Содержание

[Введение 3](#_Toc203081320)

[1 Общая характеристика предприятия прохождения практики 4](#_Toc203081321)

[1.1 Описание предприятия 4](#_Toc203081322)

[1.2 Описание продуктов предприятия 4](#_Toc203081323)

[1.3 Описание использующихся технологий 5](#_Toc203081324)

[2 Индивидуальное задание 7](#_Toc203081325)

[2.1 Анализ систем-аналогов 7](#_Toc203081326)

[2.2 Проектирование архитектуры системы 8](#_Toc203081327)

[2.3 Реализация программного продукта 9](#_Toc203081328)

[2.3.1 Система инвентаря и управления предметами 10](#_Toc203081329)

[2.3.2 Система животных: жизненный цикл, взаимодействие 10](#_Toc203081330)

[2.3.3 Экономическая модель: кошелек игрока и торговые лавки 12](#_Toc203081331)

[2.3.4 Система квестов и отслеживания прогресса 14](#_Toc203081332)

[2.3.5 Система игрового прогресса и опыта 16](#_Toc203081333)

[2.3.6 Управление камерой и навигация 16](#_Toc203081334)

[Заключение 17](#_Toc203081335)

# Введение

Производственная технологическая практика представляет собой важнейший элемент образовательной программы, направленной на подготовку квалифицированных специалистов в области информационных технологий. Её значение заключается в том, что она обеспечивает тесную взаимосвязь между полученными в ходе обучения теоретическими знаниями и реальными условиями профессиональной деятельности.

Целью данной практики является закрепление теоретических основ, усвоенных в процессе обучения, а также приобретение практического опыта в решении прикладных задач, возникающих в деятельности ИТ-специалистов. Во время прохождения практики студенту предстоит погрузиться в рабочую среду, ознакомиться с деятельностью компании, понять внутренние процессы, участвующие в разработке программных продуктов, и принять участие в выполнении конкретных производственных заданий.

Особое внимание в процессе практики уделяется изучению организационной структуры предприятия, роли и функций ИТ-специалистов, а также применяемых в компании методов управления проектами. Необходимо познакомиться с программным обеспечением, используемым на предприятии, а также с техническими средствами, обеспечивающими его работу. Кроме того, практика предоставляет возможность изучить подходы к обеспечению информационной безопасности и стандартам, применяемым в разработке программных решений. Одним из значимых аспектов практики является возможность перенять опыт у действующих специалистов, наблюдая за их работой и получая консультации по конкретным рабочим ситуациям.

По завершении необходимо подготовить отчет, в котором будет отражён процесс прохождения практики, полученные знания и приобретённые навыки. Отчёт должен соответствовать установленным требованиям и содержать анализ выполненной работы, а также информацию о структуре предприятия, применяемых технологиях и методах.

Производственная технологическая практика проходит на базе компании «Сабер БГС» в период с 23 июня по 18 июля 2025 года. Участие в работе действующего предприятия создаёт для студента уникальные условия для профессионального роста. Это позволяет не только систематизировать и углубить знания, полученные в процессе обучения, но и развить практические навыки, необходимые для успешного трудоустройства и дальнейшего карьерного развития в сфере информационных технологий.

# Общая характеристика предприятия прохождения практики

## Описание предприятия

«Сабер БГС» – это современная белорусско-американская компания, специализирующаяся на разработке и сопровождении видеоигр, и входящая в состав международной группы «Saber Interactive», являющейся частью крупного игрового холдинга «Embracer Group». Компания занимает важное место в мировой индустрии цифровых развлечений и ориентирована на выпуск игровых продуктов для широкого круга платформ.

Сегодня «Сабер БГС» участвует как в создании собственных игровых проектов, так и в сотрудничестве с международными партнёрами. Головной офис компании расположен в Нью-Джерси, США, в то время как команды разработчиков работают в разных странах мира – включая Россию, Беларусь, Испанию и Швецию. Минский офис, в частности, выполняет ключевую роль, сосредотачивая в себе одну из ведущих команд, которая участвует в полном цикле разработки, адаптации и поддержки игровых продуктов для различных платформ.

Организационная структура компании представляет собой сочетание классической иерархической модели и элементов проектного управления. Работа над каждым проектом ведётся командами, в состав которых входят специалисты разных направлений – программисты, дизайнеры, художники, тестировщики, менеджеры и аналитики. Управление проектами осуществляется с применением гибких методологий, что позволяет эффективно организовать взаимодействие между отделами и оперативно реагировать на изменения в ходе разработки. Команды используют современные цифровые инструменты и платформы для координации работы в рамках международного сотрудничества.

В процессе работы сотрудники принимают участие как в полном цикле создания игровых продуктов – от проектирования и программирования до тестирования и выпуска – так и в отдельных стадиях: например, адаптации графики под определённые устройства или оптимизации производительности на конкретной платформе.

## 1.2 Описание продуктов предприятия

Ключевым направлением деятельности компании «Сабер БГС» является создание и поддержка видеоигр для широкого спектра устройств, включая персональные компьютеры, игровые консоли и мобильные устройства. Разработка ведётся как по собственным проектам компании, так и в рамках глобальных партнёрских инициатив. Среди наиболее известных игр, к созданию или поддержке которых компания имела непосредственное отношение, можно выделить такие проекты, как World War Z, SnowRunner, Evil Dead: The Game и ряд других.

Компания также занимается технической адаптацией игр на новые платформы, обеспечением их стабильной работы, выпуском обновлений и патчей, а также портированием игровых проектов между различными устройствами. Важно отметить, что в работе учитываются как аппаратные характеристики целевых платформ, так и требования к визуальному качеству, отзывчивости интерфейса и общей производительности.

Помимо разработки самих игр, специалисты компании создают собственные программные инструменты, которые используются для автоматизации процессов, визуализации и взаимодействия с пользователем. Эти инструменты позволяют упростить процесс разработки, сделать его более гибким и управляемым, а также повысить качество конечного продукта.

## 1.3 Описание использующихся технологий

В своей деятельности «Сабер БГС» применяет широкий спектр современных технологий, языков программирования и инструментов, что позволяет успешно реализовывать сложные проекты и адаптировать их под различные операционные системы и устройства.

Основу технической части разработки составляют языки C++ и C#. C++ преимущественно используется при создании системной логики и реализации производительно-критичных участков кода, тогда как C# активно применяется в работе с игровыми движками, такими как Unity3D, а также при проектировании интерфейсов и внутренних игровых механизмов.

Разработка охватывает множество платформ: от Windows, Android и iOS до PlayStation, Xbox, Nintendo Switch и даже устаревающей, но всё ещё поддерживаемой Windows Phone. Для каждой платформы разрабатываются индивидуальные решения с учётом её архитектурных и технических особенностей, включая управление ресурсами, типы ввода, работу с экраном и др.

Значительное внимание уделяется также веб-технологиям. В частности, используется WebGL для визуализации графики непосредственно в браузере, что позволяет создавать кроссплатформенные приложения и облегчённые версии игровых интерфейсов. Кроме того, компания активно применяет open-source решения, что обеспечивает гибкость в выборе технологий и снижает зависимость от закрытых, лицензируемых платформ.

При разработке многопользовательских функций и взаимодействии клиента с сервером используются такие технологии, как .NET Sockets, обеспечивающие передачу данных в реальном времени. Также применяются протоколы на базе HTTP и REST API, особенно в тех проектах, где требуется асинхронное взаимодействие с серверной частью.

Работа с данными организована с использованием современных СУБД – таких как PostgreSQL и MySQL. Они используются для хранения различной информации: от пользовательских аккаунтов и игровых настроек до статистики, конфигурационных файлов и служебных данных. Управление базами данных осуществляется как на стороне серверов, так и в клиентских приложениях при необходимости.

Центральное место в технологическом стекe компании занимает игровой движок Unity3D, который обеспечивает кроссплатформенную разработку, удобную визуализацию игровых сцен и поддержку широкого набора графических и физических эффектов. С помощью Unity реализуются как двумерные, так и трёхмерные проекты, в том числе с использованием сложной анимации, освещения, теней и физики.

Графическая часть игры создаётся с применением технологий 2D и 3D-визуализации, включая работу с шейдерами, текстурами, визуальными эффектами, моделями и анимацией. Для этого используются как собственные, так и сторонние инструменты, обеспечивающие высокое качество графики и её плавную интеграцию в игровую среду.

«Сабер БГС» представляет собой высокотехнологичную, международную компанию, активно использующую современные инструменты разработки и подходы к реализации игровых проектов. Опыт работы в такой компании позволяет получить глубокое представление о процессе создания цифровых продуктов мирового уровня.

# Индивидуальное задание

Производственная практика проходила в рамках разработки игрового проекта в жанре симулятора фермы. Основной целью практики являлось применение теоретических знаний, полученных в ходе обучения, для решения реальных прикладных задач в области разработки программного обеспечения, а именно — видеоигр на движке Unity.

Ключевые задачи практики включали:

* Участие в командной разработке в роли ведущего программиста.
* Проектирование и реализация сложных игровых механик.
* Работа с системой контроля версий и инструментами совместной работы.
* Разработка архитектуры программного кода для обеспечения его масштабируемости.
* Взаимодействие с другими отделами (гейм-дизайн, арт) для интеграции контента в игру.
* Соблюдение установленных сроков и работа в рамках заданной методологии.

Проект представляет собой 2D-симулятор, сочетающий в себе элементы фермерства и торговли. Главной отличительной чертой игры является ее сеттинг: вся жизнь и деятельность игрока сосредоточена на постоянно движущемся поезде. Поезд периодически прибывает на станции, где игрок может торговать, получать новые задания и взаимодействовать с миром, после чего снова отправляется в путь.

## Анализ систем-аналогов

Перед началом активной разработки был проведен анализ ключевого референса — игры Stardew Valley. Это было необходимо для понимания основополагающих механик, которые делают жанр привлекательным для игроков, и для определения точек, в которых наш проект мог бы предложить уникальный опыт.

Основной игровой цикл:

* Игрок выполняет базовые действия (полив растений, уход за животными, добыча ресурсов в шахте, рыбалка).
* За свои действия игрок получает ресурсы (урожай, продукция животных, руда).
* Полученные ресурсы можно продать, чтобы заработать деньги. Деньги, в свою очередь, инвестируются в улучшение инструментов, покупку новых семян, строительство зданий, что делает первоначальные действия более эффективными. Этот цикл создает сильное чувство прогрессии и вовлеченности. Мы стремились воссоздать его в нашем проекте.
* Клик-ориентированное управление для взаимодействия с объектами, персонажами и миром было признано наиболее интуитивным для 2D-симулятора и взято за основу.

## Проектирование архитектуры системы

Вместо статичной фермы у игрока есть собственный поезд с вагонами, которые можно обустраивать: вагоны-грядки, вагоны-загоны для животных, вагон-мастерская и т.д.

Игровой процесс четко разделен на две фазы:

* Фаза «Поезд»: пока поезд находится в пути, игрок занимается фермерством, уходом за животными, крафтом. Это медитативная фаза, сфокусированная на менеджменте своей «мобильной фермы».
* Фаза «Станция»: поезд прибывает на станцию. Здесь игрок может продавать товары, покупать уникальные предметы. Это активная и торговая фаза.

Вместо одного большого открытого мира игра предлагает последовательность станций. Выполнение ключевых квестов на станции позволяет отправиться на следующий, более сложный и богатый ресурсами участок пути.

На основе концепции были сформулированы основные задачи для команды программистов:

1. Создать модульную систему управления вагонами и камерой.
2. Реализовать комплексную систему инвентаря с поддержкой основного и дополнительного инвентаря и его улучшений.
3. Разработать систему животноводства с уникальными циклами производства для каждого вида животных.
4. Создать гибкую систему магазинов для станций.
5. Реализовать систему квестов, поддерживающую разные типы целей.
6. Спроектировать систему опыта (XP) для управления переходом между фазами «Поезд» и «Станция».

Проект разрабатывался большой командой, что требовало четкой организации. Команда включала:

* Тимлид: ответственный за общее видение проекта, управление командой и принятие финальных решений.
* Гейм-дизайнеры: проектировали игровые механики, баланс, квесты и контент.
* Художники: создавали спрайты персонажей, окружения, предметов и UI.
* Программисты: реализовывали игровую логику.

В этой структуре я занимала позицию ведущего программиста. Мои обязанности включали:

* Принятие решений о том, как будут взаимодействовать ключевые системы (менеджеры, Scriptable Objects, события).
* Реализация наиболее сложных систем. Я лично отвечала за написание кода для систем инвентаря, животных, магазинов, квестов и камеры.
* Проверка кода, написанного другими программистами, на соответствие стандартам, производительность и логику.
* Техническая декомпозиция задач: Разбиение крупных задач от гейм-дизайнеров на конкретные подзадачи для команды программистов.
* Помощь другим программистам в решении сложных задач и объяснение архитектуры проекта.

Работа была организована по гибкой методологии, близкой к Scrum. Весь процесс разработки был разделен на четыре двухнедельных спринта.

* В начале каждого спринта тимлид и ведущие специалисты проводили встречу, где определяли цели на следующие две недели. Задачи из общего Roadmap выбирались и помещались в план на текущий спринт.
* Короткие утренние собрания, где каждый член команды отвечал на три вопроса: «Что я сделал вчера?», «Что я буду делать сегодня?», «Какие у меня есть проблемы/блокираторы?». Это позволяло быстро решать возникающие трудности и синхронизировать работу.
* В конце спринта проводилась демонстрация проделанной работы. Команда показывала тимлиду и гейм-дизайнерам работающий билд с новыми функциями.
* После демонстрации команда обсуждала, что в прошедшем спринте прошло хорошо, а что можно было бы улучшить в процессах работы.

Такой подход позволил гибко реагировать на изменения, регулярно получать обратную связь и поддерживать высокий темп разработки.

В качестве основного инструмента для ведения задач и отслеживания прогресса использовался документ в Google Docs, который выполнял роль Roadmap (дорожной карты).

## Реализация программного продукта

В самом начале разработки необходимо заложить прочный фундамент для всего проекта. Игра предполагала множество взаимосвязанных систем: инвентарь, экономика, квесты, животные и т.д. Была спроектирована архитектура, основанная на двух ключевых принципах: централизованные менеджеры и разделение данных и логики.

Для управления глобальными сущностями, такими как инвентарь игрока, его кошелек или состояние квестов, был использован паттерн «Синглтон» (Singleton) – это класс, который гарантирует, что в игре существует только один его экземпляр, и предоставляет глобальную точку доступа к этому экземпляру. Это позволило из любой части кода безопасно обратиться, например, к InventoryManager.Instance, чтобы добавить предмет, или к PlayerWallet.Instance, чтобы проверить баланс.

Чтобы системы не были жестко связаны друг с другом, была реализована система глобальных событий, основанная на паттерне «Наблюдатель» (Observer). Для этого был создан статический класс GameEvents. Вместо того чтобы, например, система животных напрямую вызывала методы в системе квестов, она просто «объявляла» о событии. Система квестов, в свою очередь, была «подписана» на это событие и реагировала на него. Такой подход обеспечил слабую связанность модулей: они общаются через посредника, не зная друг о друге, что значительно упрощает их поддержку и расширение.

### 2.3.1 Система инвентаря и управления предметами

Для обеспечения гибкости использована технология ScriptableObject — особый тип ассетов в Unity, позволяющий хранить данные независимо от игровых сцен. Создан ItemData — шаблон для всех предметов в игре, содержащий статическую информацию: название, иконку, описание, тип (инструмент, семена, продукт), флаг стакируемости и максимальный размер стопки. Такой подход позволяет гейм-дизайнерам создавать сотни предметов, просто добавляя новые ассеты без написания кода.

Для хранения предметов в инвентаре используется вспомогательный класс InventoryItem с полями: ссылка на ItemData и количество. Это обеспечивает разделение шаблона и конкретного экземпляра предмета.

Ядро системы — синглтон-класс InventoryManager, управляющий списком предметов. Он реализует добавление с учётом стаков, удаление и поиск свободных слотов. Поддерживается улучшение инвентаря: изначально доступен хотбар и одна строка. При покупке предмета-улучшения через магазин система через TrainUpgradeManager проверяет статус и при необходимости увеличивает размер инвентаря, активируя новые ряды слотов в UI.

Для отображения инвентаря используется префаб слота со скриптом InventorySlotUI. Перетаскивание предметов реализовано через стандартные интерфейсы Unity: IBeginDragHandler, IDragHandler, IEndDragHandler, IDropHandler. В процессе перетаскивания создаётся временная иконка, следящая за курсором. При отпускании мыши над другим слотом вызывается OnDrop, и InventoryManager перемещает предмет. Это обеспечивает удобное и привычное взаимодействие с системой инвентаря.

### 2.3.2 Система животных: жизненный цикл, взаимодействие

Для хранения информации о типах животных применялся ScriptableObject под названием AnimalData. В нем содержатся данные о префабе животного, спрайты для различных состояний (обычное, с готовым продуктом), звуки, требуемый корм, производимый продукт, удобрение и временные интервалы для каждого этапа жизненного цикла.

Чтобы отслеживать состояние каждого конкретного животного, был создан отдельный класс AnimalStateData. Он содержит ссылку на AnimalData, текущую фазу производственного цикла и таймер до следующего события. Такой подход позволил отделить данные состояния от визуального представления животного на сцене.

AnimalPenManager — синглтон, выступающий в роли базы данных всех животных, принадлежащих игроку. Он работает исключительно со списком AnimalStateData, не взаимодействуя напрямую с игровыми объектами. Отвечает за добавление и удаление данных при покупке и продаже животных, а также за проверку вместимости загонов в зависимости от уровня их улучшения.

TrainPenController — скрипт, обеспечивающий связь между данными и игровым миром. При загрузке сцены поезда он запрашивает у AnimalPenManager список животных, после чего с помощью создает соответствующие префабы на сцене.

AnimalController — компонент префаба животного, управляющий его поведением.

Для движения реализована простая машина состояний (State Machine) с режимами «Стоит» (Idle) и «Идет» (Walking). Животное случайным образом переключается между этими состояниями, выбирая новую точку для перемещения в пределах своего загона.

Вторая, более сложная машина состояний отвечает за производственный процесс. Состояния включают: «Ждет корма», «Готов продукт», «Готово удобрение».

Реализовано взаимодействие с игроком: в зависимости от текущего состояния он пытается покормить животное (проверяя предмет в руках игрока) или собрать продукт/удобрение (проверяя инвентарь).

Листинг 2.1 взят из класса AnimalController.cs. Он представляет собой ключевой метод TransitionToNextProductionState, отвечающий за логику смены состояний в производственном цикле животного. Этот метод является сердцем машины состояний.

Вместо жестко закодированной последовательности состояний, метод сначала определяет идеальный следующий шаг (например, после кормления идет производство продукта). Затем он входит в цикл, в котором проверяет условия, хранящиеся в AnimalData. Если, к примеру, для данного типа животного количество производимого продукта (productAmount) равно нулю, система пропустит этот этап и перейдет к проверке следующего — производству удобрения.

Такой подход, основанный на данных, позволяет гейм-дизайнерам создавать совершенно разные типы животных без изменения кода. Достаточно настроить параметры в соответствующем AnimalData, и логика жизненного цикла животного адаптируется автоматически. Это значительно повышает гибкость и скорость итераций при добавлении нового контента в игру.

|  |
| --- |
| public void AddQuestProgress(GoalType eventType, string targetID, int amount)  {  foreach (var quest in new List<Quest>(ActiveQuests))  {  bool questProgressed = false;  foreach (var goal in quest.goals.Where(g => !g.IsReached()))  {  bool progressMadeOnThisGoal = false;  if (goal.goalType == eventType)  {  switch (goal.goalType)  {  case GoalType.Gather:  case GoalType.Buy:  case GoalType.FeedAnimal:  if (goal.targetID == targetID) progressMadeOnThisGoal = true;  break;  case GoalType.GatherAny:  case GoalType.BuyAny:  if (goal.targetIDs.Contains(targetID)) progressMadeOnThisGoal = true;  break;  case GoalType.Earn:  case GoalType.SellFor:  progressMadeOnThisGoal = true;  break;  }  }  if (progressMadeOnThisGoal)  {  goal.UpdateProgress(amount);  questProgressed = true;  }  }  if (questProgressed)  {  OnQuestLogUpdated?.Invoke();  CheckQuestCompletion(quest);  }  }  } |

Листинг 2.1 – Реализация жизненного цикла животного

### 2.3.3 Экономическая модель: кошелек игрока и торговые лавки

Для создания полноценного игрового цикла была реализована экономическая система.

PlayerWallet — синглтон, хранящий текущее количество денег. При добавлении средств вызывается событие OnMoneyAdded, передающее сумму. Это необходимо для квестов с целью «заработать N денег».

ShopInventoryData (ScriptableObject) — определяет ассортимент конкретного магазина, цены покупки и продажи, а также доступность товара.

ShopDataManager — синглтон, отслеживающий запасы товаров на складе. При покупке или продаже соответствующие количества изменяются.

ShopUIManager и ShopItemRow — отвечают за отображение UI магазина. Ключевая задача — корректное управление активностью кнопки «Купить». ShopItemRow выполняет несколько проверок: наличие денег у игрока, доступность товара, наличие места в загоне или инвентаре, а также статус покупки улучшений. Только при выполнении всех условий кнопка становится активной.

Фрагмент из листинга 2.2 отвечает за логику отображения одного товара в интерфейсе магазина. Метод Setup является отличным примером интеграции нескольких игровых систем для принятия одного решения: может ли игрок купить данный товар?

Код последовательно обращается к разным менеджерам:

1. PlayerWallet.Instance — для проверки наличия достаточного количества денег.
2. ShopDataManager.Instance — для проверки остатка товара на складе.
3. TrainUpgradeManager.Instance — для проверки, не было ли данное улучшение уже куплено ранее.
4. AnimalPenManager.Instance — для проверки, есть ли свободное место в загоне для нового животного.
5. InventoryManager.Instance — для проверки, есть ли свободное место в инвентаре для обычных предметов.

На основе результатов этих проверок определяется, будет ли кнопка «Купить» активной. Такой подход не только предотвращает ошибки, но и обеспечивает интуитивно понятный интерфейс для игрока, визуально показывая, какие товары доступны для покупки в данный момент.

|  |
| --- |
| public void AddQuestProgress(GoalType eventType, string targetID, int amount)  {  foreach (var quest in new List<Quest>(ActiveQuests))  {  bool questProgressed = false;  foreach (var goal in quest.goals.Where(g => !g.IsReached()))  {  bool progressMadeOnThisGoal = false;  if (goal.goalType == eventType)  {  switch (goal.goalType)  {  case GoalType.Gather:  case GoalType.Buy:  case GoalType.FeedAnimal:  if (goal.targetID == targetID) progressMadeOnThisGoal = true;  break;  case GoalType.GatherAny:  case GoalType.BuyAny:  if (goal.targetIDs.Contains(targetID)) progressMadeOnThisGoal = true;  break;  case GoalType.Earn:  case GoalType.SellFor:  progressMadeOnThisGoal = true;  break;  }  }  if (progressMadeOnThisGoal)  {  goal.UpdateProgress(amount);  questProgressed = true;  }  }  if (questProgressed)  {  OnQuestLogUpdated?.Invoke();  CheckQuestCompletion(quest);  }  }  } |

Листинг 2.2– Динамическая проверка доступности товара в магазине

### 2.3.4 Система квестов и отслеживания прогресса

Система квестов построена на событийной архитектуре.

Quest (ScriptableObject) содержит данные о задании: текст, цели, награду и ссылку на следующий квест, если предусмотрена цепочка. Каждая цель описывается с помощью класса QuestGoal, включающего тип (например, Gather, Buy, Earn) и требуемое количество.

QuestManager подписывается на события от других менеджеров. При срабатывании событий вызывается метод, который сверяет события с активными квестами и увеличивает прогресс по соответствующим целям. Когда все цели выполнены, квест считается завершенным, игрок получает награду, и активируется следующий квест в цепочке.

Реализованы UI-компоненты для журнала квестов, трекера на основном экране и всплывающих уведомлений.

Данный фрагмент кода (листинг 2.3) взят из класса QuestManager.cs и является ядром системы отслеживания квестов. Метод AddQuestProgress демонстрирует применение событийно-ориентированного подхода. Он вызывается различными игровыми системами (инвентарь, кошелек, магазин) при наступлении определенных событий. Метод не зависит от источника вызова, а оперирует только абстрактными данными: типом события, идентификатором цели и количеством.

Внутри метода происходит итерация по всем активным квестам. С помощью конструкции switch определяется, соответствует ли произошедшее событие цели квеста. Такая универсальная реализация позволяет легко расширять систему новыми типами целей, не изменяя существующий код других модулей, что является признаком хорошо спроектированной архитектуры.

|  |
| --- |
| public void AddQuestProgress(GoalType eventType, string targetID, int amount)  {  foreach (var quest in new List<Quest>(ActiveQuests))  {  bool questProgressed = false;  foreach (var goal in quest.goals.Where(g => !g.IsReached()))  {  bool progressMadeOnThisGoal = false;  if (goal.goalType == eventType)  {  switch (goal.goalType)  {  case GoalType.Gather:  case GoalType.Buy:  case GoalType.FeedAnimal:  if (goal.targetID == targetID) progressMadeOnThisGoal = true;  break;  case GoalType.GatherAny:  case GoalType.BuyAny:  if (goal.targetIDs.Contains(targetID)) progressMadeOnThisGoal = true;  break;  case GoalType.Earn:  case GoalType.SellFor:  progressMadeOnThisGoal = true;  break;  }  }  if (progressMadeOnThisGoal)  {  goal.UpdateProgress(amount);  questProgressed = true;  }  }  if (questProgressed)  {  OnQuestLogUpdated?.Invoke();  CheckQuestCompletion(quest);  }  }  } |

Листинг 2.3 – Универсальный обработчик прогресса квестов

### 2.3.5 Система игрового прогресса и опыта

Система отвечает за глобальное продвижение и переходы между фазами «Поезд» и «Станция».

ExperienceManager — синглтон, хранящий уровень игры, текущую фазу и количество опыта, необходимого для перехода на следующий уровень. При получении награды от QuestManager происходит вызов метода, который, если опыт превышает порог, запускает переход на следующую фазу.

### 2.3.6 Управление камерой и навигация

Так как в игре используются разные сцены, были разработаны два отдельных контроллера камеры.

TrainCameraController — реализует плавное перемещение камеры между вагонами. Поддерживаются режимы приближения для детального взаимодействия и отдаления для свободного обзора всего состава.

StallCameraController — аналогично работает на сцене станции, фокусируясь на торговых точках.

Оба контроллера используют интерполяцию для плавных переходов камеры.

# Заключение

Прохождение производственной практики стало для меня ключевым этапом в переходе от теоретического обучения к реальной проектной деятельности. Мне удалось не просто познакомиться с внутренними процессами разработки в IT-команде, но и принять в них активное участие в ответственной роли программиста. Практика позволила глубоко погрузиться во все этапы создания цифрового продукта — от анализа аналогов и архитектурного проектирования до реализации сложных механик и подготовки итоговой демонстрации. Я на собственном опыте смогла оценить важность командной работы, строгого соблюдения сроков и высоких требований, предъявляемых к качеству программного кода.

Особое значение для меня имело выполнение поставленной задачи – разработка ключевых систем для игрового прототипа на движке Unity. В процессе работы я спроектировала и реализовала гибкую, событийно-ориентированную архитектуру. Мной были созданы сложные системы инвентаря с динамическим расширением, животноводства с уникальными машинами состояний, а также комплексные системы квестов и экономики. Это позволило мне закрепить и углубить знания в области объектно-ориентированного программирования, проектирования архитектуры, а также познакомиться с принципами построения комплексных игровых симуляторов.

Помимо непосредственной разработки, важной частью практики стало участие в организационных и методологических процессах. Работа была выстроена с разделением на четырехнедельные спринты и отслеживанием задач через общую дорожную карту. Регулярные командные обсуждения, планирование этапов и технические митапы под руководством тимлида помогли лучше понять как технические, так и организационные аспекты современной разработки.

Вся работа велась в тесном сотрудничестве с командой: мы совместно обсуждали идеи, искали оптимальные решения, распределяли задачи и помогали друг другу в их реализации. Финальные этапы практики были посвящены интеграции всех систем в единый прототип, а также подготовке демонстрационных материалов, что позволило увидеть результат нашего общего труда.

Практика дала мне бесценный опыт как самостоятельной архитектурной работы, так и эффективного командного взаимодействия. Я научилась более точно оценивать сложность задач, анализировать требования гейм-дизайнеров и находить элегантные решения для сложных технических вызовов. Полученные знания и навыки станут прочной основой для моего дальнейшего профессионального развития и, я уверена, помогут в успешной карьере в индустрии разработки видеоигр.