МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра информационных систем и цифровых технологий

Работа допущена к защите
_____Руководитель
«19» 12 2026 г.

КУРСОВОЙ ПРОЕКТ

по дисциплине «Выпуск и сопровождение программных продуктов»

на тему: «Разработка системы контроля версий состояний игры “Blackjack”»

Студент _____ Василения И.В.

Шифр 220887

Институт приборостроения, автоматизации и информационных технологий

Направление подготовки 09.03.04 Программная инженерия

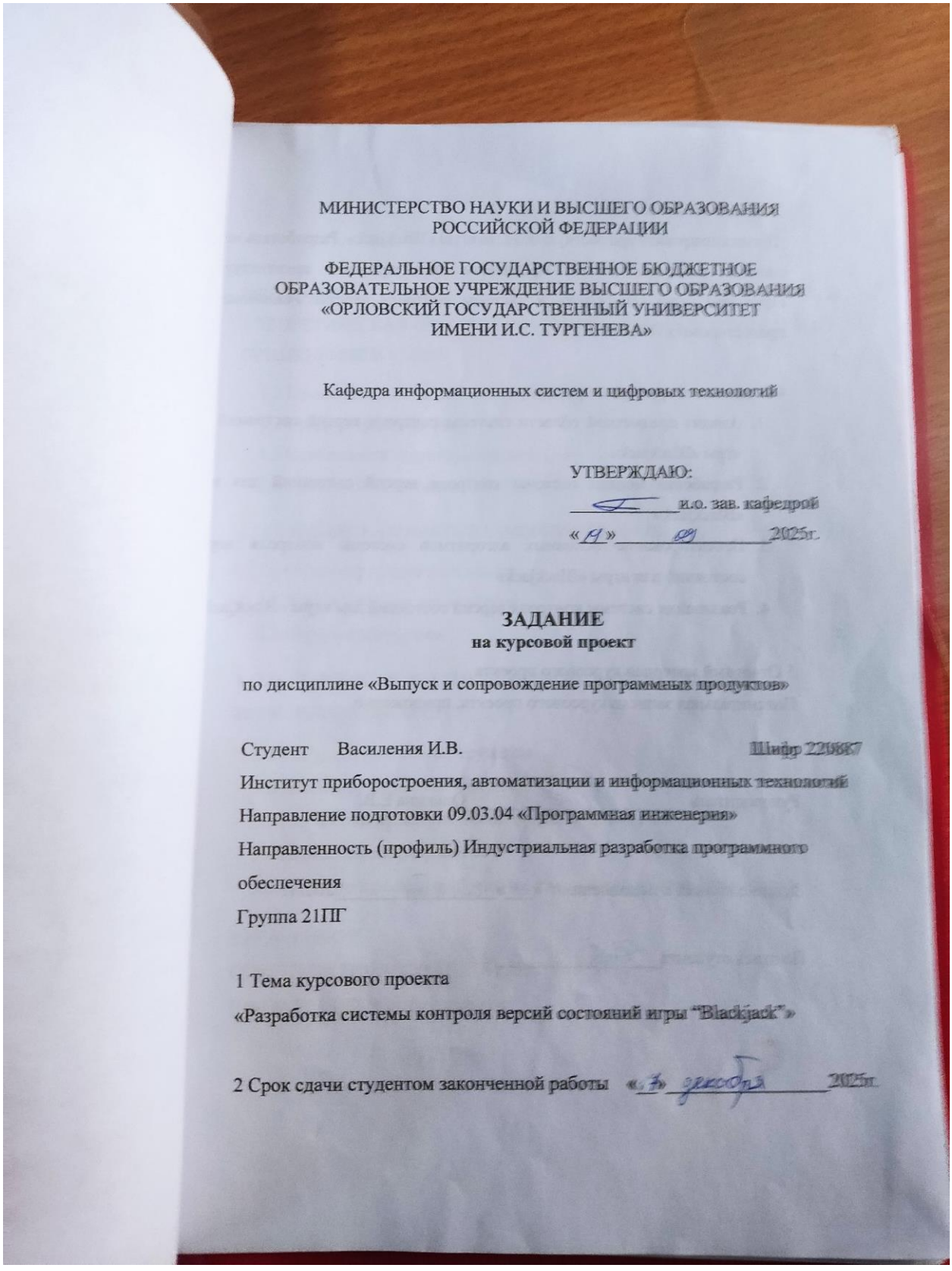
Направленность (профиль) Индустриальное производство программного обеспечения

Группа 21ПГ

Руководитель _____ Олькина Е.В.

Оценка: «хорошо» Дата 24.12.25

Орел 2025



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра информационных систем и цифровых технологий

УТВЕРЖДАЮ:
_____ и.о. зав. кафедрой
«19» 09 2025г.

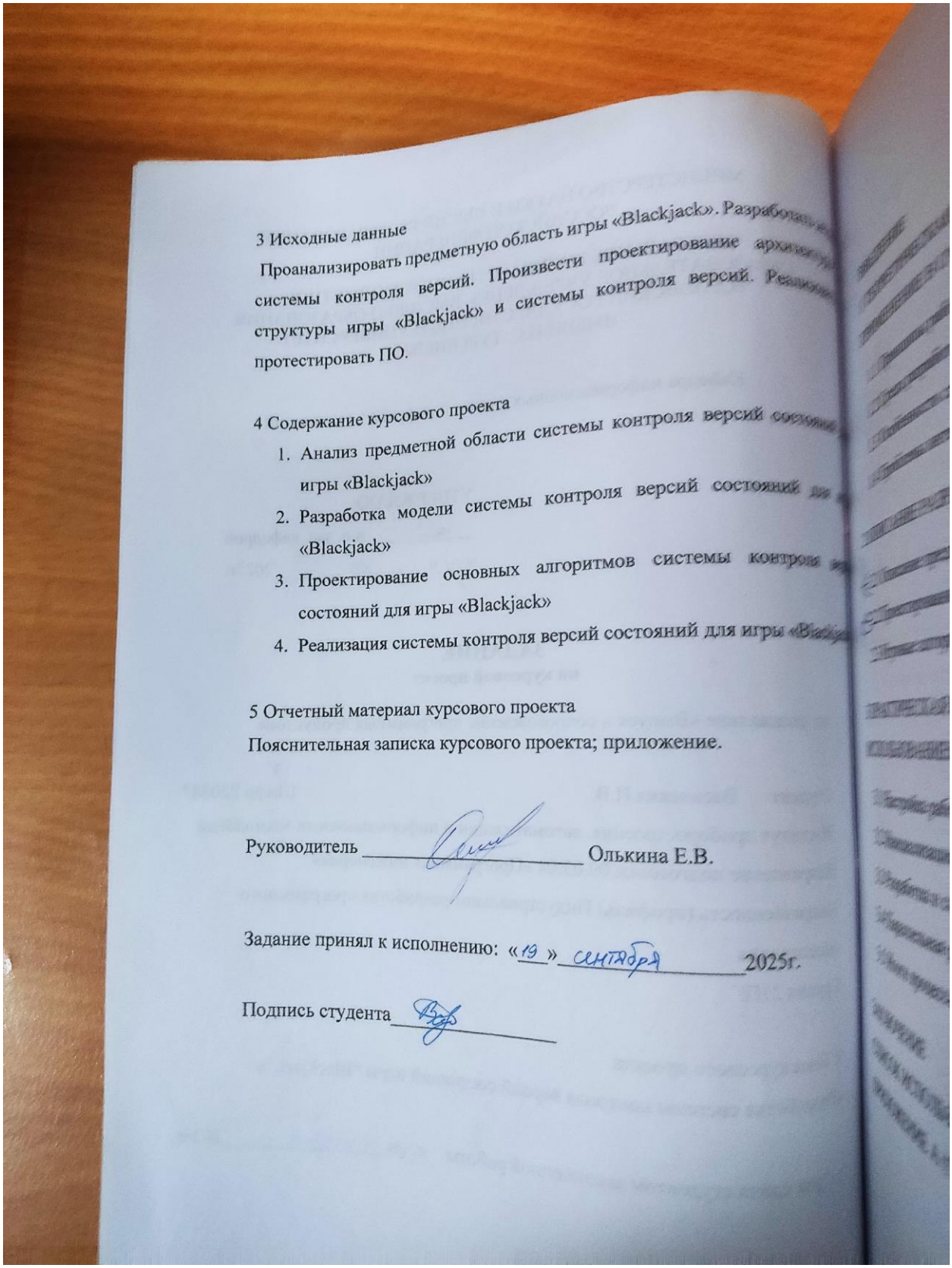
ЗАДАНИЕ
на курсовой проект

по дисциплине «Выпуск и сопровождение программных продуктов»

Студент Василения И.В. Шифр 220887
Институт приборостроения, автоматизации и информационных технологий
Направление подготовки 09.03.04 «Программная инженерия»
Направленность (профиль) Индустриальная разработка программного обеспечения
Группа 21ПГ

1 Тема курсового проекта
«Разработка системы контроля версий состояний игры "Blackjack"»

2 Срок сдачи студентом законченной работы «3» декабря 2025г.



3 Исходные данные

Проанализировать предметную область игры «Blackjack». Разработать и системы контроля версий. Произвести проектирование архитектуры структуры игры «Blackjack» и системы контроля версий. Реализовать протестировать ПО.

4 Содержание курсового проекта

1. Анализ предметной области системы контроля версий состояний игры «Blackjack»
2. Разработка модели системы контроля версий состояний для «Blackjack»
3. Проектирование основных алгоритмов системы контроля версий состояний для игры «Blackjack»
4. Реализация системы контроля версий состояний для игры «Blackjack»

5 Отчетный материал курсового проекта

Пояснительная записка курсового проекта; приложение.

Руководитель _____ Олькина Е.В.

Задание принял к исполнению: «19» сентября 2025г.

Подпись студента _____

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современная разработка программного обеспечения, особенно в сфере игровой индустрии, характеризуется высокой сложностью и необходимостью координации усилий множества специалистов. В таких условиях эффективное управление версиями проекта становится критически важным для обеспечения стабильности, отслеживания изменений и возможности отката к предыдущим состояниям системы. Одним из наиболее мощных и распространенных инструментов для решения этих задач является система контроля версий Git.

Однако использование Git в среде разработки игр, в частности в проектах, создаваемых на платформе Unity, имеет свою специфику. Стандартная конфигурация репозитория не учитывает особенности хранения метаданных активов, сцен и настроек Unity, что может приводить к конфликтам слияния, потере данных и неработоспособности проекта.

Разработка игры Blackjack служит наглядным примером проекта, где такие проблемы могут возникнуть. Этот проект включает в себя исходный код на C#, сцены, префабы, графические и аудио-активы, управление которыми требует четкой стратегии контроля версий.

Целью данной курсовой работы является разработка и внедрение эффективной системы контроля версий на основе Git для проекта игры Blackjack, создаваемого с использованием среды разработки видеоигр Unity.

Для достижения поставленной цели необходимо решить следующие задачи:

- Изучить принципы работы системы контроля версий Git и ее взаимодействие с Unity-проектами.
- Проанализировать типичные проблемы, возникающие при использовании Git в Unity, и существующие лучшие практики их решения.

— Разработать оптимальную конфигурацию Git для проекта Blackjack, включая настройку .gitignore и рабочего процесса для команды разработчиков.
— Внедрить предложенную конфигурацию в учебный проект и проверить ее на практике, оценив эффективность решения.

Актуальность работы обусловлена повсеместным использованием Unity и Git в коммерческой разработке игр. Грамотно настроенная система контроля версий позволяет значительно повысить продуктивность команды, снизить количество ошибок, связанных с версионностью, и обеспечить целостность проекта на всех этапах его жизненного цикла.

1 ТЕОРЕТИЧЕСКАЯ ОСНОВА СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ GIT И ЕЕ ПРИМЕНЕНИЕ В UNITY

1.1 Принципы работы системы контроля версий Git

Система контроля версий представляет собой специализированное программное обеспечение, предназначенное для управления изменениями в документах, исходном коде и других коллекциях информации. Git, созданный Линусом Торвальдсом в 2005 году, является одной из наиболее распространенных распределенных систем контроля версий. Его фундаментальное отличие от централизованных систем заключается в том, что каждый разработчик хранит на своем локальном компьютере полную копию репозитория со всей историей изменений. Такой подход обеспечивает высокую скорость операций, возможность работы в автономном режиме и избыточность данных.

Основополагающим принципом работы Git является хранение данных в виде снимков состояния проекта. В отличие от многих других систем контроля версий, которые хранят изменения как набор различий между файлами, Git сохраняет данные как серию снимков состояния проекта в момент каждого «коммита». Если файл не подвергался изменениям, система не сохраняет его повторно, а создает ссылку на уже существующую версию.

Важнейшей характеристикой Git является наличие трех состояний файлов. Файлы могут находиться в зафиксированном состоянии, когда данные надежно сохранены в локальной базе данных; в измененном состоянии, когда файлы были модифицированы, но еще не зафиксированы; и в индексированном состоянии, когда измененный файл был отмечен для включения в следующий коммит.

Архитектура Git включает три основных секции проекта: рабочую копию, область подготовленных файлов и каталог Git. Такой подход обеспечивает гибкое управление историей изменений. Несмотря на мощь данного инструмента, его эффективное использование в специфических средах, таких как Unity, требует дополнительной настройки и адаптации.

1.2 Среда разработки Unity

Unity — это кроссплатформенная среда разработки интерактивных приложений, в первую очередь видеоигр, созданная компанией Unity Technologies. Первая версия была выпущена в 2005 году, и с тех пор Unity стала одним из наиболее популярных инструментов в индустрии благодаря своей доступности, гибкости и мощной экосистеме.

Архитектура Unity основана на компонентно-ориентированной модели. Основной единицей является GameObject — контейнер, который может содержать множество компонентов. Компоненты определяют поведение и свойства объекта: например, компонент Transform задаёт положение, поворот и масштаб, MeshRenderer отвечает за отображение 3D-модели, а Rigidbody добавляет физическое поведение.

Скриптование в Unity выполняется на языке программирования **C#.** Скрипты также являются компонентами, которые можно прикреплять к игровым объектам. Они реагируют на события жизненного цикла (такие как Start(), Update()), обрабатывают ввод пользователя и управляют игровой логикой. Unity предоставляет богатый API для работы с графикой, физикой, анимацией, звуком и пользовательским интерфейсом (UI).

Ключевые особенности Unity, важные для разработки проекта Blackjack:

- Визуальный редактор сцен: позволяет расставлять объекты, настраивать их свойства и компоненты без написания кода.
- Префабы (Prefabs): система повторно используемых шаблонов игровых объектов, что особенно полезно для создания карт и интерфейсных элементов.
- Система ассетов: все ресурсы (скрипты, текстуры, звуки, модели) хранятся в папке Assets и импортируются с настраиваемыми параметрами.
- Кроссплатформенность: сборка проекта под различные платформы (ПК, мобильные устройства, веб) с минимальными изменениями кода.

— Интеграция с системами контроля версий: Unity генерирует мета-файлы (.meta) для каждого ассета, которые хранят уникальные идентификаторы и настройки импорта.

Визуальный рабочий процесс в Unity строится вокруг понятия сцены (Scene). Каждая сцена представляет собой отдельный уровень или экран игры. В проекте Blackjack используется одна основная сцена, содержащая игровой стол, карты, интерфейс и управляющие элементы.

Для эффективной работы в Unity также важно понимать цикл выполнения скриптов. Основные методы:

— Awake(): вызывается при создании объекта, до Start().
— Start(): вызывается один раз перед первым кадром, если скрипт активен.
— Update(): вызывается каждый кадр, используется для непрерывной логики (например, обработка ввода).
— FixedUpdate(): вызывается с фиксированным интервалом, используется для физических расчётов.

Unity активно используется в индустрии для создания 2D- и 3D-игр, AR/VR-приложений, симуляций и интерактивных визуализаций. В рамках данного проекта Unity была выбрана как оптимальная платформа для реализации карточной игры Blackjack благодаря своей простоте, поддержке 2D-графики и возможности быстрого прототипирования.

1.3 Особенности структуры проекта Unity

Проект Unity характеризуется сложной структурой, которая генерируется и управляется движком. Понимание этой структуры является необходимым условием для правильной настройки системы контроля версий.

Ключевым элементом проекта является папка Assets, содержащая все ресурсы проекта: скрипты, текстуры, модели, звуковые файлы и префабы. Эта папка должна полностью отслеживаться в системе контроля версий.

Папка Library содержит кэшированные и сгенерированные данные на основе ассетов. Данная папка не должна отслеживаться в Git, поскольку она может быть автоматически перегенерирована из папки Assets и файлов проекта. Аналогичным

образом папки Temp и Logs, содержащие временные файлы и логи работы редактора, должны быть исключены из репозитория.

Особого внимания заслуживают мета-файлы с расширением .meta. Каждому ассету в папках Assets и ProjectSettings соответствует мета-файл с одноименным названием. Эти файлы содержат критически важные для Unity данные: импорт-настройки ассета, глобальные уникальные идентификаторы и ссылки между ассетами. Без корректного управления мета-файлами проект становится неработоспособным.

1.4 Проблемы интеграции Git и Unity и пути их решения

Использование Git для проекта Unity без предварительной настройки приводит к возникновению ряда серьезных проблем, требующих системного решения.

Одной из наиболее значительных проблем являются конфликты в бинарных файлах. Такие ассеты как сцены с расширением .unity, префабы с расширением .prefab и некоторые настройки хранятся в бинарном формате. Стандартные инструменты Git не способны автоматически сливать изменения в таких файлах. При одновременном редактировании одного бинарного файла двумя разработчиками возникает конфликт, разрешить который можно только путем ручного выбора одной из версий файла, что неизбежно ведет к потере работы одного из участников команды.

Критической проблемой является отсутствие или некорректное отслеживание мета-файлов. Если мета-файлы не добавлены в репозиторий, Unity будет генерировать новые при загрузке проекта. Это приводит к потере всех ссылок между ассетами, включая связи скриптов с объектами на сцене, что делает проект полностью неработоспособным.

Существенной проблемой становится захламление репозитория автоматически генерируемыми файлами. Добавление в репозиторий папок Library, Temp, Logs, Build и пользовательских настроек из папки UserSettings приводит к резкому увеличению размера репозитория, замедлению операций и возникновению ложных конфликтов.

Решение этих проблем заключается в грамотной начальной настройке репозитория. Ключевым инструментом для этого является файл .gitignore. Специально составленный файл .gitignore для Unity указывает системе, какие файлы и папки следует игнорировать и не добавлять в отслеживание. Это позволяет включить в репозиторий только те элементы, которые необходимы для сборки проекта на другой машине: папку Assets, папку ProjectSettings и все соответствующие им .meta-файлы. Остальные сгенерированные и временные данные должны быть исключены из отслеживания.

Таким образом, процесс настройки системы контроля версий для проекта Unity сводится к корректному конфигурированию существующего инструмента под конкретную среду разработки, что и будет продемонстрировано в рамках проекта игры Blackjack.

2 ОПИСАНИЕ РАЗРАБАТЫВАЕМОЙ ПРОГРАММЫ

2.1 Описание предметной области

Проект Blackjack представляет собой цифровую реализацию классической карточной игры "двадцать одно", также известной как "блэкджек". Понимание правил и ключевых элементов игры является основой для проектирования её архитектуры и выделения независимых модулей.

Основная цель игры для игрока - набрать сумму очков, максимально близкую к 21, но не превышающую её, обыграв тем самым дилера. В начале каждого раунда игрок и дилер получают по две карты. Карты имеют следующие значения: карты от двойки до десятки оцениваются по своему номиналу, лицевые карты (валет, дама, король) - 10 очков, туз может считаться как 1 или 11 очков в зависимости от ситуации. Игрок видит обе свои карты и одну карту дилера.

После получения карт игрок может последовательно брать дополнительные карты ("Hit"), стремясь улучшить свой счет, или остановиться ("Stand"). Если сумма очков игрока превышает 21, он проигрывает ("Bust"). После завершения хода игрока дилер вскрывает свою вторую карту и обязан брать карты, пока сумма его очков не достигнет 17 или более, после чего он останавливается. Результат раунда определяется сравнением итоговых сумм: игрок выигрывает, если его счет больше счета дилера (и не превышает 21), или если дилер превысил 21. При равных суммах объявляется ничья.

Перед началом игры игрок делает ставку. В случае победы он получает обратно свою ставку и выигрыш, равный ставке. В случае победы с комбинацией из лицевой карты и туза игроку возвращается выигрыш в размере 3:2 от ставки. В случае ничьей ставка возвращается без дополнительного выигрыша, а в случае поражения ставка теряется.

Алгоритм проведения раунда игры можно изобразить в виде блок-схемы (Рисунок 2.1.1).

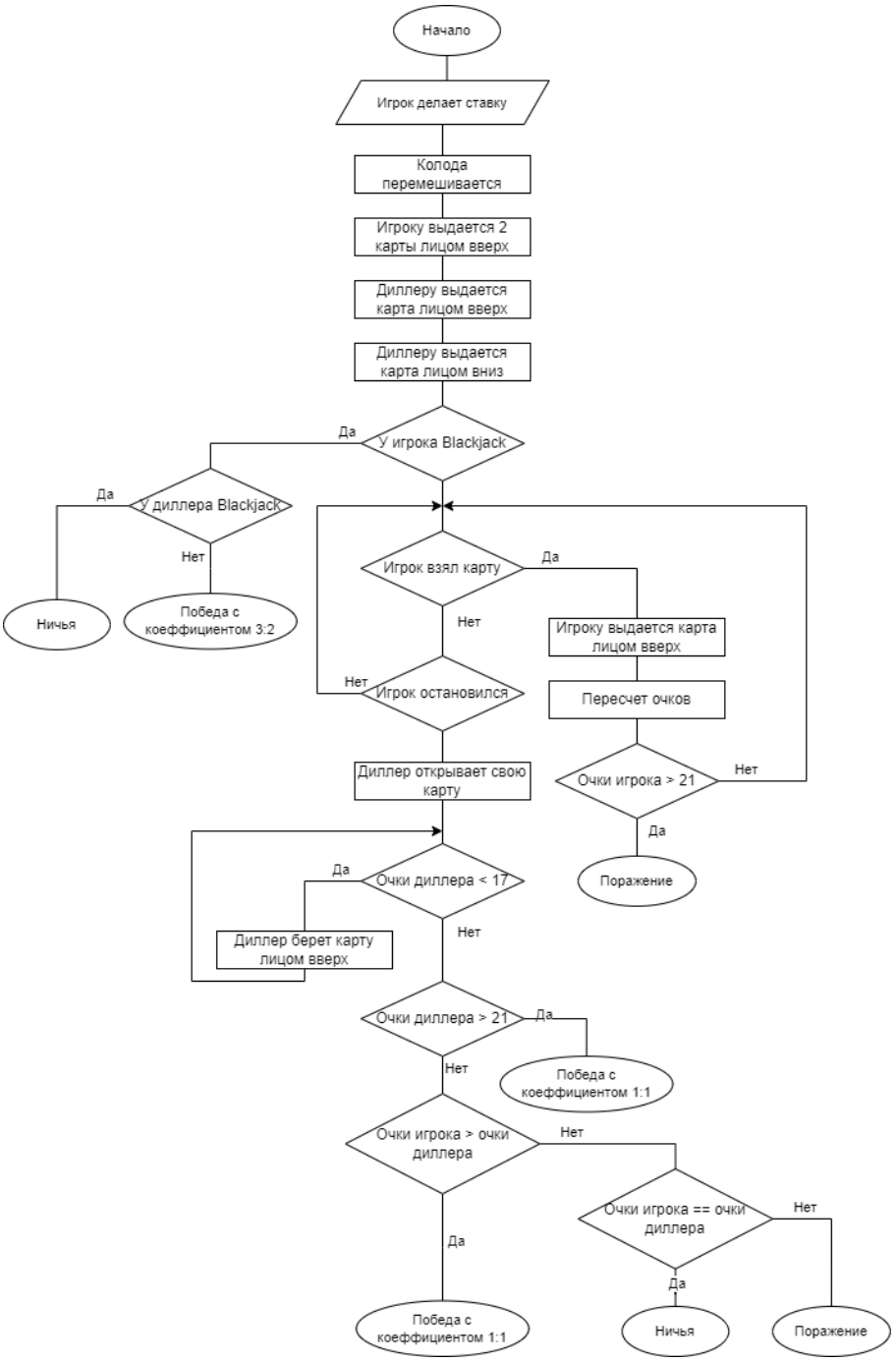


Рисунок 2.1.1 – Алгоритм раунда игры Blackjack

Данный набор правил формирует четкие требования к программной реализации, которая должна включать следующие ключевые компоненты:

— Логика колоды и карт: Механизм создания, перемешивания и выдачи карт из стандартной колоды в 52 карты.

— Игровая механика: Управление потоком раунда, обработка действий игрока и автоматизированные действия дилера, подсчет очков с учетом особого правила для туза.

— Пользовательский интерфейс: Визуальное представление игрового стола, карт, фишек для ставок и элементов управления.
— Система звуков и обратной связи: Звуковые эффекты для ключевых событий.

2.2 Проектирование архитектуры

Проектирование архитектуры игры Blackjack основывается на принципе разделения ответственности, что позволяет выделить четкие и независимые модули. Центральным элементом данных является класс, представляющий игровую карту. Этот класс содержит информацию о масти, достоинстве и базовой стоимости, выступая фундаментальной неделимой единицей, на которой строится вся остальная логика.

Для управления коллекцией карт проектируется класс игровой колоды. Его обязанность включает создание полного набора карт, реализацию алгоритма их случайного перемешивания и предоставление интерфейса для последовательной выдачи карт в процессе игры. Этот класс служит основным источником данных для игрового цикла.

Логика формирования и оценки набора карт у каждого участника инкапсулируется в классе игровой руки. Данный класс отвечает за хранение карт, вычисление итоговой суммы очков с интеллектуальной обработкой особых правил (таких как учет туза как 1 или 11 очков), а также за определение ключевых состояний, например, перебора или комбинации "блэкджек".

Общее управление ходом игры и координация взаимодействия между компонентами возлагается на главный управляющий класс. Он контролирует последовательность этапов раунда: от приема ставки и раздачи карт до обработки действий игрока, автоматизированного хода дилера и финального определения победителя. Этот класс также является центральным узлом для обработки событий, инициируемых пользовательским интерфейсом.

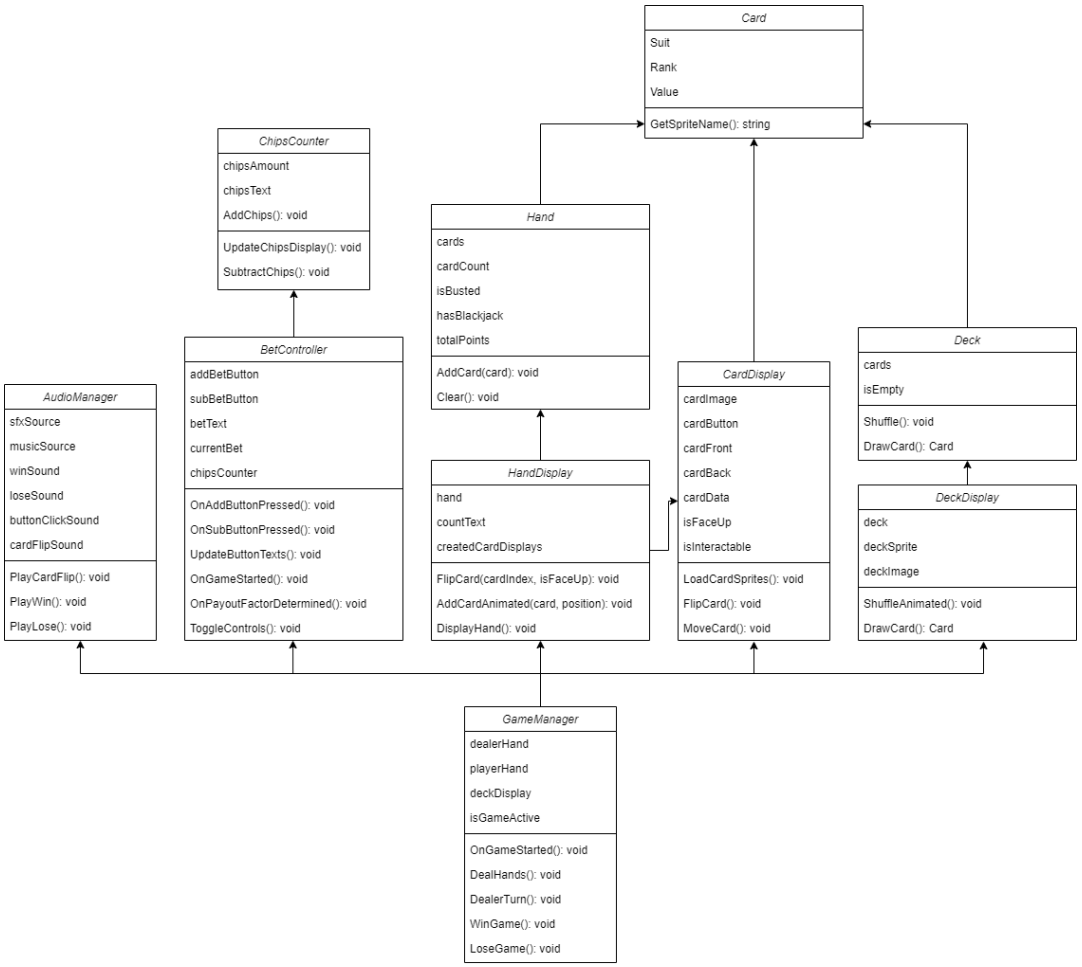


Рисунок 2.2.1 – Архитектура разрабатываемой игры в виде диаграммы классов

Визуальное представление и интерактивность обеспечиваются набором классов контроллеров интерфейса. Эти классы отвечают за отображение игровых сущностей на экране, реализацию анимаций и обработку нажатий на элементы управления, такие как кнопки "Взять карту" или "Сдать". Они действуют как прослойка между данными модели и пользователем.

Для создания звукового сопровождения проектируется отдельный класс-менеджер аудио. Его задача — централизованное управление звуковыми ресурсами и предоставление простого интерфейса для воспроизведения эффектов, соответствующих различным игровым событиям, что значительно повышает качество игрового опыта.

Также реализуются отдельные классы для подсчета очков и применения ставки. Эту функциональность необходимо вынести в отдельные классы, так

как расчёт получаемых при выигрыше фишек может меняться в зависимости от руки, выигравшей раунд. Это значит, что необходимо учитывать контекст раунда. Для этого целесообразно выделить отдельные программные модули.

Такая модульная структура (Рисунок 2.2.1), где каждый класс имеет четко очерченную зону ответственности, не только упрощает разработку и отладку, но и идеально соответствует модели параллельной работы в системе контроля версий Git, позволяя вести разработку каждого модуля в изолированной функциональной ветке.

2.3 Игровые алгоритмы

Основной алгоритм проведения раунда в игре достаточно прост, и в основном состоит из ветвлений в зависимости от очков игрока и дилера. Однако один момент заслуживает пристального внимания.

В карточных играх, и в частности в Blackjack, честная и непредсказуемая случайность распределения карт является фундаментальным требованием. Колода в игре должна перемешиваться случайным образом. При этом должен использоваться алгоритм, имеющий достаточную эффективность и обеспечивающий равномерное распределение всех возможных перестановок карт. Таким алгоритмом можно считать алгоритм Фишера-Йетса (Рисунок 2.3.1).

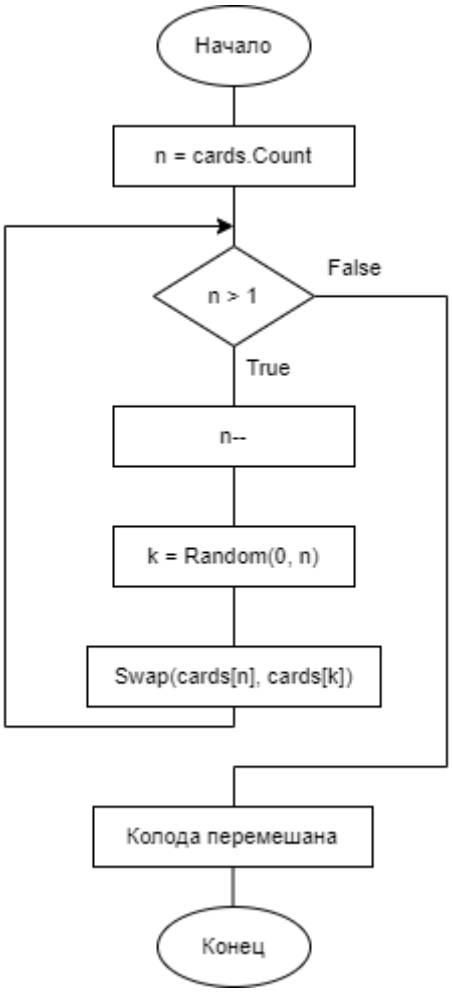


Рисунок 2.3.1 – Алгоритм Фишера-Йетса для перемешивания колоды

С его помощью можно перемешать колоду за линейное время O(n), при этом каждая перестановка карт будет равновероятной.

3 ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ ПРОЕКТА BLACKJACK С ИСПОЛЬЗОВАНИЕМ GIT

3.1 Настройка рабочего окружения

Первым этапом является подготовка рабочего окружения. Для проекта Blackjack используется Unity версии 6000.2.8f1 и Git версии 2.42.0. Инициализация репозитория начинается с создания файла «.gitignore» в корневой директории проекта.

Важным аспектом настройки является использование готовых и проверенных шаблонов файла «.gitignore». Компания GitHub поддерживает официальный репозиторий с шаблонами «.gitignore» для различных технологий и сред разработки, включая специализированный шаблон для Unity. Этот репозиторий содержит накопленный опыт сообщества разработчиков и учитывает особенности работы с движком Unity. Особое внимание уделяется настройке файла «.gitignore», который включает исключение для временных файлов Unity. В исключения попадают папки Library, Temp, Logs, Build, а также файлы пользовательских настроений из папки UserSettings. Важным моментом является сохранение в репозитории папки ProjectSettings, поскольку она содержит критически важные настройки проекта.

Основной вызов при работе с Unity проектами связан с обработкой бинарных файлов. Для решения этой проблемы в проекте Blackjack применяется стратегия блокировки бинарных файлов. Настраивается атрибут файла .gitattributes, который указывает Git рассматривать файлы с расширениями .unity, .prefab, .asset как бинарные.

Для минимизации конфликтов при работе команды устанавливается правило обязательной коммуникации при работе со сценами и префабами. Разработчики заранее оповещают команду о начале работы с конкретными бинарными файлами, что позволяет избежать одновременного редактирования.

Разрабатывается специализированный рабочий процесс, адаптированный под особенности игры Blackjack. Основной веткой разработки является master,

которая содержит только стабильные версии проекта. Для каждой новой функции создается feature-ветка с именем feature-description.

Процесс слияния изменений включает обязательное проведение code review. Каждый разработчик создает merge request после завершения работы над функциональностью. Перед слиянием выполняется проверка на отсутствие конфликтов и корректность meta файлов.

3.2 Инициализация репозитория

С помощью утилиты «curl» получим готовый шаблон файла «.gitignore» и разместим его в папке проекта. Далее создается новый репозиторий командой «git init». Необходимо добавить добавленный файл в репозиторий и зафиксировать изменения (Рисунок 3.2.1).

```
D:\KR_BLACK_JACK\BLACKJACK>curl -o .gitignore https://raw.githubusercontent.com/github/gitignore/main/Unity.gitignore
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2314  100  2314    0     0   2314      0  0:00:01 --:--:--  0:00:01  5108

D:\KR_BLACK_JACK\BLACKJACK>git init
Initialized empty Git repository in D:/KR_BLACK_JACK/BLACKJACK/.git/

D:\KR_BLACK_JACK\BLACKJACK>git add *
warning: in the working copy of '.gitignore', LF will be replaced by CRLF the next time Git touches it

D:\KR_BLACK_JACK\BLACKJACK>git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   .gitignore


D:\KR_BLACK_JACK\BLACKJACK>git commit -m "init"
[master (root-commit) e52d0be] init
 1 file changed, 99 insertions(+)
 create mode 100644 .gitignore

D:\KR_BLACK_JACK\BLACKJACK>
```

Рисунок 3.2.1 – Инициализация репозитория

Далее через менеджер проектов Unity создадим пустой проект, который будет содержать вышеупомянутый репозиторий. Без добавления игнорирующего файла объем изменений в созданном проекте измеряется в десятках тысяч файлов. Однако при добавлении игнорирующего файла количество изменений сокращается до приблизительно 50 самых важных файлов, что подтверждает важность его добавления и дополнительной конфигурации по необходимости. На данном этапе необходимо сделать коммит,

то есть зафиксировать изменения в папке проекта, сохранив его самую начальную версию. Это будет отправной точкой для добавления новых функций.

3.3 Разработка и слияние новой функциональности

В качестве первого добавленного функционального блока рассмотрим создание игрового поля. Создадим для этого новую ветку feature/gamefield, начав ее от последнего коммита в master-ветке. Далее добавим игровое поле, интерфейс и игровые кнопки. Каждый элемент будет фиксироваться отдельным коммитом. По окончанию работы, после ручного тестирования интерфейса, необходимо слить эту feture-ветку обратно в master-ветку, чтобы последующий функционал добавлялся уже для более актуальной стабильной версии игры. Для этого разработчику, как правило являющемуся частью команды из других разработчиков, нужно создать так называемый запрос на слияние (pull request, или merge request). Это возможно сделать с помощью веб-сервисов для хостинга IT проектов, как GitHub, GitLab и т.п. Такие сервисы незаменимы при совместной разработке, так как каждый разработчик может с любого компьютера, воспользовавшись специальными приложениями или командами утилиты git, получить актуальную версию проекта и продолжить разработку.

Создадим запрос на слияние новой feature-ветки с master-веткой (Рисунок 3.3.1). После одобрения руководителя процесса разработки этот запрос будет принят, и ветка master будет продолжена коммитами из feature-ветки.

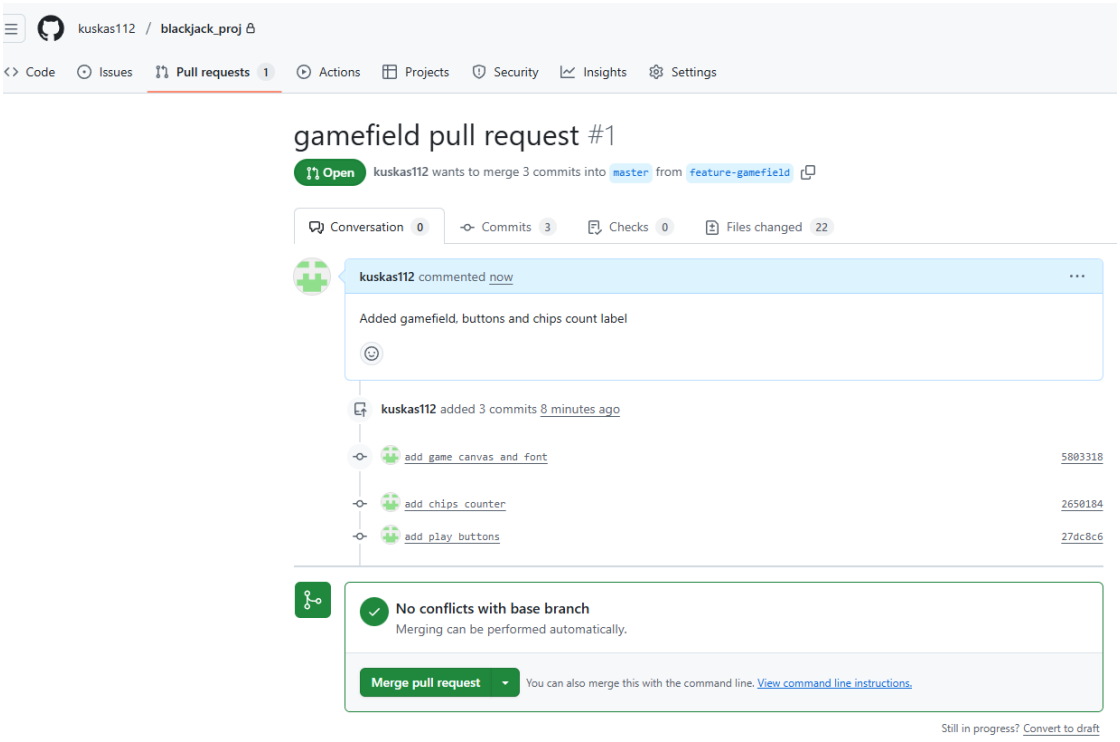


Рисунок 3.3.1 – Страница запроса на слияние в хостинге GitHub

3.4 Параллельная разработка

Одним из ключевых преимуществ распределённой системы контроля версий является возможность одновременной работы нескольких разработчиков над разными частями проекта без риска конфликтов и потери данных.

Для обеспечения эффективной совместной работы над проектом Blackjack применяется стратегия параллельной разработки на основе функциональных веток. Основная ветка master содержит только проверенный и стабильный код, что позволяет в любой момент иметь работоспособную версию проекта.

Параллельная разработка организуется следующим образом: при поступлении новой задачи, такой как реализация системы звуков или создание ядра игровой логики, от ветки master создаётся отдельная функциональная ветка с именем, отражающим суть задачи, например, feature-audio-management или feature-deck-logic. Каждый разработчик или группа разработчиков работают строго в своей выделенной ветке, внося изменения, фиксируя промежуточные результаты и тестируя функциональность в изоляции от основной кодовой базы.

Важным элементом процесса является поддержание актуальности функциональных веток. Периодически, а также перед созданием запроса на

слияние, ветка должна обновляться из актуальной ветки master для интеграции изменений, внесённых другими участниками команды. Это позволяет своевременно обнаруживать и разрешать потенциальные конфликты на раннем этапе.

Когда работа над функциональностью в ветке завершена и протестирована, создаётся запрос на слияние. Запрос на слияние служит центральной точкой для обсуждения кода, проведения ревью и автоматизированных проверок. Участники команды могут оставлять комментарии, предлагать улучшения и утверждать изменения перед их интеграцией.

После прохождения ревью и всех проверок запрос на слияние утверждается и изменения из функциональной ветки переносятся в master (Рисунок 3.4.1). Этот последовательный и контролируемый процесс позволяет нескольким разработчикам или командам работать над разными частями игры одновременно, минимизируя конфликты и обеспечивая стабильное состояние основного репозитория на всех этапах разработки проекта Blackjack.

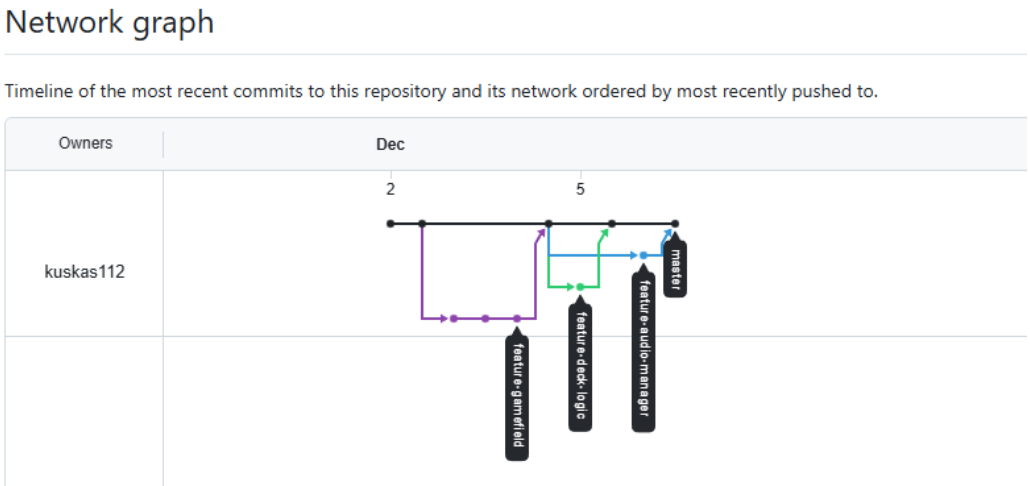


Рисунок 3.4.1 – Визуализация master-ветки со слияниями feature-веток

3.5 Итоги процесса разработки и финальное состояние репозитория

В результате практической реализации системы контроля версий для проекта Blackjack удалось достичь поставленных целей и подтвердить эффективность предложенного подхода. Разработка велась в соответствии с описанным рабочим процессом, что позволило обеспечить стабильность основного кода при параллельной работе над несколькими функциональными модулями.

За время разработки было создано шесть функциональных веток (Рисунок 3.5.1), каждая из которых соответствовала отдельному модулю игры: от базовой сцены и системы карт до звукового сопровождения, и финальной полировки. Все ветки успешно интегрированы в основную ветку через механизм merge request с обязательным code review. Это позволило не только избежать серьезных конфликтов при слиянии, но и поддерживать высокое качество кода за счет коллективной проверки изменений.

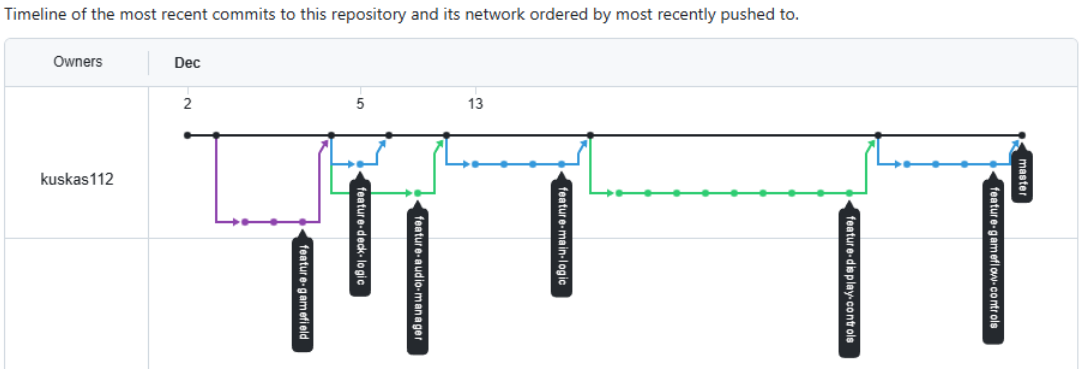


Рисунок 3.5.1 – Финальное состояние репозитория

Финальная структура репозитория отражает модульный принцип построения проекта. В папке Assets организованы отдельные директории для скриптов, сцен, префабов, графических ресурсов и аудиофайлов. Особое внимание уделено корректному хранению метаданных Unity — все .meta файлы сохранены и отслеживаются в репозитории, что гарантирует целостность проекта при переносе на другие рабочие станции.

Применение заранее настроенного файла .gitignore показало свою необходимость: без него репозиторий содержал бы десятки тысяч временных

и сгенерированных файлов, что увеличило бы его размер более чем в 30 раз. Благодаря корректной конфигурации в системе контроля версий хранятся только те файлы, которые необходимы для сборки проекта — папки Assets и ProjectSettings со всеми сопутствующими метафайлами.

История коммитов демонстрирует четкую ветвящуюся структуру, где каждая функциональность разрабатывалась изолированно, а затем вливалась в основную ветку. Такой подход позволил вести разработку одновременно несколькими участниками без риска нарушения работоспособности основной версии проекта. Визуально история представляет собой дерево с короткоживущими функциональными ветками, регулярно синхронизируемыми с main, что минимизировало расхождения и упростило финальные слияния.

Проведенная работа подтвердила, что использование Git в Unity-проектах требует не только понимания принципов работы системы контроля версий, но и учета специфики игрового движка. Правильно настроенный репозиторий становится надежным фундаментом для командной разработки, обеспечивая как техническую стабильность проекта, так и эффективное взаимодействие между разработчиками.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была успешно разработана и внедрена эффективная система контроля версий на основе Git для проекта игры Blackjack в среде Unity. Поставленные задачи были решены в полном объеме.

Были изучены ключевые принципы работы Git и специфика структуры Unity-проектов. Проанализированы основные проблемы их совместного использования, такие как конфликты в бинарных файлах и управление метаданными, и предложены практические решения на основе корректной настройки файла .gitignore.

Практическая реализация на примере Blackjack продемонстрировала полный цикл работы: от инициализации репозитория до организации параллельной разработки через функциональные ветки и контролируемого слияния изменений. Примененный рабочий процесс доказал свою эффективность, позволив вести разработку независимых модулей одновременно при сохранении стабильности основного кода.

Таким образом, работа подтвердила, что грамотная настройка и использование Git в Unity-проектах являются критически важными для обеспечения продуктивности команды, целостности проекта и успешной совместной разработки.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Git – Официальная документация | git-scm.com [Электронный ресурс]. Режим доступа: https://git-scm.com/doc (Дата обращения: 01.12.2025)

2. Unity – Работа с системой контроля версий | wikipedia.org [Электронный ресурс]. Режим доступа: https://docs.unity3d.com/Manual/VersionControl.html (Дата обращения: 01.12.2025)

3. GitHub – Gitignore templates for Unity | github.com [Электронный ресурс]. Режим доступа: https://github.com/github/gitignore/blob/main/Unity.gitignore (Дата обращения: 01.12.2025)

4. Правила работы с Git в команде: лучшие практики | habr.com [Электронный ресурс]. Режим доступа: https://habr.com/ru/companies/mailru/articles/550592/ (Дата обращения: 01.12.2025)

ПРИЛОЖЕНИЕ А (Обязательное)

Листинг фрагмента исходного кода программы

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

public class Hand
{
    public List<Card> cards;
    public int cardCount => cards.Count;
    public bool isBusted => totalPoints > 21;
    public bool hasBlackjack => cardCount == 2 && totalPoints == 21;

    public Hand()
    {
        cards = new List<Card>();
    }

    public void AddCard(Card card)
    {
        if (card != null)
        {
            cards.Add(card);
        }
    }

    public int totalPoints
    {
        get
        {
            int total = 0;
            int aceCount = 0;

            foreach (Card card in cards)
            {
                if (card.rank == Card.Rank.Ace)
                {
                    aceCount++;
                    total += 11;
                }
                else
                {
                    total += card.Value;
                }
            }

            while (total > 21 && aceCount > 0)
            {
                total -= 10;
                aceCount--;
            }

            return total;
        }
    }

    public List<int> GetPossiblePoints()
    {
        List<int> points = new List<int> { 0 };

        foreach (Card card in cards)
        {
            List<int> newPoints = new List<int>();
```

```
            foreach (int point in points)
            {
                if (card.rank == Card.Rank.Ace)
                {
                    newPoints.Add(point + 1);
                    newPoints.Add(point + 11);
                }
                else
                {
                    newPoints.Add(point + card.Value);
                }
            }

            points = newPoints;
        }

        points.Sort();
        return points;
    }

    public void Clear()
    {
        cards.Clear();
    }

    public override string ToString()
    {
        string handString = "Карты в руке: ";
        foreach (Card card in cards)
        {
            handString += $"{card.rank} of {card.suit}, ";
        }
        handString += $"Очки: {totalPoints}";

        if (hasBlackjack) handString += " (BLACKJACK!)";
        else if (isBusted) handString += " (ПЕРЕБОР!)";

        return handString;
    }

    public Card GetFirstCard()
    {
        if (cards.Count > 0)
            return cards[0];
        return null;
    }

    public List<Card> GetCardsExceptFirst()
    {
        if (cards.Count <= 1)
            return new List<Card>();

        return cards.GetRange(1, cards.Count - 1);
    }
}
```