**1 АНАЛИТИЧЕСКИЙ ОБЗОР СФЕРЫ РАЗРАБОТКИ ИГРОВЫХ ПРИЛОЖЕНИЙ**

**1.1 Описание предметной области**

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

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

В структуре современной игровой индустрии можно выделить следующие уровни:

– платформы;

– игровые движки;

– разработка видеоигр;

– издание и оперирование;

– популяризация и потребление.

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

– персональные компьютеры на базе *Windows*, *Mac/OS X* или *Linux*;

– игровые консоли (специализированные устройства для игр, *Xbox One*, *PlayStation 4*, *Nintendo Wii U*;

– мобильные устройства (*iOS*, *Android*, *Windows*).

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

Разработка игр. Большое количество компаний и независимых команд занимаются созданием компьютерных игр. В разработке участвует специалисты разных профессий: программисты, гейм-дизайнеры, художники, *QA* специалисты и др. К разработке крупных коммерческих игровых продуктов привлекаются большие профессиональные команды. Стоимость разработки подобных проектов может составлять десятки миллионов долларов. Однако вполне успешные игровые проекты могут воплощаться и небольшими командами энтузиастов. Этому способствует присутствие на рынке большого количества открытых и распространенных платформ, качественных и практически бесплатных движков, площадок по привлечению «народных» инвестиций (краудфандинг) и доступных каналов распространения.

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

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

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

**1.2 Жанр аркадного фермерского симулятора**

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

Разработчики фермерских игр должны учитывать разнообразные аспекты, включая управление фермой, выращивание культур, разведение скота, улучшение оборудования и многое другое. Важно создать баланс между реализмом и увлекательным игровым процессом, чтобы игроки могли насладиться аутентичным опытом фермерской деятельности.

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

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

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

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

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

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

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

**1.3 Аналитический обзор представителей игр жанра аркадного фермерского симулятора**

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

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

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

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

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

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

Лесное хозяйство. В версии *Farming Simulator 15* была введена возможность заниматься лесоводством, что добавляет ещё один интересный элемент в игровой процесс. Этот аспект игры также требует специфических знаний и навыков для успешного ведения бизнеса.

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

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

На рисунке 1.1 представлен кадр из *Farming Simulator*.



Рисунок 1.1 Кадр из *Farming Simulator*

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

Сюжет и персонажи. В основе сюжета лежит история персонажа, который получает в наследство ферму от своего дедушки. Игрок может настроить внешность и пол персонажа, однако эти параметры не влияют на игровой процесс. Главный герой свободен в своих действиях: можно заниматься фермерством, исследовать пещеры, общаться с местными жителями или выполнять квесты.

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

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



Рисунок 1.2 – Кадр из игры *Stardew Valley*

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

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

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

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

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

**1.4 Игровой движок *Unity***

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

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

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

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

На рисунке 1.3 представлен интерфейс игрового движка Unity.



Рисунок 1.3 – интерфейс движка *Unity*

Дополнением к производительности, которую обеспечивает редактор, служит сильная межплатформенная поддержка набора инструментов *Unity*. В данном случае это словосочетание подразумевает не только места развертывания (игру можно развернуть на персональном компьютере, в интернете, на мобильном устройстве или на консоли), но и инструменты разработки (игры создаются на машинах, работающих под управлением как *Windows*, так и *Mac OS*). Эта независимость от платформы явилась результатом того, что изначально приложение *Unity* предназначалось исключительно для компьютеров *Mac*, а позднее было перенесено на машины с операционными системами семейства *Windows*. Первая версия появилась в 2005 году, а к настоящему моменту вышли уже пять основных версий (с множеством небольших, но частых обновлений). Изначально разработка и развертка поддерживались только для машин *Mac*, но через несколько месяцев вышло обновление, позволяющее работать и на машинах с Windows. В следующих версиях добавлялись все новые платформы развертывания, например межплатформенный веб-плеер в 2006-м, *iPhone* в 2008- м, *Android* в 2010-м и даже такие игровые консоли, как *Xbox* и *PlayStation*. Позднее появилась возможность развертки в *WebGL* – новом фреймворке для трехмерной графики в веб-браузерах. Немногие игровые движки поддерживают такое количество целевых платформ развертывания, и ни в одном из них развертка на разных платформах не осуществляется настолько просто. Дополнением к этим основным достоинствам идет и третье, менее бросающееся в глаза преимущество в виде модульной системы компонентов, которая используется для конструирования игровых объектов. «Компоненты» в такой системе представляют собой комбинируемые пакеты функциональных элементов, поэтому объекты создаются как наборы компонентов, а не как жесткая иерархия классов. В результате получается альтернативный (и обычно более гибкий) подход к объектно-ориентированному программированию, в котором игровые объекты создаются путем объединения, а не наследования.

Оба подхода схематично показаны на рисунке 2.1.

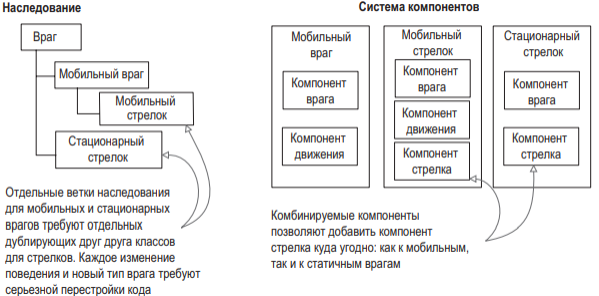


Рисунок 1.4 – Сравнение наследования с компонентной системой

Каждое изменение поведения и новый тип врага требуют серьезной перестройки кода. Комбинируемые компоненты позволяют добавить компонент стрелка куда угодно: как к мобильным, так и к статичным врагам. В компонентной системе объект существует в горизонтальной иерархии, поэтому объекты состоят из наборов компонентов, а не из иерархической структуры с наследованием, в которой разные объекты оказываются на разных ветках дерева. Разумеется, ничто не мешает написать код, реализующий вашу собственную компонентную систему, но в *Unity* уже существует вполне надежный вариант такой системы, органично встроенный в визуальный редактор. Эта система дает возможность не только управлять компонентами программным образом, но и соединять и разрывать связи между ними в редакторе. Разумеется, возможности не ограничиваются составлением объектов из готовых деталей; в своем коде вы можете воспользоваться наследованием и всеми наработанными на его базе шаблонами проектирования [5, c.17].

**1.5 Игровой движок *Unreal Engine***

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

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

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

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

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

– широкий функционал;

– визуальное программирование;

– бесплатная лицензия;

– возможность создать кросс-платформер;

– большая база пользователей.

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

На рисунке 1.4 представлен интерфейс *Unreal Engine*.

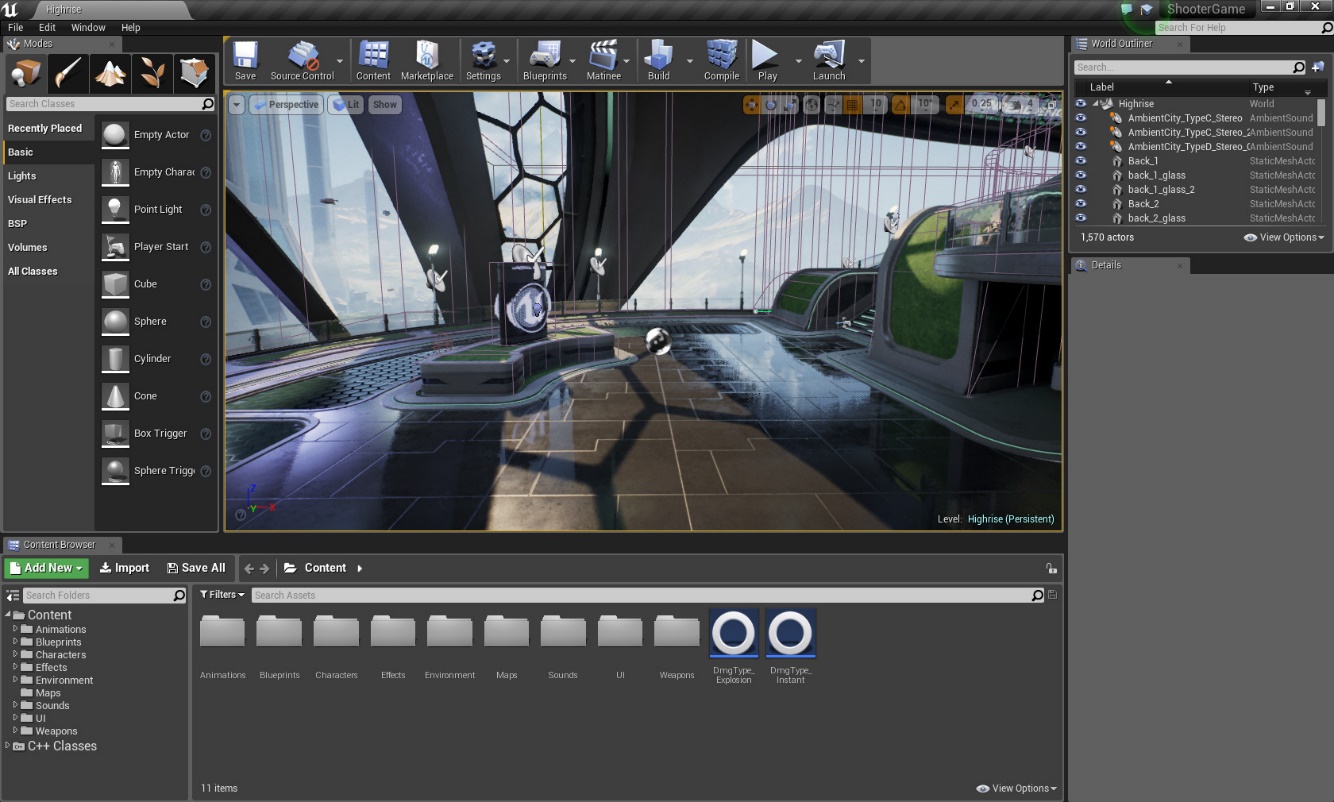


Рисунок 1.4 – интерфейс *Unreal Engine*

Для тех, кто далек от *кодинга*, корпорация предложила простую и удобную в использовании систему *Blueprints* *Visual* *Scripting*. С ее помощью можно легко создать прототип любой игры, имея минимум теоретических знаний. Конечно, умение работать с функциональным и объектно-ориентированным программированием будет плюсом, но начать разработку геймплея в *UE* можно и без него.

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

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

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

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

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

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

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

**1.6 Сравнение *Unity* и *Unreal Engine***

Первая область сравнения – редакторы для создания уровней, которые очень похожи. В них есть браузеры контента для ассетов, скриптов и других файлов проекта. Игровые объекты можно перетаскивать в область сцены и, таким образом, добавлять в её иерархию.

Объекты в редакторе сцены изменяются с помощью инструментов перемещения, поворота и масштабирования – они похожи в обоих движках. Свойства *Unity*-объектов отображаются в *Inspector*, а *UE4* – в части *Details*. *Jayanam* также сравнивает возможности *Unity* *Prefabs* c *Blueprints*.

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

Анимации, включённые для скелетных мешей, также можно импортировать. В *Unity* они прикрепляются к импортированному объекту, как клипы анимации (*animation clips*), а в *UE4* называются последовательностями анимации (*animation sequences*). В первом движения управляются с помощью контроллеров анимации (*animation* *controllers*), а во втором – по тому же принципу действуют анимационные *Blueprints*.

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

В *UE4* система называется *Persona*, а в *Unity* – *Mecanim*. В них возможно применение скелетных мешей одного скелета к другим, но в *Unity* это в основном используется для анимирования гуманоидов. В *UE4* анимации можно редактировать, в *Unity* – практически нет, особенно плохо дело обстоит с движениями гуманоидов. Движки не подходят для профессионального анимирования персонажей – лучше использовать программы вроде *Blender* или *Maya*, а результат импортировать в виде *FBX*-файлов. Прикреплённый к объектам материал добавляется в проект, но его свойства вроде шейдера или текстур придётся применять вручную [7].

Для этого в *Unity* нужно задать материалу шейдер и добавить в его слоты текстуры – карты шероховатостей, нормалей или диффузии. Собственные шейдеры придётся писать самостоятельно или с помощью сторонних инструментов вроде *Shader* *Forge* или *ASE*. А в *UE4* встроен очень мощный редактор материалов, основанный, как и система *Blueprints*, на нодах.

Для программирования в *UE4* используется язык *C++*, который не все любят из-за сложности и продолжительности компилирования. Однако *Jayanam* считает, что у движка понятный *API* и приемлемый период компиляции. В *UE4* очень мощная и проработанная система визуального скриптования – *Blueprints*, с помощью которой можно достичь практически тех же результатов, что и *c C++*.

*Unity 5* поддерживает языки *C#* и *UnityScript*. *API* и его концепт очень похож на аналог из *UE4*. При использовании управляемого языка вроде C#, программист не обязан использовать указатели (*pointers*), компилирование происходит быстро. В *Unity* нет системы визуального скриптования, и чтобы использовать что-то подобное, разработчик вынужден покупать сторонние дополнения вроде *Playmaker*.

Для *2D*-разработки в *Unity* есть великолепные инструменты – *sprite creator*, *sprite editor* и *sprite packer*. *UE4* также поддерживает спрайты в *Paper 2d*, но решения из *Unity* мощнее, кроме того, в последнем есть отдельный физический движок для *2d*-объектов.

В *UE4* встроен *постпроцессинг*. К сцене можно применять *bloom*-эффект, тонирование и антиалиасинг как глобально, так и к отдельным её частям (при помощи компонента *PostProcessVolume*).

В *Unity* есть стек постпроцессинга, который можно скачать из магазина ассетов движка. Система менее гибкая, чем в *UE4* – эффекты применяются только стеком или скриптами к камере.

*Sequencer* в *UE4* можно использовать для создания синематиков. Это мощный инструмент, работающий по принципу добавления объектов на временную шкалу. К настоящему моменту в *Unity* 5.6 нет системы для синематиков, но *timeline*-редактор добавили в *Unity* 2017.

Для реализации приложения был выбрал игровой движок *Unity*, т.к. он обладает наибольшим инструментарием и информационной базой, которые в один момент являются удобными и простыми [6].

1.7 Требования к проектируемому программному обеспечению

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

Программное обеспечение должно включать следующие функциональные возможности:

– cоздание нового игрового состояния: возможность выбрать название нового игрового состояния;

– загрузка выбранного игрового состояния: возможность выбора сохраненного игрового состояния;

– управление фермой: посадка и уход за различными культурами, управление ресурсами и инвентарем;

– социальное взаимодействие: общение с *NPC* (неигровыми персонажами);

– экономическая система: покупка и продажа продуктов, управление бюджетом фермы;

– атмосферное составляющее: смена дня и ночи;

Игровые механики включают:  
1. посадка и сбор урожая:

– механика вспашки земли;

– посадка семян различных культур;

– полив растений;

– сбор урожая по мере созревания;

2. механика загрузки и сохранения игрового состояния;

3. общение с *Npc*;

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

5. экономика:

– продажа продукции;

– покупка семян;

6. смена дня и ночи;

7. возможности настройки игровой среды:

– смена разрешения экрана;

– смена полноэкранного режима на оконный и наоборот;

– громкости музыки и звуковых эффектов;

– смена управления;

8. сохранение и загрузка настроек игровой среды.

Графические и визуальные элементы включают следующие пункты:

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

2. Персонажи: спрайты персонажей с анимацией движения;

3. Окружение: спрайты растительности, зданий и объектов;

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

5. Эффекты и анимация:

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

– визуальные эффекты для улучшения восприятия;

6. Звуковые эффекты и музыка:  
 – эффекты для действий;

– фоновая музыка, создающая атмосферу спокойствия и умиротворенности.

Входные данные:  
 – данные о игровом состоянии (игровая валюта);

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

Условия эксплуатации:

– программное обеспечение должно работать на платформе *Windows*;

– поддержка разрешений экрана от *1024x768* и выше;

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

– обеспечение возможности сохранения и загрузки игрового процесса.

Требования к надежности и быстродействию:

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

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

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

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

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

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

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

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

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

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

– четкое определение структуры приложения и его компонентов;

– оценка сложности приложения и определение возможных рисков и проблем;

– обеспечение гибкости и расширяемости приложения в будущем;

– упрощение процесса разработки за счет более четкого понимания того, что нужно разрабатывать и как компоненты приложения взаимодействуют друг с другом;

– улучшение качества кода и снижение затрат на его сопровождение.

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

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

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

2.1 Определение основных функций

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



Рисунок 2.1 – Основные функциональные задачи

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

Были выделены следующие задачи и подзадачи:

а) обновление игровых объектов;

б) отрисовка графики;

в) визуальные и звуковые эффекты;

г) социальное взаимодействие:

– общение с NPC;

д) экономическая система:

– покупка и продажа продукции;

е) управление фермой:

– посадка урожая;

– сбор урожая;

– уход за растениями;

ж) управление ресурсами:

– активный инвентарь;

– дополнительный или вспомогательный инвентарь;

– сундук или сундуки;

з) смена дня и ночи;

к) сохранение и загрузка игрового состояния;

л) настройка игровой среды;

м) сохранение и загрузка настроек игровой среды.

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

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

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

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

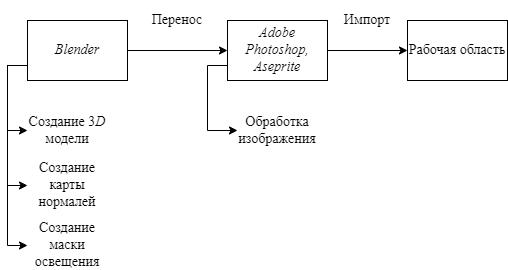


Рисунок 2.2 – Схема инициализации компонентов, описанных в пунктах б и в

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

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

2.2 Структурные элементы

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

Основные структурные элементы, которые будут использоваться в проекте включают:

– модуль управления конструированием игрового мира;

– модуль управления фермой;

– модуль социального взаимодействия;

– модуль экономической системы;

– модуль управления инвентарем;

– модуль визуальных и звуковых эффектов;

– модуль времени;

– модуль сохранения и загрузки;

– модуль управления игровыми сценами;

– модуль инициализации.

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

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

Схема модуля показана на рисунке 2.3.



Рисунок 2.3 – Схема работы модуля

Пользователь будет взаимодействовать с вышеописанными модулями через графический пользовательский интерфейс (*GUI*). Разделение работы модуля на взаимодействие с *GUI* и логическую составляющую является хорошей практикой, позволяющей улучшить модульность и управляемость кода. *GUI* будет служить мостом между пользователем и логикой приложения, предоставляя интуитивно понятные элементы управления и обратную связь.

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

Модуль управления фермой отвечает за любые аспекты управления фермой:  
 – посадка урожая;

– сбор урожая;

– рыхление земли;

– полив растений;

– сбор продукции по мере роста.

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

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

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

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

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

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

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

Модуль управления игровыми сценами отвечает за смену игровых сцен.

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

2.3 Устройство внутренней архитектуры и схема разделения данных

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

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

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

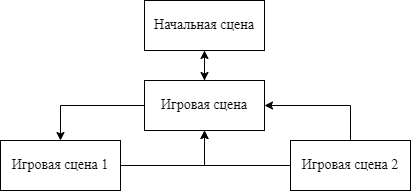


Рисунок 2.4 – Схема сцен

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

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



Рисунок 2.5 – Схема наличия модулей на начальной и игровой сцене

После запуска сцены первым делом запускается модуль инициализации.

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

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



Рисунок 2.6 – Схема инициализации программных модулей после запуска сцены

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

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

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

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

Паттерн внедрения зависимостей (*Dependency Injection Pattern*) – это дизайн-шаблон, который использует принцип *IoC*, чтобы обеспечить, что класс не принимает участие в создании экземпляра зависимого класса и не управляет его жизненным циклом. Иными словами, процесс создания класса и инициализации его переменных, поступающих через конструктор или методы, полностью возлагается на платформу, то есть на более высокий уровень.

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

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

Для чего нужен *DI*-контейнер:

– создание экземпляров объектов, включая разрешение зависимостей, в том числе и иерархических;

– управление жизненным циклом объектов (*lifetime*);

– доступ из любого места в программе, практически в любом конструкторе.

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

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

Данные в приложении разделяются на несколько типов:

– игровые данные;

– экономические данные;

– визуальные и звуковые данные;

– временные данные;

– данные для загрузки и сохранения.

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

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

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

Звуковые данные относятся к звуковым эффектам и фоновой музыке.

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

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

На рисунке 2.7 представлена схема взаимодействия сущностей.

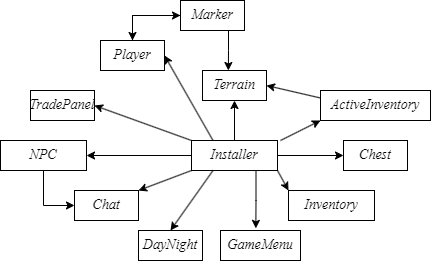


Рисунок 2.7 – Схема взаимодействия сущностей

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

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

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

Сущность *InventoryItem* содержит данные об предмете инвентаря.

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

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

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

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

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

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

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

2.4 Архитектура пользовательского интерфейса

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

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

Начальная сцена будет представлять из себя главное меню, где пользователь с помощью *GUI* будет запускать игровую сцену. На рисунке 2.5 представлен каркас пользовательского интерфейса на начальной сцене.

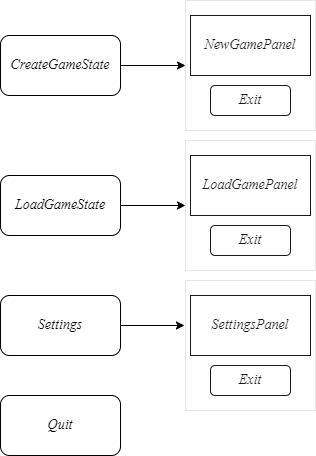


Рисунок 2.5 – Каркас пользовательского интерфейса на начальной сцене

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

– кнопка, которая отображает панель создания нового игрового состояния;

– панель создания нового игрового состояния;

– кнопка, которая отображает панель загрузки игрового состояния;

– панель загрузки игрового состояния;

– кнопка, которая отображает панель настроек игровой среды.

– панель настроек игровой среды;

– кнопка, с помощью которой можно закрыть игровое приложения.

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

В игровой сцене также должны присутствовать следующие элементы пользовательского интерфейса:

– кнопка, которая отвечает за продолжение игрового сеанса;

– кнопка, которая отображает панель настроек игровой среды;

– кнопка, которая отвечает за переход на начальную сцену.

Каркас пользовательского интерфейса на игровой сцене показан на рисунке 2.6.

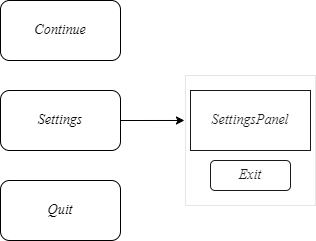


Рисунок 2.6 – Каркас пользовательского интерфейса на игровой сцене

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

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

Все элементы *HUD* по сути являются отдельными интерфейсами, каждый из которых существует для передачи игроку информации определенного типа и решения своей уникальной задачи. При этом *HUD*, полностью или частично, практически всегда находится непосредственно перед глазами игрока, так как все его элементы отображаются на переднем плане геймплея и/или в игровом мире во время игры.

В игровой сцене необходимо придумать каскад для следующих элементов интерфейса:

– текущее время суток;

– игровая валюта;

– активный инвентарь;

– вспомогательный инвентарь.

На рисунке 2.8 представлен каркас для игрового *HUD*.

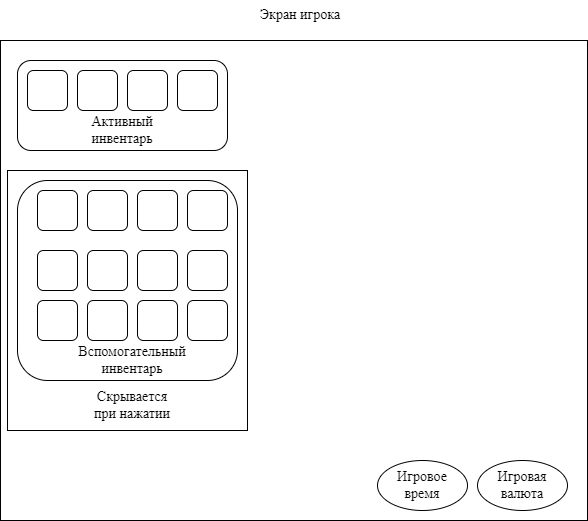


Рисунок 2.8 – Каркас для игрового *HUD*

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

3 СТРУКТУРА ИГРОВОГО ПРИЛОЖЕНИЯ «*FARMER'S VALLEY*»