbyConnection.OnCreateServerEndPointEvent +=
                OnCreateServerEndPoint;
            _lobbyConnection.OnCreateConnectionStringFailedEvent +=
                OnCreateConnectionStringFailedEvent;
            _levelLoader.OnCanceledOrFailedLoadLevelSignalEvent
                += OnCanceledOrFailedLoadLevelSignal;
            _levelLoader.OnStartCheckLoadLevelSignalEvent
                += OnStartCheckLoadLevelSignal;
            _levelLoader.OnStopCheckLoadLevelSignalEvent
                += OnStopCheckLoadLevelSignal;
            _lobbyConnection.OnCreateConnectionSeccessEvent
```

```
        += OnCreateConnectionSeccess;
    _lobbyConnection.OnCreateConnectionFailedEvent
        += OnCreateConnectionFailed;
}

private void OnDisable()
{
    _lobbyConnection.OnStartCreateConnectionEvent
        -= OnStartCreateConnection;
    _lobbyConnection.OnCancelClientConnectionEvent
        -= OnCancelClientConnection;
    _lobbyConnection.OnCancelServerConnectionEvent
        -= OnCancelServerConnection;
    _lobbyConnection.OnCreateServerEndPointEvent -=
        OnCreateServerEndPoint;
    _lobbyConnection.OnCreateConnectionSeccessEvent
        -= OnCreateConnectionSeccess;
    _lobbyConnection.OnCreateConnectionFailedEvent
        -= OnCreateConnectionFailed;
    _lobbyConnection.OnCreateConnectionStringFailedEvent -=
        OnCreateConnectionStringFailedEvent;
    _levelLoader.OnCanceledOrFailedLoadLevelSignalEvent
        -= OnCanceledOrFailedLoadLevelSignal;
    _levelLoader.OnStartCheckLoadLevelSignalEvent
        -= OnStartCheckLoadLevelSignal;
    _levelLoader.OnStopCheckLoadLevelSignalEvent
        -= OnStopCheckLoadLevelSignal;
}
public void OnCanceledOrFailedLoadLevelSignal()
{
    _connectCancelButton.SetActive(true);
    _createCancelButton.SetActive(true);
    _processText.gameObject.SetActive(true);
    _processText.text = c_CheckSignalFailedText;
    Task.Delay(1000);
    _processText.gameObject.SetActive(false);
}
public void OnStartCheckLoadLevelSignal()
{
    _endPointText.gameObject.SetActive(false);
    _processText.text = c_CheckText;
}
public void OnStopCheckLoadLevelSignal()
{
    _connectCancelButton.SetActive(false);
    _createCancelButton.SetActive(false);
    _processText.text = c_LoadLevelText;
    _processText.gameObject.SetActive(false);
    _loadImage.SetActive(false);
}
public async void OnCancelServerConnection()
{
    _processText.text = c_CanceledText;
    await Task.Delay(1000);
    DeactivateProgressUI();
}
```

```
        public async void OnCancelClientConnection()
        {
            _processText.text = c_CanceledText;
            await Task.Delay(1000);
            DeactivateProgressUI();
        }
        public void OnStartCreateConnection()
        {
            ActivateProgressUI();
            _processText.text = c_ConnectionText;
        }
        public void OnCreateConnectionSeccess()
        {
            _processText.text = c_SuccsessConnectionText;
        }
        public void OnCreateConnectionFailed()
        {
            _processText.text = c_ConnectionFailedText;
            _processText.gameObject.SetActive(false);

        }
        public void OnCreateServerEndPoint(string endPoint)
        {
            _endPointText.gameObject.SetActive(true);
            _endPointText.text = endPoint;
        }
        public async void OnCreateConnectionStringFailedEvent()
        {
            _processText.gameObject.SetActive(true);
            _processText.text = c_FailedCreateConnectionStringText;
            await Task.Delay(1000);
            _processText.gameObject.SetActive(false);
            _connectionStringPanel.SetActive(true);
            _connectCancelButton.SetActive(false);
        }

        void ActivateProgressUI()
        {
            _processText.gameObject.SetActive(true);
            _loadImage.SetActive(true);
        }
        void DeactivateProgressUI()
        {
            _processText.gameObject.SetActive(false);
            _loadImage.SetActive(false);
        }
    }
}
```

Класс LevelSignalChecker.cs:

```
using ClientServer;
using ClientServer.Client;
using ClientServer.Server;
using System;
```

```
using System.Threading;
using System.Threading.Tasks;

namespace Assets.Code.Scripts.Lobby.LoadingLevel
{
    public class LevelSignalChecker
    {
        const float c_waitLoadLevelSignalTime = 5f;
        Task<int> _sendLoadLevelSignalTask;
        Task<LoadLevelSignal> _recvLoadLevelSignalTask;
        Task<Task> _returnComplitedTask;
        bool _result;

        public async Task<bool> CheckServerLevelSignal(Server server,
            CancellationToken ct)
        {
            _result = false;
            ct.Register(() =>
            {
                ct.ThrowIfCancellationRequested();
                server.Stop();
            });

            LoadLevelSignal loadLevelSignal
                = new LoadLevelSignal(ConnectionType.Server);

            InitializeLevelLoadCompliteTask(
                loadLevelSignal,
                server);

            int sendBytes = await _sendLoadLevelSignalTask;

            if (sendBytes < 1)
            {
                return false;
            }

            _result = await WaitForSignalOrReciveSignal(ConnectionType.Client);

            return _result;
        }

        public async Task<bool> CheckClientLevelSignal(Client client,
            CancellationToken ct)
        {
            _result = false;
            ct.Register(() =>
            {
                ct.ThrowIfCancellationRequested();
                client.Stop();
            });

            LoadLevelSignal loadLevelSignal
                = new LoadLevelSignal(ConnectionType.Client);

            InitializeLevelLoadCompliteTask(
```

```
        loadLevelSignal,
        client);

    int sendBytes = await _sendLoadLevelSignalTask;
    //return false;
    ///если нужно протестировать случай, когда
    ///клиент не отправил сигнал
    if (sendBytes < 1)
    {
        return false;
    }
    _result = await WaitForSignalOrReciveSignal(ConnectionType.Server);

    return _result;
}

void InitializeLevelLoadCompliteTask(LoadLevelSignal loadLevelSignal,
    TCPBase user)
{
    _sendLoadLevelSignalTask = user.SendAcync(loadLevelSignal);
    _recvLoadLevelSignalTask = user.RecvAcync<LoadLevelSignal>();

    _returnComplitedTask = Task.WhenAny(
        _recvLoadLevelSignalTask,
        Task.Delay(TimeSpan.FromSeconds(c_waitLoadLevelSignalTime)
        ));
}

async Task<bool> WaitForSignalOrReciveSignal(ConnectionType connectionType)
{
    bool result = false;

    Task complitedTask = await _returnComplitedTask;

    switch (complitedTask)
    {
        case Task<LoadLevelSignal> recvTask:
            {
                LoadLevelSignal levelSignal = recvTask.Result;

                if (levelSignal != null &
                    levelSignal.ConnectionType == connectionType)
                {
                    result = true;
                }
                else
                    result = false;
                break;
            }
        default:
            {
                result = false;
                break;
            }
    }
    return result;
```

```
    }
  }
}
```

Класс LoadLevelSignal.cs:

```
namespace Assets.Code.Scripts.Lobby
{
  public class LoadLevelSignal
  {
    public ConnectionType ConnectionType { get; private set; }
    public LoadLevelSignal(ConnectionType connectionType)
    {
      ConnectionType = connectionType;
    }
  }
}
```

Класс ClientConnectionCreator.cs:

```
using System.Net;
using System.Threading;
using System.Threading.Tasks;
using ClientServer.Client;
using UnityEngine;
using System.Text.RegularExpressions;

namespace Assets.Code.Scripts.Lobby.Connection
{
  public class ClientConnectionCreator
  {
    const string c_endPointPattern
      = @"^(?'IP'\d+\.\d+\.\d+\.\d+):(?'Port'\d+)$";
    Regex _endPointRegex;
    EndPoint _endPoint;
    Client _client;
    public ClientConnectionCreator()
    {
      _endPointRegex = new Regex(c_endPointPattern);
    }
    public bool InitializeEndPoint(string endPoint)
    {
      int port;
      string ip;
      if (_endPointRegex.IsMatch(endPoint))
      {
        Match match = _endPointRegex.Match(endPoint);
        ip = match.Groups["IP"].Value;
        if (!int.TryParse(match.Groups["Port"].Value, out port))
        {
          return false;
        }
        else
        {
```

```
            IPAddress iPAddress;
            if (!IPAddress.TryParse(ip, out iPAddress))
            {
                return false;
            }
            else
            {
                if (port > IPEndPoint.MinPort && port < IPEndPoint.MaxPort)
                {
                    _endPoint = new IPEndPoint(iPAddress, port);
                    return true;
                }
                else
                    return false;
            }
        }
    }
    else
        return false;
}
public async Task<bool> CreateClientConnection
    (CancellationToken cancellationToken)
{
    CancellationToken ct = cancellationToken;

    ct.ThrowIfCancellationRequested();

    _client = new Client(_endPoint);

    Debug.Log($"Connect to : {_client.EndPoint}");

    bool result = false;

    ct.Register(() =>
    {
        _client.Stop();
    });

    Task waitConnectionTask = Task.Run(async () =>
    {
        while (true)
        {
            if (ct.IsCancellationRequested == true)
            {
                result = false;
                break;
            }
            if (await _client.TryConnectAsync())
            {
                result = true;
                Debug.Log($"Подключился к {_client.EndPoint}");
                break;
            }

            await Task.Delay(1000);
        }
```

```
        }, ct);

        await waitConnectionTask;

        return result;
      }

      public Client GetClient()
      {
        return _client;
      }
    }
}
```

Класс ServerConnectionCreator.cs:

```
using ClientServer.Server;
using System;
using System.Threading;
using System.Threading.Tasks;
using UnityEngine;

namespace Assets.Code.Scripts.Lobby
{
  public class ServerConnectionCreator
  {
    Server _server;
    public async Task<bool> CreateServerConnection
      (CancellationToken cancellationToken,
      Action<string> onCreateServerEndpoint)
    {
      bool result = false;
      _server?.Stop();
      _server = new Server();
      if (!_server.TryBindPoint())
      {
        Debug.Log(_server.GetLastError());

        return _server.Stop();
      }
      onCreateServerEndpoint?.Invoke(_server.GetLocalPoint());
      if (!_server.Listen())
      {
        Debug.Log(_server.GetLastError());
        return false;
      }

      Debug.Log($"Слушаю на {_server.EndPoint}");

      cancellationToken.Register(() =>
      {
        result = false;
        _server.Stop();
        Debug.Log("Canceled in register!");
        cancellationToken.ThrowIfCancellationRequested();
```

```
        });

        result = await _server.TryAcceptAsync();
        if(result)
            Debug.Log($"Подключение от {_server.GetRemotePoint()}");

        return result;
      }

      public Server GetServer()
      {
        return _server;
      }
    }
}
```

Класс SeedSO.cs:

```
using Assets.Code;
using UnityEngine;
[CreateAssetMenu(fileName = "Seed", menuName = "ScriptableObjects/Seed")]
public class SeedSO : ScriptableObject
{
   public Item SeedType;
   public float GrowingTime;
   public Sprite GrowingUpSprite;
   public float Money;
}
```

Класс Seed.cs:

```
using Assets.Code.Scripts.Gameplay.PlantingTerritory;
using UnityEngine;

namespace Assets.Code.Scripts.Gameplay
{
   [RequireComponent(typeof(SpriteRenderer))]
   public class Seed : MonoBehaviour
   {
      [SerializeField] Item Type;
      public PlayerType ParentPlayer;
      protected float GrowingTime;
      protected Sprite GrowingUpSprite;
      public float Money { get; private set; }
      SpriteRenderer _sprite;
      Color _defaultColor;
      public GrowStatus Status { get; private set; }
      float _currentTime;

      PlantTerritory _parent;

      private void Start()
      {
        _currentTime = 0;
```

```
    Status = GrowStatus.Growing;
    _sprite = GetComponent<SpriteRenderer>();
    _defaultColor = _sprite.color;
}

private void Update()
{
    if (Status == GrowStatus.Growing)
    {
        _currentTime += Time.deltaTime;

        if (_currentTime >= GrowingTime)
        {
            Status = GrowStatus.Ready;
            _currentTime = 0;
            _sprite.sprite = GrowingUpSprite;
        }
    }
}

public void Initialize(SeedSO seedSO, PlantTerritory parent, PlayerType playerType)
{
    GrowingTime = seedSO.GrowingTime;
    GrowingUpSprite = seedSO.GrowingUpSprite;
    Money = seedSO.Money;
    ParentPlayer = playerType;
    _parent = parent;
}
public void Initialize(SeedSO seedSO)
{
    GrowingTime = seedSO.GrowingTime;
    GrowingUpSprite = seedSO.GrowingUpSprite;
    Money = seedSO.Money;
}

public void Boost()
{
    _currentTime += 1f;
}
public Item GetSeedType()
{
    return Type;
}
public void OnPick()
{
    _parent?.SetEmpty(true);
}

public string GetParentTerritoryName()
{
    return _parent.name;
}
private void OnMouseEnter()
{
    _sprite.color = new Color(_sprite.color.r, _sprite.color.g,
        _sprite.color.b, _sprite.color.a / 2);
```

```
        }
        private void OnMouseExit()
        {
            _sprite.color = _defaultColor;
        }
    }
}
```

Класс SeedPrefabService.cs:

```
using System.Collections.Generic;
namespace Assets.Code.Scripts.Gameplay
{
    public class SeedsService
    {
        Dictionary<Item, Seed> _seedsPrefabs;

        Dictionary<Item, SeedSO> _seedsSoDictionary;

        public SeedsService(Dictionary<Item, Seed> seedsPrefabs, Dictionary<Item, SeedSO> seedsSoDictionary)
        {
            _seedsPrefabs = seedsPrefabs;
            _seedsSoDictionary = seedsSoDictionary;
        }

        public Seed GetSeedFor(Item seed)
        {
            return _seedsPrefabs[seed];
        }
        public SeedSO GetSeedSOFor(Item seed)
        {
            return _seedsSoDictionary[seed];
        }
    }
}
```

Класс CursorRay.cs:

```
using Assets.Code.Scripts.Boot;
using Assets.Code.Scripts.Boot.Communication;
using System;
using UnityEngine;
using Zenject;

namespace Assets.Code.Scripts.Gameplay
{
    public class CursorRay : MonoBehaviour
    {
        Camera _camera;
        InputService _inputService;
        public RaycastHit2D Hit;
        public string HitObject;
        public event Action<RaycastHit2D> OnHitTerritoryEvent;
        public event Action<RaycastHit2D> OnHitSeedEvent;
```

```
        public LayerMask SeedMask;
        public LayerMask TerritoryMask;
        [Inject]
        public void Construcotr(InputService inputService)
        {
            _inputService = inputService;
        }

        private void Start()
        {
            _camera = Camera.main;
        }

        private void Update()
        {
            if (_inputService.IsMouseLeftButton())
            {
                Ray ray = _camera.ScreenPointToRay(_inputService.GetMousePosition());

                CheckIntersection(ray, TerritoryMask, OnHitTerritoryEvent);
                CheckIntersection(ray, SeedMask, OnHitSeedEvent);

                HandleCommunicationData();
            }
            void CheckIntersection(Ray ray, LayerMask mask, Action<RaycastHit2D> hitEvent)
            {
                RaycastHit2D hit = Physics2D.GetRayIntersection(ray, 20, mask);
                if (hit.collider != null)
                {
                    hitEvent?.Invoke(hit);
                    HitObject = hit.collider.gameObject.name;
                }
            }
            void HandleCommunicationData()
            {
                if (User.IsConnectionCreated)
                {
                    if (_inputService.IsMouseLeftButton())
                    {
                        Communicator.SendData.IsLeftButton = true;
                    }
                    else
                    {
                        Communicator.SendData.IsLeftButton = false;
                    }
                }
            }
        }
    }
}
```

Класс InvenotryInputService.cs:

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

```
using Zenject;

namespace Assets.Code.Scripts.Gameplay
{
   public class InventoryInputService : MonoBehaviour
   {
      InputService _inputService;
      public event Action<int> OnChooseCellNumberEvent;
      public event Action<Item> OnChooseCellTypeEvent;
      [Inject]
      public void Constructor(InputService inputService)
      {
         _inputService = inputService;
      }

      public void Update()
      {
         if (_inputService.IsCell_1())
         {
            OnChooseCellTypeEvent?.Invoke(Item.Basket);
            OnChooseCellNumberEvent?.Invoke(1);
         }
         if (_inputService.IsCell_2())
         {
            OnChooseCellTypeEvent?.Invoke(Item.Watering);
            OnChooseCellNumberEvent?.Invoke(2);
         }
         if (_inputService.IsCell_3())
         {
            OnChooseCellTypeEvent?.Invoke(Item.Wheat);
            OnChooseCellNumberEvent?.Invoke(3);
         }
         if (_inputService.IsCell_4())
         {
            OnChooseCellTypeEvent?.Invoke(Item.SunFlower);
            OnChooseCellNumberEvent?.Invoke(4);
         }
         if (_inputService.IsCell_5())
         {
            OnChooseCellTypeEvent?.Invoke(Item.Pumpking);
            OnChooseCellNumberEvent?.Invoke(5);
         }
         if (_inputService.IsCell_6())
         {
            OnChooseCellTypeEvent?.Invoke(Item.Reed);
            OnChooseCellNumberEvent?.Invoke(6);
         }
         if (_inputService.IsCell_7())
         {
            OnChooseCellTypeEvent?.Invoke(Item.InfernalGrowth);
            OnChooseCellNumberEvent?.Invoke(7);
         }
      }
   }
}
```

Класс Inventory.cs:

```
using System;
using System.Collections.Generic;
using Random = UnityEngine.Random;

namespace Assets.Code.Scripts.Gameplay
{
   public class Inventory
   {
      Dictionary<Item, int> _itemsDictionary;

      public Inventory()
      {
         _itemsDictionary = new Dictionary<Item, int>();
         InitItemsDictionaryRandom();
      }

      public int this[Item item]
      {
         get
         {
            {
               return _itemsDictionary[item];
            }
         }
         set
         {
            _itemsDictionary[item] = value;
         }
      }

      void InitItemsDictionaryRandom()
      {
         Item[] items = (Item[])Enum.GetValues(typeof(Item));

         for (int i = 0; i < items.Length; i++)
         {
            if (items[i] != Item.Basket)
            {
               _itemsDictionary[items[i]] = Random.Range(2, 5);
            }
         }
      }
   }
}
```

Класс Cell.cs:

```
using UnityEngine;
using UnityEngine.UI;
using Zenject;
using TMPro;
```

```
namespace Assets.Code.Scripts.Gameplay
{
    [RequireComponent(typeof(Image))]
    public class Cell : MonoBehaviour
    {
        public Item Item;
        Image _image;
        Sprite _defaultSprite;
        Sprite _choosenSprite;
        InventoryInputService _inventoryInputService;
        public TextMeshProUGUI Text;
        Inventory _inventory;
        [Inject]
        public void Constructor(InventoryInputService inventoryInputService, Inventory inventory)
        {
            _inventoryInputService = inventoryInputService;
            _inventory = inventory;
        }
        private void OnEnable()
        {
            _inventoryInputService.OnChooseCellTypeEvent += OnChooseCell;
        }
        private void OnDisable()
        {
            _inventoryInputService.OnChooseCellTypeEvent -= OnChooseCell;
        }
        private void Start()
        {
            _image = GetComponent<Image>();
            _defaultSprite = _image.sprite;
        }
        private void Update()
        {
            if(Item != Item.Basket)
            {
                Text.text = _inventory[Item].ToString();
            }
        }
        public void SetChoosenSprite(Sprite sprite)
        {
            _choosenSprite = sprite;
        }


        void OnChooseCell(Item cellType)
        {
            if (Item == cellType)
                _image.sprite = _choosenSprite;
            else
                _image.sprite = _defaultSprite;
        }
    }
}
```

Класс UseItems.cs:

```
using Assets.Code.Scripts.Gameplay.PlantingTerritory;
using System.Collections.Generic;
using UnityEngine;
using Random = UnityEngine.Random;
using Zenject;
using Assets.Code.Scripts.Boot;
using Assets.Code.Scripts.Boot.Communication;
using Assets.Code.Scripts.Boot.Data;

namespace Assets.Code.Scripts.Gameplay
{
   [RequireComponent(typeof(CursorRay))]
   public class UseItems : MonoBehaviour
   {
      [SerializeField] float Distance = 30f;

      CursorRay _cursorRay;
      Item _cuurentChoosenItem;
      InventoryInputService _inventoryInputService;
      SeedsService _seedsService;
      Inventory _inventory;
      List<ItemCommand> _sendedCommands;
      [Inject]
      public void Constructor(InventoryInputService inventoryInputService,
         SeedsService seedsService, Inventory inventory)
      {
         _inventoryInputService = inventoryInputService;
         _seedsService = seedsService;
         _inventory = inventory;
      }

      private void OnDisable()
      {
         _cursorRay.OnHitTerritoryEvent -= OnHitTerritory2DCollider;
         _cursorRay.OnHitSeedEvent -= OnHitSeed2DCollider;
         _inventoryInputService.OnChooseCellTypeEvent -= OnChooseCell;
      }
      private void Start()
      {
         _sendedCommands = new List<ItemCommand>();
         _cursorRay = GetComponent<CursorRay>();
         _cursorRay.OnHitTerritoryEvent += OnHitTerritory2DCollider;
         _cursorRay.OnHitSeedEvent += OnHitSeed2DCollider;
         _inventoryInputService.OnChooseCellTypeEvent += OnChooseCell;
      }

      private void Update()
      {
      }
      public void OnHitTerritory2DCollider(RaycastHit2D hit)
      {
         if (hit.distance < Distance)
         {
```

```
        GameObject go = hit.collider.gameObject;
        UsingCrops(go);
    }
}
public void OnHitSeed2DCollider(RaycastHit2D hit)
{
    if (hit.distance < Distance)
    {
        GameObject go = hit.collider.gameObject;
        UsingTools(go);
    }
}

void UsingCrops(GameObject go)
{
    if (go.TryGetComponent(out PlantTerritory territory))
    {
        if (territory.IsTerritoryContain(_cuurentChoosenItem) == false ||
            territory.IsEmpty == false ||
            _inventory[_cuurentChoosenItem] < 1)
        {
            return;
        }

        Seed seedPrefab = _seedsService.GetSeedFor(_cuurentChoosenItem);
        SeedSO seedSO = _seedsService.GetSeedSOFor(_cuurentChoosenItem);

        seedPrefab = Instantiate(seedPrefab, territory.transform);

        seedPrefab.Initialize(seedSO, territory, User.Instance.PlayerType);
        territory.SetSeed(seedPrefab);

        territory.SetEmpty(false);

        _inventory[_cuurentChoosenItem]--;

        if (User.IsConnectionCreated)
        {
            Communicator.SendData.AddNotFreeTerritory(territory.name);
            ItemCommand itemCommand = new ItemCommand();
            itemCommand.ObjectType = _cuurentChoosenItem;
            itemCommand.ParentTerritoryName = territory.name;
            itemCommand.CommandType = CommandType.Spawn;
            itemCommand.PlayerType = User.Instance.PlayerType;
            _sendedCommands.Add(itemCommand);
            Communicator.SendData.AddItemCommand(itemCommand);
        }
    }
}

void UsingTools(GameObject go)
{
    if (go.TryGetComponent(out Seed seed))
    {
        switch (_cuurentChoosenItem)
        {
```

```
                case Item.Basket:
                    {
                        if (seed.Status == GrowStatus.Ready
                            && seed.ParentPlayer == User.Instance.PlayerType)
                        {
                            GameEvents.InvokePickSeedEvent(seed.Money);
                            _inventory[Item.Watering] += Random.Range(0, 2);
                            _inventory[seed.GetSeedType()] += Random.Range(1, 3);
                            seed.OnPick();
                            Destroy(seed.gameObject);

                            if (User.IsConnectionCreated)
                            {
                                ItemCommand itemCommand = new ItemCommand();
                                itemCommand.ParentTerritoryName = seed.GetParentTerritoryName();
                                itemCommand.CommandType = CommandType.Delete;
                                itemCommand.ObjectType = _cuurentChoosenItem;
                                Communicator.SendData.AddItemCommand(itemCommand);
                                Communicator.SendData.NotFreeTerritoryList
                                    .Remove(seed.GetParentTerritoryName());
                            }
                        }

                        break;
                    }
                case Item.Watering:
                    {
                        if (_inventory[_cuurentChoosenItem] < 1
                            && seed.ParentPlayer == User.Instance.PlayerType)
                            break;

                        if (seed.Status == GrowStatus.Growing)
                        {
                            seed.Boost();
                            _inventory[_cuurentChoosenItem]--;
                        }

                        break;
                    }
            }
        }
    }
    void OnChooseCell(Item item)
    {
        _cuurentChoosenItem = item;
    }
  }
}
```

Класс ItemCommandsHandler.cs:

```
using Assets.Code.Scripts.Boot;
using Assets.Code.Scripts.Boot.Communication;
using Assets.Code.Scripts.Boot.Data;
using Assets.Code.Scripts.Gameplay.PlantingTerritory;
```

```
using System.Collections.Generic;
using System.Linq;
using UnityEngine;
using Zenject;

namespace Assets.Code.Scripts.Gameplay
{
   public class ItemCommandsHandler : MonoBehaviour
   {
      SeedsService _seedsService;
      TerritoryService _territoryService;
      [Inject]
      public void Constructor(SeedsService seedsService, TerritoryService territoryService)
      {
         _seedsService = seedsService;
         _territoryService = territoryService;
      }

      private void Update()
      {
         if (User.IsConnectionCreated)
         {
            HandleCommands();
            DeleteComplitedCommands();
         }
      }

      void HandleCommands()
      {
         if (Communicator.RecvData.ItemCommands.Count > 0)
         {
            List<ItemCommand> commands = Communicator.RecvData.ItemCommands;
            List<string> NotFreeTerr = Communicator.RecvData.NotFreeTerritoryList;
            for (int i = 0; i < commands.Count; i++)
            {
               switch(commands[i].CommandType)
               {
                  case CommandType.Spawn:
                     {
                        Seed seedPrefab = _seedsService.GetSeedFor(commands[i].ObjectType);
                        SeedSO seedSO = _seedsService.GetSeedSOFor(commands[i].ObjectType);

                        PlantTerritory terr =
                           _territoryService.GetTerritiryByObjectName(commands[i].ParentTerritoryName);

                        if (terr.IsEmpty == true)
                        {
                           seedPrefab = Instantiate(seedPrefab,
                              terr.transform);

                           seedPrefab.Initialize(seedSO, terr, commands[i].PlayerType);
                           terr.SetEmpty(false);
                           terr.SetSeed(seedPrefab);
                           Communicator.SendData.AddComplitedCommand(commands[i]);
                           Communicator.SendData.AddNotFreeTerritory(commands[i].ParentTerritoryName);
```

```
                    }
                    break;
                }
            case CommandType.Delete:
                {
                    PlantTerritory terr =
                        _territoryService.GetTerritiryByObjectName(commands[i].ParentTerritoryName);
                    terr.DestroySeed();
                    terr.SetEmpty(true);
                    Communicator.SendData.AddComplitedCommand(commands[i]);
                    break;
                }
            }
            commands.RemoveAt(i);

        }
      }
    }

    public void DeleteComplitedCommands()
    {
      if (Communicator.SendData.ItemCommands.Count > 0)
      {

        List<ItemCommand> sendCommands = Communicator.SendData.ItemCommands;

        List<ItemCommand> recvComplitedCommands = Communicator.RecvData.CompletedCommands;

        List<ItemCommand> commonCommands = sendCommands.Intersect(recvComplitedCommands).ToList();

        sendCommands.RemoveAll(item => commonCommands.Contains(item));
        recvComplitedCommands.RemoveAll(item => commonCommands.Contains(item));
      }
    }
  }
}
```

Класс Movement.cs:

```
using UnityEngine;

namespace Assets.Code.Scripts.Gameplay.Player.PlayerControl
{
  [RequireComponent(typeof(Animator))]
  [RequireComponent(typeof(Rigidbody2D))]
  public abstract class Movement : MonoBehaviour
  {
    protected Rigidbody2D RigidBody;
    protected float MovementSpeed = 3f;
    protected float _smoothTime = 2f;
    Vector2 _velocity;
    protected Vector2 CurrentPosition;
    protected Vector2 NewPosition;
    protected Animator Animator;
    protected int XKey;
```

```
        protected int YKey;

        private void Start()
        {
            RigidBody = GetComponent<Rigidbody2D>();
            Animator = GetComponent<Animator>();
            XKey = Animator.StringToHash("X");
            YKey = Animator.StringToHash("Y");
            OnStart();
        }

        protected void InputAndAnimateInFouthDirections(ref Vector2 input)
        {
            if (input.x > 0)
            {
                Animator.SetFloat(XKey, 1);
                Animator.SetFloat(YKey, 0);
                input.y = 0;
            }
            else if (input.x < 0)
            {
                Animator.SetFloat(XKey, -1);
                Animator.SetFloat(YKey, 0);
                input.y = 0;
            }
            else if (input.y > 0)
            {
                Animator.SetFloat(YKey, 1);
                Animator.SetFloat(XKey, 0);
                input.x = 0;
            }
            else
            {
                Animator.SetFloat(YKey, -1);
                Animator.SetFloat(XKey, 0);
                input.x = 0;
            }

            if (input.x == 0 && input.y == 0)
            {
                Animator.SetFloat(YKey, 0);
                Animator.SetFloat(XKey, 0);
            }
        }

        protected Vector2 Move(Vector2 currPos, Vector2 newPos)
        {
            newPos = Vector2.SmoothDamp(currPos, newPos, ref _velocity,
                Time.fixedDeltaTime, MovementSpeed);
            RigidBody.MovePosition(newPos);
            return RigidBody.position;
        }
        public abstract void OnStart();
    }
}
```

Класс ConnectedPlayerMovement.cs:

```
using Assets.Code.Scripts.Boot;
using Assets.Code.Scripts.Boot.Communication;
using UnityEngine;

namespace Assets.Code.Scripts.Gameplay.Player.PlayerControl
{
    public class ConnectedPlayerMovement : Movement
    {
        public override void OnStart()
        {
        }

        private void Update()
        {
            if(User.IsConnectionCreated)
            {
                Vector2 input = Communicator.RecvData.GetDirection();

                InputAndAnimateInFouthDirections(ref input);
            }
        }

        private void FixedUpdate()
        {
            if (User.IsConnectionCreated)
            {
                NewPosition = Communicator.RecvData.GetPosition();
                CurrentPosition = RigidBody.position;
                CurrentPosition = Move(CurrentPosition, NewPosition);

            }
        }
    }
}
```

Класс PlayerMovement.cs:

```
using Zenject;
using UnityEngine;
using Assets.Code.Scripts.Boot.Communication;
using Assets.Code.Scripts.Boot;

namespace Assets.Code.Scripts.Gameplay.Player.PlayerControl
{
    public class PlayerMovement : Movement
    {
        InputService _inputService;
        Vector2 _input;
        [Inject]
        public void Constructor(InputService inputService)
        {
            _inputService = inputService;
```

```
        }
        public override void OnStart()
        {
        }

        protected virtual void Update()
        {
            _input = _inputService.GetMovement();
            if (User.IsConnectionCreated)
            {
                Communicator.SendData.SetDirection(_input);

            }
            InputAndAnimateInFouthDirections(ref _input);

        }

        private void FixedUpdate()
        {
            NewPosition = RigidBody.position + _input;
            if(User.IsConnectionCreated)
                Communicator.SendData?.UpdatePosition(NewPosition);
            CurrentPosition = RigidBody.position;
            CurrentPosition = Move(CurrentPosition, NewPosition);
        }
    }
}
```

Класс TerritoryService.cs:

```
using System.Collections.Generic;
using System.Linq;


namespace Assets.Code.Scripts.Gameplay.PlantingTerritory
{
    public class TerritoryService
    {
        List<PlantTerritory> _plantTerritories;

        public TerritoryService()
        {
            _plantTerritories = new List<PlantTerritory>();
        }

        public void AddTerritory(PlantTerritory plantTerritory)
        {
            _plantTerritories.Add(plantTerritory);
        }

        public PlantTerritory GetTerritiryByObjectName(string name)
        {
            return _plantTerritories.FirstOrDefault(terr => terr.gameObject.name == name);
        }
    }
```

```
}
```

Класс PlantTerritory.cs:

```
using System.Collections.Generic;

using UnityEngine;
using Zenject;
using Random = UnityEngine.Random;

namespace Assets.Code.Scripts.Gameplay.PlantingTerritory
{
   [RequireComponent(typeof(SpriteRenderer))]
   public class PlantTerritory : MonoBehaviour
   {
      public bool IsEmpty { get; protected set; }
      [SerializeField] List<Item> SeedsToGrow;
      protected SpriteRenderer Sprite;
      Color _defaultColor;
      TerritoryService _territoryService;
      Seed _seed;
      [Inject]
      public void Constructor(TerritoryService territoryService)
      {
         _territoryService = territoryService;
      }
      private void Start()
      {
         IsEmpty = true;
         Sprite = GetComponent<SpriteRenderer>();
         _defaultColor = Sprite.color;
         _territoryService.AddTerritory(this);
         RandomlyFlip();
      }
      void RandomlyFlip()
      {
         // Генерируем случайное булево значение для flipX
         bool randomFlipX = Random.value > 0.5f;

         // Генерируем случайное булево значение для flipY
         bool randomFlipY = Random.value > 0.5f;

         // Устанавливаем свойства flipX и flipY
         Sprite.flipX = randomFlipX;
         Sprite.flipY = randomFlipY;
      }
      void OnMouseEnter()
      {
         Sprite.color = new Color(Sprite.color.r, Sprite.color.g,
            Sprite.color.b, Sprite.color.a / 2);
      }
      void OnMouseExit()
      {
         Sprite.color = _defaultColor;
      }
```

```
        public bool IsTerritoryContain(Item seed)
        {
            return SeedsToGrow.Contains(seed);
        }
        public void SetSeed(Seed seed)
        {
            _seed = seed;
        }
        public void DestroySeed()
        {
            if(_seed != null)
            {
                Destroy(_seed.gameObject);
            }
        }
        public void SetEmpty(bool isEmpty)
        {
            IsEmpty = isEmpty;
        }

    }
}
```

Класс SpawnInstaller.cs:

```
using Zenject;
using UnityEngine;
using Assets.Code.Scripts.Boot;

namespace Assets.Code.Scripts.Gameplay.Installers
{
    public class SpawnInstaller : MonoInstaller
    {
        [SerializeField] GameObject PlayerPrefab;
        [SerializeField] Transform HostSpawnPoint;
        [SerializeField] GameObject ConnectedPlayerPrefab;
        [SerializeField] Transform ConnectedPlayerSpawnPoint;

        public override void InstallBindings()
        {
            Container.BindInstance(ConnectedPlayerSpawnPoint).WithId("ConnectedSpawn");
            Container.BindInstance(HostSpawnPoint).WithId("HostSpawn");

            switch (User.Instance.ConnectionType)
            {
                case ConnectionType.Server:
                    {
                        Container.InstantiatePrefab(PlayerPrefab,
                            HostSpawnPoint.position,
                            Quaternion.identity, null);
                        Container.InstantiatePrefab(ConnectedPlayerPrefab,
                            ConnectedPlayerSpawnPoint.position,
                            Quaternion.identity, null);
                        break;
                    }
```

```
            case ConnectionType.Client:
                {
                    Container.InstantiatePrefab(PlayerPrefab,
                        ConnectedPlayerSpawnPoint.position,
                        Quaternion.identity, null);
                    Container.InstantiatePrefab(ConnectedPlayerPrefab,
                      HostSpawnPoint.position,
                      Quaternion.identity, null);
                    break;
                }
        }
    }
  }
}
```

Класс SeedsInstaller.cs:

```
using System.Collections.Generic;
using Zenject;

namespace Assets.Code.Scripts.Gameplay.Installers
{
  public class SeedsInstaller : MonoInstaller
  {
    public List<SeedSO> SeedsSO;
    public List<Seed> SeedsPrefabs;
    Dictionary<Item, Seed> _seedsGO;
    Dictionary<Item, SeedSO> _seedsSO;
    public override void InstallBindings()
    {
      InitSeedsGO();
      InitSeedsSO();

      SeedsService seedsService = new SeedsService(_seedsGO, _seedsSO);

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

    void InitSeedsGO()
    {
      _seedsGO = new Dictionary<Item, Seed>();

      foreach (var seed in SeedsPrefabs)
      {
        _seedsGO[seed.GetSeedType()] = seed;
      }
    }
    void InitSeedsSO()
    {
      _seedsSO = new Dictionary<Item, SeedSO>();

      foreach (var seedSO in SeedsSO)
      {
        _seedsSO[seedSO.SeedType] = seedSO;
      }
```

```
        }
    }
}
```

Класс PlayersInstaller.cs:

```
using Assets.Code.Scripts.Boot;
using Assets.Code.Scripts.Gameplay.Player.PlayerControl;
using UnityEngine;
using Zenject;
using Cinemachine;

namespace Assets.Code.Scripts.Gameplay.Installers
{
    public class PlayersInstaller : MonoInstaller
    {
        [SerializeField] GameObject PlayerPrefab;
        [SerializeField] Transform HostSpawnPoint;
        [SerializeField] GameObject ConnectedPlayerPrefab;
        [SerializeField] Transform ConnectedPlayerSpawnPoint;

        public PlayerMode Mode;
        [SerializeField] CinemachineVirtualCamera Camera;
        GameObject _playerGameoObject;
        public override void InstallBindings()
        {
            BindSpawnPoints();
            BindMovement();

            switch (Mode)
            {
                case PlayerMode.Single:
                    {
                        SingleMode();
                        break;
                    }
                case PlayerMode.Multiple:
                    {

                        SpawnPlayersOnSpawnPoints();
                        break;
                    }
            }
        }
        void SpawnPlayersOnSpawnPoints()
        {
            switch (User.Instance.ConnectionType)
            {
                case ConnectionType.Server:
                    {
                        _playerGameoObject = Container.InstantiatePrefab(PlayerPrefab,
                            HostSpawnPoint.position,
                            Quaternion.identity, null);
                        Container.InstantiatePrefab(ConnectedPlayerPrefab,
                            ConnectedPlayerSpawnPoint.position,
```

```
                    Quaternion.identity, null);
                break;
            }
        case ConnectionType.Client:
            {
                _playerGameoObject = Container.InstantiatePrefab(PlayerPrefab,
                    ConnectedPlayerSpawnPoint.position,
                    Quaternion.identity, null);
                Container.InstantiatePrefab(ConnectedPlayerPrefab,
                  HostSpawnPoint.position,
                  Quaternion.identity, null);
                break;
            }
      }
      if (_playerGameoObject != null)
      {
        Camera.Follow = _playerGameoObject.transform;
        Camera.LookAt = _playerGameoObject.transform;
      }

    }
    void SingleMode()
    {
      _playerGameoObject = Container.InstantiatePrefab(PlayerPrefab, HostSpawnPoint.position,
Quaternion.identity, null);
      Camera.Follow = _playerGameoObject.transform;
      Camera.LookAt = _playerGameoObject.transform;
    }
    void BindSpawnPoints()
    {
      Container.BindInstance(ConnectedPlayerSpawnPoint).WithId("ConnectedSpawn");
      Container.BindInstance(HostSpawnPoint).WithId("HostSpawn");
    }
    void BindMovement()
    {
      Container.Bind<PlayerMovement>().AsSingle();
    }
  }
}
```

Класс InventoryInstaller.cs:

```
using System.Collections.Generic;
using Zenject;
using UnityEngine;


namespace Assets.Code.Scripts.Gameplay.Installers
{
  public class InventoryInstaller : MonoInstaller
  {
    [SerializeField] InventoryInputService InventoryInputService;
    [Header("UI Cells:")]
    [SerializeField] List<Cell> Cells;
    [Header("UI Choosen Cells sprites:")]
```

```
        [SerializeField] List<Sprite> ChoosenCells;
        public override void InstallBindings()
        {
            BindInventoryInputService();
            BindCells();
            BindInventory();

            for (int i = 0; i < Cells.Count; i++)
            {
                Cells[i].SetChoosenSprite(ChoosenCells[i]);
            }
        }

        void BindInventoryInputService()
        {
            Container.BindInstance(InventoryInputService).AsSingle();
        }
        void BindCells()
        {
            Container.Bind<Cell>().AsTransient();
        }
        void BindInventory()
        {
            Inventory inventory = new Inventory();
            Container.BindInstance(inventory).AsSingle();
        }
    }
}
```

Класс GameInstaller.cs:

```
using Assets.Code.Scripts.Gameplay.PlantingTerritory;
using Zenject;

namespace Assets.Code.Scripts.Gameplay.Installers
{
    public class GameInstaller : MonoInstaller
    {
        public override void InstallBindings()
        {
            BindInputSystem();
            BindTerritoryService();
            Container.Bind<CursorRay>().AsSingle();
            Container.Bind<UseItems>().AsSingle();
            Container.Bind<ItemCommandsHandler>().AsSingle();
            Container.Bind<PlantTerritory>().AsTransient();
        }

        void BindInputSystem()
        {
            InputService inputService = new InputService();
            Container.BindInstance(inputService).AsSingle();
        }

        void BindTerritoryService()
```

```
        {
            TerritoryService territoryService = new TerritoryService();
            Container.BindInstance(territoryService).AsSingle();
        }
    }
}
```

Класс GaneOverUI.cs:

```
using UnityEngine;
using TMPro;
using UnityEngine.SceneManagement;

namespace Assets.Code.Scripts.Gameplay
{
    public class GameOverUI : MonoBehaviour
    {
        const string c_menuSceneName = "MenuScene";
        [SerializeField] GameObject GameOverPanel;
        [SerializeField] TextMeshProUGUI LoseText;
        [SerializeField] TextMeshProUGUI WinText;
        [SerializeField] MoneyDisplayer PlayerCoin;
        [SerializeField] MoneyDisplayer ConnectedPlayerCoin;
        private void Start()
        {
            GameEvents.OnGameOverEvent += OnGameOver;
            GameOverPanel.SetActive(false);
            LoseText.gameObject.SetActive(false);
            WinText.gameObject.SetActive(false);
        }
        private void OnDisable()
        {
            GameEvents.OnGameOverEvent -= OnGameOver;
        }

        void OnGameOver()
        {
            GameOverPanel.SetActive(true);
            if (PlayerCoin.GetMoney() > ConnectedPlayerCoin.GetMoney())
            {
                WinText.gameObject.SetActive(true);
            }
            else
                LoseText.gameObject.SetActive(true);
        }

        public void OnExit()
        {
            SceneManager.LoadScene(c_menuSceneName);
        }
    }
}
```

Класс Lamp.cs:

```
using UnityEngine;

namespace Assets.Code.Scripts.Gameplay.Days
{
   [RequireComponent(typeof(Light))]
   public class Lamp : MonoBehaviour
   {
      Light _light;
      private void Start()
      {
         _light = GetComponent<Light>();
         GameEvents.OnDayStartEvent += OnDay;
         GameEvents.OnNightStartEvent += OnNight;
         _light.intensity = -1;
      }

      private void OnDisable()
      {
         GameEvents.OnDayStartEvent -= OnDay;
         GameEvents.OnNightStartEvent -= OnNight;
      }

      void OnDay()
      {
         _light.intensity = -1;
      }
      void OnNight()
      {
         _light.intensity = 1;
      }
   }
}
```

Класс DayNight.cs:

```
using UnityEngine;
using TMPro;

namespace Assets.Code.Scripts.Gameplay
{
   public class DayNight : MonoBehaviour
   {
      public TextMeshProUGUI DaysText;
      public float DayIntensivity;
      public float NightIntensivity;
      [SerializeField] Light Lighting;
      public int DaysToEnd;
      public float DayTime;
      public float TestDayTime;
      public static int CurrentDays;
      public float _currentTime;
      public float AddIntensivity;
```

```
        [SerializeField] bool IsTest;
        bool _flag;
        bool _isGameOver;
        private void Start()
        {
            _flag = true;
            _currentTime = 0f;
            _isGameOver = false;
            if(IsTest)
            {
                DayTime = TestDayTime;
            }
        }

        private void Update()
        {
            if(CurrentDays >= DaysToEnd && _isGameOver == false)
            {
                _isGameOver = true;
                CurrentDays = 0;
                GameEvents.InvokeGameOverEvent();
                return;
            }
            _currentTime += Time.deltaTime;
            if (_currentTime >= DayTime)
            {
                if(_flag == false)
                {
                    GameEvents.InvokeOnDayStartEvent(DayIntensivity);
                    CurrentDays++;
                }
                else
                {
                    GameEvents.InvokeOnNightStartEvent(NightIntensivity);
                }

                _flag = !_flag;
                _currentTime = 0;
            }

            if (_flag)
            {
                Lighting.intensity = Mathf.Lerp(Lighting.intensity, DayIntensivity, AddIntensivity * Time.deltaTime);
            }
            else
            {
                Lighting.intensity = Mathf.Lerp(Lighting.intensity, NightIntensivity, AddIntensivity * Time.deltaTime);
            }

            DaysText.text = CurrentDays.ToString();
        }
    }
}
```

Класс MoneyDisplayer.cs:

```
using TMPro;
using UnityEngine;

namespace Assets.Code.Scripts.Gameplay
{
   public abstract class MoneyDisplayer : MonoBehaviour
   {
      public TextMeshProUGUI CoinText;
      protected float CurrentMoney;

      private void Start()
      {
         CurrentMoney = 0;
         OnStart();
      }

      private void OnDisable()
      {
         Disable();
      }
      protected abstract void OnStart();
      protected abstract void Disable();


      public float GetMoney()
      {
         return CurrentMoney;
      }
   }
}
```

Класс Coin.cs:

```
using Assets.Code.Scripts.Boot;
using Assets.Code.Scripts.Boot.Communication;

namespace Assets.Code.Scripts.Gameplay
{
   public class Coin : MoneyDisplayer
   {
      private void Update()
      {
         CoinText.text = CurrentMoney.ToString();
      }
      public void OnPickSeed(float money)
      {
         CurrentMoney += money;
         if(User.IsConnectionCreated)
         {
            Communicator.SendData.Money = CurrentMoney;
         }
      }
```

```
        protected override void OnStart()
        {
            GameEvents.OnPickSeedEvent += OnPickSeed;
        }

        protected override void Disable()
        {
            GameEvents.OnPickSeedEvent -= OnPickSeed;
        }
    }
}
```

Класс ConnectedCoint.cs:

```
using Assets.Code.Scripts.Boot;
using Assets.Code.Scripts.Boot.Communication;

namespace Assets.Code.Scripts.Gameplay.Coins
{
    public class ConnectedCoin : MoneyDisplayer
    {
        private void Update()
        {
            if(User.IsConnectionCreated)
            {
                CoinText.text = Communicator.RecvData.Money.ToString();
            }
        }
        protected override void Disable()
        {
        }

        protected override void OnStart()
        {
        }
    }
}
```

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 c* 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 аркадной платформой для многопользовательской 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;

– компьютерные средства: клавиатура, мышь;

– устройство вывода: компьютерный 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 длиться до тех пор, пока не пройдет определенное количество дней. После этого игра закончится и появится экран окончания, на котором следует нажать соответствующую клавишу.

3. Обращение к программе.

Для старта работы приложения необходимо запустить файл с расширением *FarmVille.exe*.

4. Входные и выходные данные.

В качестве входных данных 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 «Ферма» является аркадной платформой для многопользовательской игры на одном устройстве. Основное применение – получение досуга. Доступен лишь один режим – игра по сети.

Приложение имеет три этапа работы: главное меню, непосредственный игровой процесс и меню окончания игры.

2. Назначение и условия применения.

Игровое приложение «Ферма» является аркадной платформой для многопользовательской игры по сети.

Программа показала свою максимальную производительность при взаимодействии с:

– операционной системой *Windows* 7;

– компьютерные средства: клавиатура, мышь;

– устройство вывода: компьютерный монитор.

3. Подготовка к работе.

Для старта работы приложения необходимо запустить файл с расширением *FarmVille.exe*.

4. Описание операций.

Основные манипуляции со стороны игрока:

– старт игры;

– перемещение игрового персонажа по игровому полю влево и вправо;

– перемещение игрового персонажа по игровому полю вверх;

– 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 столько замечательной и увлекательной игрой.

ПРИЛОЖЕНИЕ Д
(обязательное)
Внешний вид окон интерфейса приложения

После запуска приложения открывается одно окно. После загрузки приложения перед пользователем открывается главное меню. На рисунке Д.1 представлен вид главного меню.

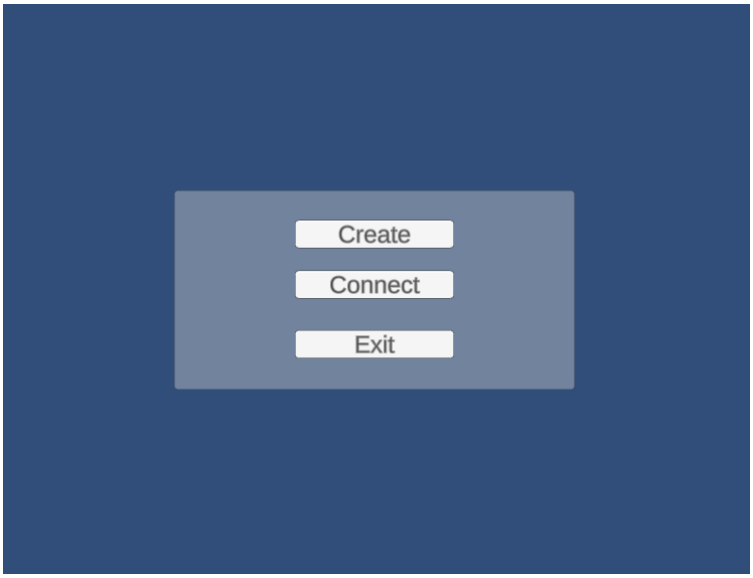

Рисунок Д.1 – Вид главного меню

При успешном создании подключения пользователю открывается игровая сцена. На рисунке Д.2 представлен вид игры после загрузки уровня от лица двух игроков.

Рисунок Д.2 – Вид игры после загрузки уровня от лица двух игроков

ПРИЛОЖЕНИЕ Ё
(обязательное)
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;

– широкополосное интернет-соединение для загрузки контента и сетевого взаимодействия.

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 уровня двумя пользователями;

– синхронизация игрового времени;

– корректная передача и отображение данных;

– закрытие приложения.

Необходимости в доработке или устранении проблем не выявлено, так как в процессе эксплуатации не было выявлено никаких ошибок или недочетов.

ПРИЛОЖЕНИЕ Ж

(обязательное)

Схема структуры приложения