ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ИРКУТСКОЙ ОБЛАСТИ
«АНГАРСКИЙ ТЕХНИКУМ СТРОИТЕЛЬНЫХ ТЕХНОЛОГИЙ»

Курсовая работа

Тема: Разработка текстовой квестовой игры

Разработал _____Ковалев Н.К.

Руководитель _____Денисюк А.В.

Ангарск, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В условиях стремительного развития индустрии компьютерных игр, когда на рынке доминируют высокобюджетные проекты с сложной графикой и масштабными мирами, текстовые квесты сохраняют свою популярность благодаря уникальному подходу к повествованию и вовлечению игрока. Эти игры строятся вокруг сюжетов, где основное внимание уделяется выбору игрока и последствиям принятых решений. Простота визуальной части компенсируется глубиной проработки мира и персонажей, что делает текстовые квесты востребованными среди широкой аудитории, ценящей атмосферные истории и нелинейные нарративы.

Одним из факторов актуальности разработки текстовых игр является растущий интерес к инди-проектам и возможность создания полноценных игровых продуктов небольшими командами или даже одним разработчиком. Использование специализированных инструментов, таких как Twine, Ink, Ren'Py или даже стандартные языки программирования, делает процесс доступным для широкой аудитории начинающих гейм-дизайнеров и сценаристов. Создание текстового квеста требует не только умения работать с техническими средствами, но и развития навыков сценарного мастерства, проектирования интерактивных историй и понимания особенностей пользовательского опыта.

Разработка текстового квеста представляет собой комплексный процесс, включающий несколько ключевых этапов. На первом этапе осуществляется разработка общей концепции игры: определение тематики, целевой аудитории, стилистики повествования и механик выбора. Следующим важным шагом является создание сценария, который включает в себя проработку основных квестов. Далее происходит техническая реализация проекта: разработка системы логических переходов между сценами, настройка интерфейса, организация баз данных для хранения

состояний игры. Завершающим этапом становится тестирование, направленное на выявление логических ошибок в структуре сюжета и улучшение пользовательского опыта.

Цель данной работы заключается в разработке оригинальной текстовой квестовой игры, в которой будет реализована система повествования с возможностью выбора, влияющего на развитие сюжета. Проект должен включать в себя тщательно проработанных персонажей, систему последствий решений и обеспечивать высокую степень вовлечённости игрока в происходящее. Предполагается, что итоговая игра сможет продемонстрировать особенности жанра текстовых квестов и подчеркнуть их потенциал как эффективного средства взаимодействия с аудиторией через повествование.

Задачи работы:

1. Изучить особенности жанра текстовых квестов и проанализировать существующие примеры подобных игр.
2. Определить наиболее подходящие технологии и инструменты для создания текстового квеста.
3. Разработать структуру сюжетных линий и систему принятия решений.
4. Реализовать основные элементы игры: сценарий, механизмы выбора, систему переходов между сценами.
5. Провести тестирование игры, выявить логические ошибки и доработать сюжетные ветви при необходимости.

Практическая значимость проекта заключается в приобретении опыта полного цикла разработки текстовой игры — от проектирования сюжета до реализации и тестирования. Созданная игра может быть использована в качестве проекта при трудоустройстве или опубликована на любом сайте для публикаций инди-игр.

I. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1.1 Анализ жанра текстовых игр

1.1.1 Особенности жанра интерактивной литературы

Текстовые квесты, также известные как интерактивная литература, представляют собой особую разновидность цифровых игр, где всё взаимодействие между игроком и игровым миром осуществляется посредством текста. В отличие от традиционных видеоигр, где основную роль играют визуальные и звуковые элементы, текстовые квесты фокусируются на повествовании и вовлечении пользователя через тщательно написанные описания событий, диалоги и варианты выбора. Игрок не просто наблюдает за происходящим, а активно участвует в формировании истории, принимая решения, влияющие на её дальнейшее развитие. При этом система выбора может быть как ограниченной (выбор из нескольких заранее заданных опций), так и более свободной, с возможностью текстового ввода команд.

Одной из ключевых особенностей текстовых квестов является богатое ветвление сюжета. Разработчики стремятся создать множество альтернативных путей и концовок, чтобы игрок чувствовал реальное влияние своих действий на ход повествования. Это, в свою очередь, требует от авторов глубокого сценарного планирования, продуманной логики событий и создания запоминающихся персонажей. Поскольку визуальные образы в таких играх практически отсутствуют, особое значение приобретает качество текста: стилистика, выразительность описаний, убедительность диалогов и грамотная работа с ритмом подачи информации.

Ещё одной характерной чертой текстовых квестов является их сравнительная технологическая доступность. Благодаря минимальным требованиям к ресурсам компьютера и простоте базовой реализации

(особенно в консольных приложениях), жанр открыт для инди-разработчиков и начинающих программистов. Это позволяет использовать текстовые квесты как идеальную платформу для первых проектов в области создания игр. В то же время высокая планка по части литературного мастерства делает разработку таких игр серьёзным вызовом для сценаристов: недостаточная проработка текста или слабая мотивация выбора может привести к тому, что игрок быстро потеряет интерес.

Текстовые квесты могут быть различными по степени интерактивности: от строго линейных историй с минимальными развилками до сложных симуляторов с десятками параметров, влияющих на сюжет. Некоторые проекты включают элементы генерации событий в реальном времени, добавляя элемент неожиданности и многократной реиграбельности. Другие делают акцент на эмоциональной вовлечённости игрока, поднимая важные социальные или философские темы. В зависимости от задач проекта текстовый квест может стремиться либо к максимальной свободе выбора, либо, наоборот, к жёсткому контролю повествования для достижения авторского замысла.

Таким образом, текстовый квест представляет собой уникальный жанр, сочетающий в себе элементы литературы, интерактивности и игрового дизайна. Его успешная разработка требует сочетания навыков программирования, сторителлинга и глубокого понимания психологии восприятия текста. Разработка таких проектов даёт уникальную возможность экспериментировать с повествовательными структурами, исследовать механизмы вовлечения пользователя и создавать необычные игровые опыты при сравнительно небольших технических затратах.

1.1.2 Историческое развитие текстовых квестов

Текстовые приключения, или интерактивные книги, начали своё развитие в середине XX века, когда компьютерные технологии только начинали формироваться. Одной из первых известных игр подобного рода считается "Adventure", созданная Уиллом Краутером в 1975 году.

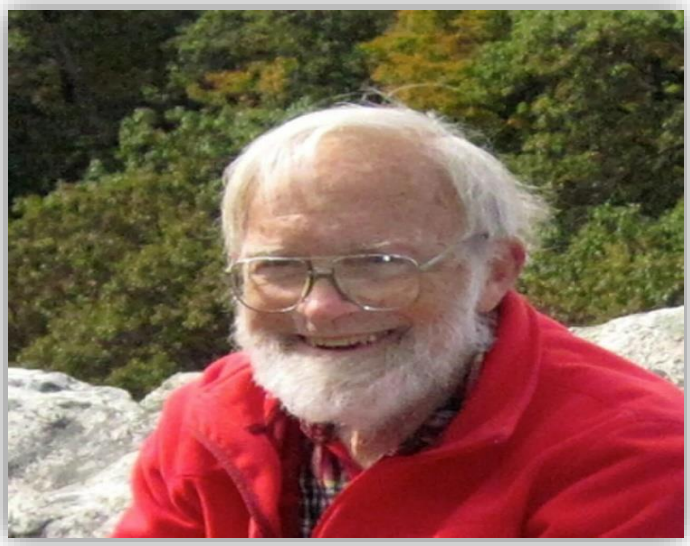


Рис.1 Уильям Краутер

Изначально проект задумывался как средство симуляции спелеологических экспедиций, однако вскоре перерастает в полноценное приключение с элементами фэнтези. Эта игра положила начало целому жанру, где основной формой взаимодействия пользователя с системой являлся текстовый ввод команд и получение текстовых описаний последствий.

В последующие годы текстовые приключения стали активно развиваться благодаря увеличению вычислительных мощностей компьютеров и распространению персональных ЭВМ. В 1980-х годах компания Infocom стала одним из пионеров индустрии, выпустив целую серию культовых игр, среди которых особое место занимает "Zork". Эти проекты отличались высоким качеством написания текстов, сложными

сюжетными развилками и вниманием к деталям игрового мира, что позволило им завоевать популярность среди широкой аудитории.

С развитием графики в компьютерных играх текстовые квесты постепенно утратили лидирующие позиции на рынке. Однако интерес к ним никогда полностью не исчезал. В 1990-х и начале 2000-х годов наблюдается всплеск популярности визуальных новелл в Японии — игр, которые представляли собой эволюцию текстовых квестов, сочетая текст с иллюстрациями и музыкальным сопровождением. Проекты вроде "Clannad" или "Steins;Gate" доказали, что текстовая форма взаимодействия по-прежнему может быть невероятно эффективной в создании эмоциональных и глубоких историй.

Современный этап развития текстовых приключений связан с ростом инди-сцены. Благодаря доступным инструментам разработки и платформам вроде Steam, itch.io или мобильных магазинов приложений, текстовые квесты получили вторую жизнь. Проекты вроде "80 Days", "Choice of Robots" и целая серия от Choice of Games продемонстрировали, что качественная интерактивная литература по-прежнему востребована. Сегодня текстовые квесты активно используют возможности искусственного интеллекта, процедурной генерации и нелинейного повествования, расширяя границы жанра и предлагая новые способы взаимодействия с игроком.

Таким образом, история текстовых приключений представляет собой постоянный процесс адаптации и переосмысления жанра в ответ на изменения технологической базы и вкусов аудитории. Из скромных экспериментов энтузиастов текстовые квесты превратились в полноценную форму цифрового искусства, сочетающую литературные и игровые традиции.

1.1.3. Основные принципы построения сюжета в текстовых квестах

Создание сюжета для текстового квеста требует особого подхода, отличающегося от традиционных методов написания литературы. Поскольку в текстовых играх игроку предоставляется возможность влиять на ход событий, важно продумывать не только основную сюжетную линию, но и многочисленные ответвления, каждое из которых должно быть логически обоснованным и интересным.

Одним из базовых принципов построения сюжета является четкая разработка сценарной карты. Она представляет собой схему всех возможных развилок и их последствий. На этапе проектирования важно определить ключевые моменты истории, так называемые "узлы", в которых игрок принимает важные решения. Каждый выбор должен иметь реальные последствия, чтобы поддерживать у пользователя ощущение значимости собственных действий.

Неотъемлемой частью успешного сюжета является баланс между свободой игрока и авторским контролем над историей. С одной стороны, чрезмерная свобода может привести к расфокусировке и потере интереса. С другой стороны, излишне линейное повествование нивелирует саму суть интерактивности. Оптимальной стратегией является создание структуры с множественными ветвлениями, которые периодически сходятся в общие сюжетные арки. Это позволяет одновременно сохранить контроль над развитием истории и дать игроку ощущение свободы выбора.

Большое значение имеет проработка персонажей и мира. Персонажи должны быть живыми, обладающими собственной мотивацией и характером. Мир должен быть логичным, самодостаточным, с внутренними правилами, понятными игроку. Детали, такие как особенности локаций, традиции общества, история мира, помогают усилить эффект погружения.

Особую роль играет язык повествования. Поскольку текст является основным каналом передачи информации, важно поддерживать грамотный, выразительный стиль, способный вызывать у игрока эмоциональный отклик. Использование различных литературных приёмов — описательных эпитетов, метафор, диалогов, внутренней речи персонажей — позволяет сделать текст насыщенным и живым.

Таким образом, построение сюжета текстового квеста требует сочетания сценарного мастерства, системного мышления и литературного таланта. Только гармоничное взаимодействие всех этих компонентов позволяет создать захватывающую, глубоко вовлекающую историю.

1.2. Особенности проектирования игр на языке Java

Java — это универсальный высокоуровневый язык программирования, обладающий множеством особенностей, делающих его удобным инструментом для разработки текстовых квестов. Главным преимуществом Java является его концепция платформенной независимости: программы, написанные на этом языке, могут выполняться на любых устройствах, где установлен Java Virtual Machine (JVM). Это свойство позволяет создавать игры, которые одинаково хорошо работают на Windows, Linux, macOS и мобильных операционных системах без необходимости их переработки под каждую платформу.

Процесс проектирования игры на Java начинается с выбора архитектурной модели. В большинстве случаев используется объектно-ориентированный подход, поскольку он обеспечивает удобную организацию кода и облегчает сопровождение проекта. Каждая важная сущность текстового квеста — персонажи, предметы, локации, квесты — реализуется в виде отдельных классов с чётко определёнными свойствами и методами. Такой подход позволяет легко расширять функциональность проекта,

добавляя новые элементы без необходимости переписывать уже существующую базу кода.

Особое внимание при разработке текстовой игры на Java уделяется обработке пользовательского ввода. Так как игрок взаимодействует с игрой через текстовые команды, необходимо реализовать эффективный парсер, способный распознавать разнообразные формы ввода и корректно интерпретировать их. Это может включать как простую обработку фиксированных команд (например, "взять меч", "поговорить с торговцем"), так и более сложные системы, позволяющие анализировать грамматическую структуру предложений. При этом необходимо учитывать обработку ошибок: игра должна уметь корректно реагировать на некорректные команды, предоставляя пользователю подсказки, а не просто выводя сообщения об ошибке.

Кроме того, важной особенностью проектирования является реализация системы сохранения и загрузки состояния игры. В Java это может быть реализовано с помощью сериализации объектов, записи состояния в текстовые или бинарные файлы, а также через базы данных. Корректная работа механизмов сохранения обеспечивает удобство для пользователя и повышает общую надёжность приложения.

Эффективное использование стандартных библиотек Java также играет важную роль в разработке текстового квеста. Коллекции (например, ArrayList, HashMap) позволяют удобно управлять списками предметов, квестов, NPC. Работа с потоками ввода-вывода (java.io, java.nio) обеспечивает гибкость при обработке текстовых данных. Система исключений (try-catch-finally) позволяет организовать надёжную обработку непредвиденных ошибок без краха программы.

Проектирование на Java требует также внимания к вопросам производительности и масштабируемости. Хотя текстовые квесты сравнительно нетребовательны к ресурсам, неэффективная архитектура

может привести к замедлению работы на слабых устройствах. Поэтому важно использовать эффективные алгоритмы поиска и обработки данных, избегать избыточного потребления памяти и продумывать структуру классов таким образом, чтобы минимизировать дублирование кода и упростить его сопровождение.

Таким образом, разработка текстового квеста на Java представляет собой сочетание грамотного применения объектно-ориентированных принципов, эффективной работы с текстом и продуманного управления состоянием игры. Этот процесс требует внимательности к деталям и системного подхода к организации проекта.

1.3. Инструменты и технологии, применяемые в разработке

Для успешной реализации текстового квеста на Java необходимо правильно подобрать инструменты разработки, которые обеспечат эффективную работу над проектом на всех этапах — от написания кода до тестирования и отладки. На современном этапе существует широкий спектр инструментов, доступных как для профессиональных разработчиков, так и для студентов и инди-команд.

В первую очередь, основным инструментом является интегрированная среда разработки (IDE). Для Java одной из наиболее популярных IDE считается IntelliJ IDEA, предоставляющая широкий спектр возможностей: интеллектуальную подсказку кода (IntelliSense), удобную навигацию по проекту, встроенные средства рефакторинга и проверки качества кода. Другими популярными вариантами являются Eclipse и NetBeans. Использование мощной IDE существенно ускоряет процесс разработки, позволяет быстрее выявлять ошибки и поддерживать высокое качество проекта.

Для управления зависимостями и сборкой проекта часто применяются системы автоматизации, такие как Maven или Gradle. Хотя текстовый квест

может обходиться без сторонних библиотек, использование Maven или Gradle обеспечивает упрощённое подключение библиотек, настройку проекта и возможность удобного развертывания. Кроме того, применение систем сборки упрощает процесс тестирования и интеграции различных компонентов игры.

В процессе разработки текстового квеста важно также использовать системы контроля версий, такие как Git. С помощью Git можно отслеживать изменения в проекте, работать над разными частями игры параллельно, откатывать изменения при необходимости и организовывать командную работу. Платформы вроде GitHub или GitLab позволяют удобно хранить репозитории и организовывать коллективную разработку, даже если проект разрабатывается одним человеком.

Дополнительно, для тестирования текстовой части и качества взаимодействия можно использовать библиотеки для юнит-тестирования, такие как JUnit. Создание тестов для ключевых компонентов игры, например системы парсинга команд или механики квестов, позволяет заранее выявлять ошибки и повышать стабильность проекта.

При разработке текстового квеста иногда возникает необходимость в генерации случайных событий или динамических описаний. В этом случае полезно использовать встроенные возможности Java для работы с генераторами случайных чисел (java.util.Random) или сторонние библиотеки для процедурной генерации контента.

На этапе тестирования пользовательского опыта можно применять простейшие системы логирования, такие как java.util.logging, или сторонние решения вроде Log4j, чтобы анализировать поведение игроков, выявлять часто встречающиеся ошибки и улучшать игровой процесс.

Таким образом, использование современных инструментов и технологий позволяет значительно упростить и ускорить процесс создания

текстового квеста на Java, повысить его надёжность и обеспечить высокое качество итогового продукта.

1.4. Принципы построения логики текстового квеста

Построение внутренней логики текстового квеста является одной из важнейших задач при разработке игры, так как именно логика определяет, насколько целостным, живым и правдоподобным будет восприниматься игровой мир. От качества реализации игровых механик напрямую зависит глубина вовлечения игрока в происходящее, а значит, и успех проекта в целом.

Первым этапом при проектировании логики квеста является определение основных сущностей и их взаимосвязей. В контексте текстового квеста сущностями обычно выступают персонажи, предметы, локации, задания (квесты) и состояния игрока. Важно не только выделить эти элементы, но и установить между ними чёткие связи: какие предметы можно подобрать, с какими персонажами можно взаимодействовать, какие события могут происходить в различных локациях. В рамках объектно-ориентированного подхода в Java для каждой сущности создаются отдельные классы, а их взаимодействие строится через методы и атрибуты объектов.

Следующим важным аспектом является реализация механизма обработки команд пользователя. Игрок должен иметь возможность давать различные команды — например, "осмотреть комнату", "подобрать предмет", "поговорить с персонажем" или "использовать ключ на двери". Для обработки таких команд требуется создать парсер, который будет анализировать вводимый текст, выделять ключевые слова и определять, какое действие необходимо выполнить. Простейший вариант парсера — это использование операторов if-else или switch-case, однако в более сложных играх целесообразно строить дерево разбора команд или применять регулярные выражения для большей гибкости.

Особое внимание уделяется системе состояний игры. Состояние включает в себя всю текущую информацию о ходе игры: расположение игрока, инвентарь, пройденные квесты, взаимодействия с NPC и изменения в игровом мире. Для корректной работы логики необходимо предусмотреть механизмы сохранения и загрузки состояния. Это может быть реализовано с использованием сериализации объектов, сохранения информации в JSON/XML форматах или применения баз данных для более крупных проектов.

Отдельную роль играет сценарная логика квестов. Квесты в текстовых играх — это не только задачи, поставленные перед игроком, но и важные элементы повествования, через которые развивается сюжет. Каждый квест должен содержать чёткие условия начала, этапы выполнения и критерии завершения. Например, чтобы завершить задание, игрок должен получить определённый предмет или выполнить цепочку действий в правильной последовательности. Для реализации этой логики в Java используются структуры данных вроде списков (List), ассоциативных массивов (Map) и деревьев (Tree), что позволяет гибко управлять зависимостями между заданиями.

Также важно реализовать обработку случайных событий или вариативных исходов. Для создания ощущения динамичности мира в игру можно встроить элементы случайности: шанс на успешный исход переговоров, вероятность найти редкий предмет или случайное изменение погодных условий. Такие элементы легко реализуются с использованием стандартных библиотек Java для работы со случайными числами (java.util.Random).

Не менее важным аспектом является создание системы обратной связи. Игра должна адекватно реагировать на все действия пользователя, предоставляя текстовые описания результатов и последствий его поступков. Каждое действие должно вызывать логичную и интересную реакцию со

стороны игрового мира, будь то открытие нового пути, изменение отношения персонажей или появление новых возможностей.

Таким образом, построение логики текстового квеста требует тщательной архитектурной проработки всех сущностей, их взаимодействий и возможных состояний. Четкая организация кода, применение эффективных алгоритмов обработки текста и правильное управление состоянием игры позволяют создать увлекательный, гибкий и логически целостный игровой мир.

1.5. Основные сложности при разработке текстового квеста

Создание текстового квеста, несмотря на внешнюю простоту жанра, сопряжено с рядом серьёзных трудностей, которые необходимо учитывать на всех этапах разработки. Осознание потенциальных проблем позволяет заранее планировать пути их решения и снижать риски срыва сроков или ухудшения качества итогового продукта.

Одной из первых и наиболее очевидных сложностей является проектирование нелинейного повествования. Для того чтобы предоставить игроку ощущение свободы выбора, разработчику необходимо продумать множество альтернативных сценариев развития событий. При этом важно не только создать альтернативные пути, но и обеспечить их логическую связность и равную интересность для игрока. Без тщательной проработки возможен дисбаланс, когда один путь будет намного привлекательнее остальных, или, наоборот, все альтернативы будут казаться малозначимыми, что снижает мотивацию игрока исследовать различные варианты.

Второй важной проблемой является парсинг пользовательского ввода. Даже если используется ограниченный набор команд, игроки могут вводить их в разных формах: с ошибками, синонимами, сокращениями. Чтобы избежать постоянных недопониманий между игроком и игрой, необходимо либо разработать очень гибкий парсер, либо чётко ограничить формат ввода

и обучить игрока использовать его правильно через систему подсказок и примеров. Разработка эффективного парсера требует знания принципов работы конечных автоматов, регулярных выражений и основ естественной обработки текста (NLP).

Следующей сложностью является управление состоянием игрового мира. В текстовом квесте любое действие игрока потенциально может изменить множество элементов мира: открыть новые локации, изменить диалоги NPC, активировать или деактивировать квесты. Без продуманной архитектуры кода существует риск возникновения противоречий или ошибок состояния, когда игра "забывает" о действиях игрока или неправильно их интерпретирует. Для борьбы с этой проблемой применяются принципы инкапсуляции данных, централизованного хранения состояния и строгого контроля всех изменений через интерфейсы взаимодействия.

Особую сложность представляет тестирование текстового квеста. Из-за наличия множества ветвлений и вариантов прохождения требуется тщательно проверять все возможные сценарии. Часто тестирование проводится вручную, что занимает много времени. Автоматизация тестов с использованием фреймворков вроде JUnit помогает упростить этот процесс, но требует предварительного построения чёткой модели всех переходов между состояниями игры.

Наконец, трудности могут возникнуть на уровне сценарного мастерства. Поскольку текстовый квест делает ставку на качество повествования, от разработчика требуется не только умение программировать, но и навыки создания увлекательных, эмоционально насыщенных историй с интересными персонажами и правдоподобными диалогами.

Таким образом, успешная разработка текстового квеста требует всесторонней подготовки — как технической, так и литературной. Только

комплексный подход к решению всех указанных сложностей позволяет создать действительно качественный продукт.

1.6. Средства хранения данных в текстовом квесте

В процессе разработки текстовой квестовой игры одной из ключевых задач становится организация эффективной системы хранения данных. Правильный выбор подхода к сохранению информации о состоянии игрового мира, персонажах, заданиях и прогрессе игрока напрямую влияет на стабильность работы приложения, удобство его сопровождения и перспективы дальнейшего развития проекта.

На современном этапе существует множество различных решений для хранения данных, каждое из которых обладает своими достоинствами и недостатками. Важно тщательно проанализировать возможные варианты, сопоставить их с требованиями проекта и принять обоснованное решение.

1.6.1. Анализ существующих решений

На практике для хранения данных в текстовых играх могут использоваться следующие подходы:

1. **Сериализация объектов в Java.**

Описание метода:

Сериализация представляет собой процесс преобразования объектов в последовательность байтов, которая затем сохраняется в файл. В языке Java данная возможность реализована средствами стандартной библиотеки, что делает её особенно привлекательной для разработчиков.

Для небольших проектов или прототипов сериализация объектов является удобным и эффективным способом хранения данных. Однако по мере увеличения сложности проекта возможны трудности с совместимостью и поддержкой сохранений.

2. **Форматы JSON и XML.**

Описание метода:

Альтернативным подходом является хранение данных в текстовых форматах, таких как JSON или XML. Эти форматы удобны для восприятия человеком и широко поддерживаются множеством инструментов.

Форматы JSON и XML отлично подходят для проектов, где важно обеспечивать доступность и редактируемость данных. Они также являются хорошим выбором при необходимости обмена данными с внешними системами или создания сетевых возможностей в будущем.

3. **Локальные базы данных: SQLite, H2 Database.**

Описание метода:

При увеличении объёма данных или усложнении структуры игрового мира становится целесообразным использование локальных баз данных. Базы данных позволяют организовать хранение информации в табличной форме, обеспечивая быстрый доступ и гибкие возможности поиска.

Использование локальных баз данных оправдано в проектах средней и высокой сложности. Оно позволяет строить более устойчивую и расширяемую архитектуру хранения информации.

4. **Облачные базы данных и удалённые серверы.**

Описание метода:

При разработке многопользовательских игр или необходимости синхронизации прогресса между устройствами может потребоваться использование облачных решений для хранения данных.

Для одиночного текстового квеста применение облачных технологий является избыточным. Однако данное направление может стать перспективным при дальнейшем развитии проекта.

1.6.2. Принятое решение

На основании детального анализа доступных методов хранения данных было принято решение использовать локальную базу данных SQLite для реализации системы сохранения и управления данными в проекте текстового квеста.

SQLite представляет собой лёгкую встроенную базу данных, не требующую отдельного сервера и обеспечивающую удобную работу с табличной структурой данных. Этот выбор обоснован несколькими ключевыми факторами:

1. Надёжность хранения: Использование базы данных значительно снижает риск потери или повреждения данных при росте объёмов или усложнении структуры игрового мира.
2. Гибкость и масштабируемость: При добавлении новых сущностей (например, дополнительных квестов, предметов, персонажей) расширение базы данных будет происходить с минимальными затратами.
3. Поддержка сложных запросов: Возможность выполнения выборок, фильтраций и сортировок позволяет легко реализовать функции поиска предметов в инвентаре, отслеживания выполнения заданий и отображения состояния игры.

Кроме того, применение SQLite предоставляет разработчику возможность использовать стандартные инструменты SQL-запросов, что облегчает последующую интеграцию проекта с внешними системами или возможное расширение игры до многопользовательского формата.

Таким образом, использование SQLite будет способствовать построению более устойчивой, масштабируемой и профессиональной архитектуры хранения данных в проекте.

1.7. Выбор вспомогательных библиотек и инструментов

Разработка даже относительно небольшого проекта требует использования разнообразных вспомогательных средств, которые позволяют значительно упростить процесс программирования, тестирования и сопровождения проекта.

1.7.1. Выбор интегрированной среды разработки

Среда разработки оказывает серьёзное влияние на продуктивность работы разработчика.

Сравнение IDE:

1. IntelliJ IDEA — мощная профессиональная среда, обладающая широким набором инструментов для Java-разработки, включая поддержку систем сборки, средств контроля версий и работы с базами данных.
2. Eclipse — надёжная, но требующая более тщательной настройки среда, подходящая для крупных корпоративных проектов.
3. NetBeans — удобная для обучения и небольших проектов, однако уступающая конкурентам в поддержке современных технологий.

Для выполнения данного проекта будет использоваться IntelliJ IDEA, поскольку она обеспечивает наиболее комфортную и эффективную разработку Java-приложений.

II. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ ТЕКСТОВОЙ КВЕСТОВОЙ ИГРЫ

2.1. Разработка игры

2.1.1 Класс Player

Класс Player представляет собой игрока в игре, включая его основные характеристики, инвентарь и логику взаимодействия с базой данных. Он содержит различные методы для работы с данными игрока, включая загрузку из базы данных, сохранение изменений и регистрацию нового игрока.

1. Поля класса

Класс Player имеет несколько приватных полей:

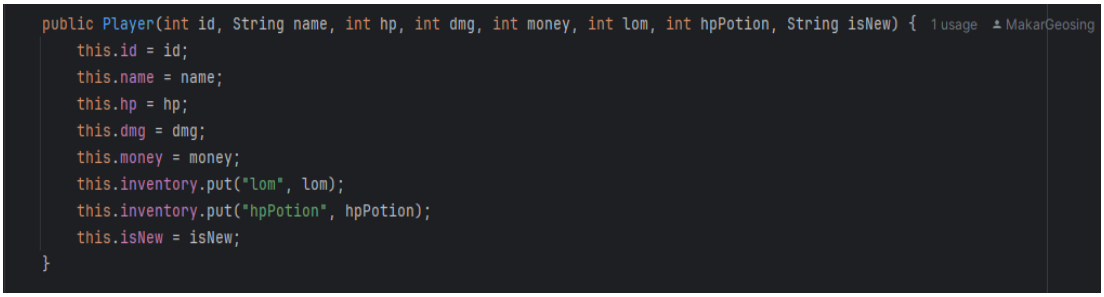
- id: уникальный идентификатор игрока в базе данных.
- name: имя игрока.
- hp: количество здоровья игрока.
- dmg: урон, который игрок может наносить.
- money: количество денег у игрока.
- inventory: инвентарь игрока, который хранится в HashMap, где ключом является название предмета (например, "lom" или "hpPotion"), а значением — количество этого предмета.
- isNew: строка, которая может указывать на статус нового игрока.

```
public class Player {  MakarGeosing +1
    private final int id;  3 usages
    private final String name;  2 usages
    private int hp;  4 usages
    private int dmg;  4 usages
    private int money;  4 usages
    private final HashMap<String, Integer> inventory = new HashMap<>();  10 usages
    private String isNew;  4 usages
```

Рис.2 Поля класса Player

2. Конструктор

Конструктор класса инициализирует все поля класса:



```
public Player(int id, String name, int hp, int dmg, int money, int lom, int hpPotion, String isNew) {  1 usage  MakarGeosing
    this.id = id;
    this.name = name;
    this.hp = hp;
    this.dmg = dmg;
    this.money = money;
    this.inventory.put("lom", lom);
    this.inventory.put("hpPotion", hpPotion);
    this.isNew = isNew;
}
```

Рис.3 Конструктор Player

Он принимает параметры для всех характеристик игрока, включая начальные значения для инвентаря и статус нового игрока.

3. Методы взаимодействия с базой данных

Класс реализует методы для работы с базой данных:

- load — метод для загрузки данных игрока из базы данных по логину.
- save — метод для сохранения изменений данных игрока в базе данных.
- register — метод для регистрации нового игрока.
- checkPassword — метод для проверки пароля игрока.

4. Методы обновления инвентаря

- invUpdate — метод для обновления отображения инвентаря на пользовательском интерфейсе, если это необходимо.
- setInventory — метод для добавления предметов в инвентарь.

Роль класса в игре:

Класс Player является центральным элементом для хранения данных о каждом игроке. Он отвечает за:

- Загрузку и сохранение состояния игрока.
- Управление инвентарем игрока.
- Регистрацию и проверку данных игрока в базе данных.

Взаимодействие с другими частями игры

Игрок может взаимодействовать с другими компонентами игры через его инвентарь, статус здоровья, а также денежные средства. Например, когда игрок использует предмет (например, зелье), состояние его здоровья и количество предметов в инвентаре обновляются, что может быть отражено на пользовательском интерфейсе.

2.1.2 Класс Mob

Класс Mob представляет собой сущность монстра в текстовой квестовой игре. Он обладает характеристиками здоровья (hp), урона (dmg) и имени (name). Основная цель этого класса — моделировать поведение монстров, а также их взаимодействие с игроком в процессе игры. Класс также имеет методы для атаки, получения урона и управления инвентарем игрока во время сражений.

1. Поля класса

- id: уникальный идентификатор монстра.
- hp: здоровье монстра.
- dmg: урон, который монстр наносит игроку.
- name: имя монстра.
- allMobNames: массив с именами монстров, из которого выбирается имя для монстра случайным образом.
- player1: ссылка на объект игрока, с которым происходит взаимодействие (взята из контроллера главного окна).
- gameLogs и playerLogs: объекты, управляющие журналами игры и журналом игрока, куда записываются различные события игры (например, нанесенный урон или смерть монстра).

```
public class Mob {  MakarGeosing +1
    private final int id;  2 usages
    private int hp, dmg;  3 usages
    private String name;  3 usages
    private static final String[] allMobNames = new String[]{"Боб", "Джо", "Риппи", "Алекс", "Альберт", "Бакстер",  3 usages
            "Брендон", "Вилсон", "Хаус", "Джейк", "Кевин",
            "Майкл", "Джозеф"};
    private final Player player1 = MainWindowController.getPlayer();  16 usages
    private final Gamelogs gameLogs = MainWindowController.getGameLogs();  3 usages
    private final Gamelogs playerLogs = MainWindowController.getPlayerLogs();  4 usages
```

Рис.4 Поля класса Mob

2. Конструктор

Конструктор принимает параметры для создания монстра с заданными характеристиками:

```
public Mob(int id, String name, int hp, int dmg) {  1 usage  MakarGeosing
    this.id = id;
    this.name = name;
    this.hp = hp;
    this.dmg = dmg;
}
```

Рис.5 Конструктор Mob

3. Методы

- createMob — статический метод для создания нового монстра с случайными характеристиками. Имя монстра выбирается случайным образом из массива allMobNames, а здоровье и урон инициализируются стандартными значениями (100 и случайный урон соответственно).
- getRndMobName — метод для выбора случайного имени монстра из массива allMobNames.
- mobTakeDmg — метод, который обрабатывает получение урона монстром. В зависимости от типа атаки (Обычная или Ломом), здоровье монстра уменьшается, а также в журнал записываются соответствующие события. Если атака обычная и здоровье монстра больше 10, то монстр получает урон от игрока, и затем атакует игрока в ответ, если атака происходит с использованием лома, то монстр получает в два раза больше урона, а лом исчезает из инвентаря игрока. Если здоровье монстра

становится меньше или равно нулю, то он умирает. Игрок получает деньги, и зависит от текущего задания, могут происходить дополнительные события (например, нахождение кошки).

- mobAttack — метод, который моделирует ответную атаку монстра на игрока. Урон от монстра уменьшает здоровье игрока, а также записывается в журнал.

Роль класса в игре:

Класс Mob реализует монстров, с которыми игрок сражается. Монстры обладают характеристиками (здоровье, урон) и могут взаимодействовать с игроком через механизмы атаки, получения урона и выполнения действий после смерти.

Монстры — это важный элемент игрового процесса, предоставляющий возможность для получения опыта, денег и выполнения заданий. Сражения с монстрами являются ключевой частью игрового процесса, добавляя элемент стратегии и планирования для игрока.

Кроме того, события, связанные с монстрами, могут влиять на дальнейший ход игры (например, выполнение квестов), что также помогает продвигать сюжет игры.

2.1.3 Класс Actions

Этот класс управляет основными игровыми действиями, такими как сражения, торговля и выполнение квестов. Большинство методов в классе отвечают за создание и настройку состояния игры, отображение графических элементов (например, изображений монстра, игрока и интерфейса) и взаимодействие с пользователем.

Основные методы класса:

1. mobFightStart()

Этот метод запускает бой с монстром. Он создает нового монстра с помощью Mob.createMob(), обновляет статистику монстра и игрока, а затем отображает изображения персонажей на экране.

Также скрывает или показывает соответствующие элементы меню, чтобы игрок мог выбирать действия, такие как обычная атака или использование ломов.

2. shopStart()

Метод отвечает за начало торгового процесса. Игрок может приобрести ломы и зелья здоровья.

Стоимость предметов генерируется случайным образом через RandomNums.randomCost(), а также отображается на экране.

3. shopBuy(List<String> items)

В этом методе происходит обработка покупки предметов в магазине. Он проверяет, достаточно ли денег у игрока, чтобы совершить покупку, и затем обновляет его инвентарь.

4. rndEvent()

Метод генерирует случайное событие (бой, торговля или квест) в зависимости от случайного числа. Это добавляет элемент непредсказуемости в игровой процесс.

5. updateStats()

Этот метод обновляет информацию на экране о состоянии игрока или монстра (например, имя, здоровье, урон, деньги). Он также проверяет, не умер ли игрок, и если здоровье игрока падает до нуля, игра заканчивается.

Взаимодействие с базой данных:

1. saveGameParameters()

Этот метод сохраняет параметры игры в базе данных. Он проверяет, существует ли уже запись для текущего игрока. Если записи нет, она создается, если есть — обновляется.

2. loadGameParameters()

Метод загружает параметры игры из базы данных. Он восстанавливает данные игрока и монстра, а также текущие игровые события.

2.1.4 Класс MainWindowController

1. Поля класса

- ResourceBundle resources: используется для загрузки локализованных ресурсов (например, текстов в интерфейсе).
- TextField moveField: поле ввода текста для ввода действия игрока.
- MenuItem firstMenuItem, secMenuItem, thirdMenuItem, fourthMenuItem: элементы меню, которые отображаются в зависимости от ситуации в игре.
- TextArea gameLogsTA, playerLogsTA, playerStatsTA, mobStatsTA: текстовые области для отображения логов игры, логов игрока, статистики игрока и монстра.
- ImageView mobAvatarGame, playerAvatar, playerAvatarGame, mobAvatar, lomAvatarGame: изображения аватаров игрока и монстра.
- Label playerAvatarLbl, mobAvatarLbl: подписи для аватаров игрока и монстра.
- AnchorPane mainPaneMobFight, actionsPane, mainPaneShop, mainPaneBlank, actionsPaneBlank, mainPaneQuest1, mainPaneQuest2: различные панели для отображения разных экранов игры (бой, магазин, квесты).

2. Основная логика

Игровые элементы:

- Класс использует переменные типа Player и Gamelogs для управления состоянием игрока и хранения логов игры.
- Есть несколько статичных изображений для отображения аватаров монстров, игрока и других игровых элементов.
- Логика игры активно зависит от переменных, таких как quest (квест) и флага cat, которые меняются в ходе игры.

Методы для обработки событий:

- initialize(): Этот метод инициализирует все компоненты интерфейса и запускает игру в зависимости от загруженных параметров. Он проверяет, в какой игровой сцене находится игрок, и запускает соответствующие действия (бой, магазин, квесты).

Обработчики действий игрока:

- submitMenuAction(): Обрабатывает введенный игроком ход, в зависимости от текущей сцены (например, бой, магазин, квесты).

Методы для каждого типа действия:

- Бой (handleMobFight): Обрабатывает различные действия, такие как атака или отступление от монстра.
- Магазин (handleShop): Обрабатывает покупку и добавление предметов в корзину.
- Квесты (handleQuest1, handleQuest2): Обрабатывает события и диалоги, связанные с квестами.
- Логика для действий с инвентарем: Методы для добавления и удаления предметов из корзины, а также использования предметов в инвентаре (например, зелье здоровья).

Методы для обновления интерфейса:

- showInvalidMoveError(): показывает ошибку, если введено неверное действие.
- clearGameLogsBtnAction(): очищает логи игры.
- undoClearGameLogsAction(): возвращает очищенные логи.
- exitBtnAction(): сохраняет данные игрока и завершает игру.

3. Интерактивность с интерфейсом

Обработчики событий для кнопок и панелей:

События мыши, такие как lomShopClicked, hpPotionShopClicked, обрабатывают добавление или удаление предметов из корзины в магазине.

Методы, связанные с наведением мыши на аватары (playerAvatarEntered, mobAvatarEntered), показывают подсказки о персонажах.

Взаимодействие с элементами интерфейса (например, с MenuItem, ImageView, Label) происходит через биндинг свойств и динамическое обновление видимости элементов.

2.1.5 Класс RegLogController

Этот класс отвечает за обработку действий пользователя на экране регистрации и логина в игре. Он использует JavaFX для работы с графическим интерфейсом пользователя, предоставляя два основных функционала: регистрацию нового пользователя и вход в систему для уже зарегистрированных пользователей. Весь процесс регистрации и логина сопровождается проверками данных, а также передачей управления между окнами игры после успешной аутентификации.

Поля класса:

- resources — ресурсный биндинг, используемый для загрузки локализованных строк.
- location — URL-адрес для загрузки ресурса (например, fxml файл).

- logPasswordField, regPasswordField — поля для ввода пароля в формах логина и регистрации.
- logLoginField, regLoginField — поля для ввода логина в формах логина и регистрации.
- loginBtn, regBtn — кнопки для входа и регистрации пользователя.
- fieldKeys — карта для хранения ключей полей формы (логин и пароль).
- login — строка для хранения текущего логина пользователя.

Основные методы:

- initialize() — инициализация класса. На данном этапе дополнительных действий не выполняется.
- loginBtnAction(ActionEvent event) — метод, вызываемый при нажатии кнопки логина. Он извлекает введённый логин и пароль, затем вызывает метод валидации для проверки их корректности.
- regBtnAction(ActionEvent event) — метод, вызываемый при нажатии кнопки регистрации. Выполняет проверку введённых данных и, если всё правильно, регистрирует нового пользователя.
- changeStage(String FxmlPath, Button BtnName, String title) — метод для изменения сцены (экрана) в приложении. Он загружает новый fxml-файл и заменяет текущую сцену на новую.
- validation(TextField logField, PasswordField passwordField, String type) — метод для проверки введённых данных. Он проверяет: пустые поля, логин на соответствие регулярному выражению. Пароль на соответствие более сложному регулярному выражению (проверка на наличие цифр, заглавных и строчных букв, специальных символов, и отсутствие пробелов).
- В зависимости от типа ("log" или "reg") выполняет проверку логина и пароля либо для регистрации нового пользователя, либо для логина.

- showAlert(Alert.AlertType alertType, String msg) — статический метод для показа всплывающего окна с сообщением об ошибке или информацией.
- showAlert(Alert.AlertType alertType, String msg, String title, String header) — метод для показа всплывающего окна с дополнительными параметрами для заголовка и текста.
- getLogin() — метод для получения текущего логина пользователя.

2.1.6 Класс Gamelogs

Этот класс управляет логами игры, выводя текстовые сообщения в компонент TextArea.

Поля:

- gameLogsTA — TextArea, в который выводятся логи игры.

Методы:

- Конструктор Gamelogs(TextArea gameLogsObject) — инициализирует объект с переданным TextArea.
- appendLogs(String text, Object... args) — добавляет форматированный текст в логи.
- clearLogs() — очищает текст в TextArea.

2.1.7 Класс Database

Класс Database является частью приложения, отвечающим за управление подключением к базе данных SQLite и инициализацию её структуры (создание таблиц). Это основа для хранения информации о пользователях и игровых параметрах. Ниже приведено подробное описание всех элементов класса.

1. Константы

DB_URL — строка подключения к базе данных. В данном случае используется SQLite:

2. Методы

- getConnection() — возвращает подключение к базе данных:
- initDatabase() — создает таблицы в базе данных, если они ещё не существуют: Этот метод выполняет два SQL-запроса для создания двух таблиц:

1. player: содержит информацию об игроках:

   Id — уникальный идентификатор игрока, автоинкрементируемый.

   playerLogin — уникальный логин игрока.

   playerPassword — пароль игрока.

   playerHp — здоровье игрока (по умолчанию 100).

   playerDmg — урон игрока (по умолчанию 10).

   playerMoney — деньги игрока (по умолчанию 20).

   playerLom — лом игрока (по умолчанию 0).

   playerHpPotion - количество зелий здоровья у игрока

   playerIsNew — флаг нового игрока (по умолчанию true).

2. gameParameters: хранит параметры текущей игры:

   player — логин игрока.

   mobName — имя монстра.

mobHp — здоровье монстра.

mobDmg — урон монстра.

lomCost — стоимость лома.

hpPotionCost — стоимость зелья здоровья.

playerLogs — журналы игрока.

gameLogs — игровые журналы.

currentAction — текущее состояние игры (по умолчанию mainPaneQuest1).

Если таблицы уже существуют, запросы с CREATE TABLE IF NOT EXISTS не будут выполняться снова.

2.1.8 Класс RandomNums

Класс RandomNums предназначен для генерации случайных чисел, используемых в игре. Он включает методы для получения случайных чисел в различных диапазонах, таких как стоимость, урон и другие случайные числа с ограничением.

1. Методы

- randomCost() — генерирует случайную стоимость в диапазоне от 5 до 15: Этот метод использует класс Random для генерации случайного числа в диапазоне от 5 до 14 (верхняя граница не включается). Это может быть использовано, например, для вычисления стоимости игровых предметов, товаров или услуг.
- randomNum(int bound) — генерирует случайное число от 0 до заданного верхнего предела (bound): Этот метод принимает параметр

bound, который определяет верхнюю границу диапазона для случайного числа. Генерируемое число будет от 0 до bound - 1. Например, при вызове randomNum(10) результатом будет случайное число от 0 до 9.

- rndDmg() — генерирует случайный урон в диапазоне от 1 до 11: Этот метод генерирует случайное значение для урона в диапазоне от 1 до 11 (верхняя граница не включается). Подходит для определения урона персонажа или противника в игре.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была разработана текстовая квестовая игра на Java, которая демонстрирует ключевые особенности жанра интерактивной литературы: влияние выбора игрока на развитие событий и глубокую проработку персонажей. Проект позволил закрепить навыки объектно-ориентированного программирования, работы с базами данных и создания графических интерфейсов.

Основные достижения:

1. Успешная реализация системы сохранения и загрузки игры с использованием SQLite.
2. Создание интерактивных квестов с множественными вариантами развития.
3. Разработка удобного интерфейса с использованием JavaFX.

Проблемы и пути их решения:

1. Сложность управления состоянием игры: Решено за счет использования базы данных и четкого разделения логики между классами.
2. Ограниченная вариативность сюжета: В будущем планируется расширить количество квестов и вариантов выбора.

Перспективы развития проекта:

1. Добавление новых квестов и локаций.
2. Улучшение безопасности (хэширование паролей).
3. Реализация сетевого режима для многопользовательской игры.

Разработанная игра может быть опубликована на сайтах для инди игр, а также как основа для более сложных игровых приложений. Работа над проектом подтвердила, что текстовые квесты остаются актуальным жанром, сочетающим простоту реализации с богатыми возможностями для творчества.

СПИСОК ЛИТЕРАТУРЫ

1. Черпаков, И. В. Основы программирования : учебник и практикум для среднего профессионального образования / И. В. Черпаков. — 2-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2025. — 196 с. — (Профессиональное образование). — ISBN 978-5-534-18760-1. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/561922 (дата обращения: 14.05.2025).
2. Кубенский, А. А. Функциональное программирование: учебник и практикум для вузов / А. А. Кубенский. — Москва: Издательство Юрайт, 2025. — 348 с. — (Высшее образование). — ISBN 978-5-9916-9242-7. — Текст: электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/561074 (дата обращения: 29.04.2025).
3. Функциональное программирование. Теоретические и практические основы для разных языков : учебник для вузов / под общей редакцией А. Ю. Анисимова, А. Е. Трубина, Ф. А. Мастяева. — Москва: Издательство Юрайт, 2025. — 135 с. — (Высшее образование). — ISBN 978-5-534-20518-3. — Текст: электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/558300 (дата обращения: 29.04.2025).
4. Зыков, С. В. Программирование: учебник и практикум для вузов / С. В. Зыков. — 2-е изд., перераб. и доп. — Москва: Издательство Юрайт, 2025. — 285 с. — (Высшее образование). — ISBN 978-5-534-16031-4. — Текст: электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/560815 (дата обращения: 29.04.2025).
5. Нестеров, С. А. Базы данных: учебник и практикум для вузов / С. А. Нестеров. — 2-е изд., перераб. и доп. — Москва: Издательство Юрайт, 2024. — 258 с. — (Высшее образование). — ISBN 978-5-534-

18107-4. — Текст: электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/536687 (дата обращения: 29.04.2025).

6. Стружкин, Н. П. Базы данных: проектирование: учебник для среднего профессионального образования / Н. П. Стружкин, В. В. Годин. — Москва: Издательство Юрайт, 2025. — 477 с. — (Профессиональное образование). — ISBN 978-5-534-11635-9. — Текст: электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/566509 (дата обращения: 29.04.2025).
7. Стасышин, В. М. Базы данных: технологии доступа: учебник для вузов / В. М. Стасышин, Т. Л. Стасышина. — 2-е изд., испр. и доп. — Москва: Издательство Юрайт, 2025. — 164 с. — (Высшее образование). — ISBN 978-5-534-08687-4. — Текст: электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/562868 (дата обращения: 29.04.2025).
8. Стружкин, Н. П. Базы данных: проектирование. Практикум: учебник для вузов / Н. П. Стружкин, В. В. Годин. — Москва: Издательство Юрайт, 2025. — 291 с. — (Высшее образование). — ISBN 978-5-534-00739-8. — Текст: электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/561215 (дата обращения: 29.04.2025).
9. Гордеев, С. И. Организация баз данных: учебник для вузов / С. И. Гордеев, В. Н. Волошина. — 2-е изд., испр. и доп. — Москва: Издательство Юрайт, 2025. — 691 с. — (Высшее образование). — ISBN 978-5-534-21115-3. — Текст: электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/559377 (дата обращения: 29.04.2025).
10. Илюшечкин, В. М. Основы использования и проектирования баз данных: учебник для среднего профессионального образования /

В. М. Илюшечкин. — Москва: Издательство Юрайт, 2025. — 213 с. — (Профессиональное образование). — ISBN 978-5-534-01283-5. — Текст: электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/562514 (дата обращения: 29.04.2025).

11. Эндрю Дэвидсон Pro Java 8 Games Development (англ.) — Apress, 2018. — 450 p.
12. Рауль-Габриэль Урма, Марио Фуско, Алан Майкрофт «Современный Java: Лямбды, потоки, функциональное программирование» (Modern Java in Action) — М.: ДМК Пресс, 2021. — 592 с.
13. Скотт Оукс «Java: Оптимизация производительности» (Java Performance: The Definitive Guide) — СПб.: БХВ-Петербург, 2020. — 528
14. Венкат Субраманьям «Java 8: Полное руководство» (Java 8 Programming for Beginners) — М.: Эксмо, 2023. — 768 с.
15. Джошуа Блох Java. Справочник разработчика — М.: ДМК Пресс, 2021. — 400 с.
16. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования — СПб.: Питер, 2022. — 496 с.
17. Браун Д. Разработка игр на Java. Руководство для начинающих —М.: ДМК Пресс, 2020. — 320 с.
18. Эккель Б. Философия Java — СПб.: Питер, 2021. — 1168 с.
19. Руководство по sqlite [Электронный ресурс] // Онлайн-руководство. URL: https://metanit.com/sql/sqlite/
20. Руководство по java [Электронный ресурс] // Онлайн-руководство. URL: https://metanit.com/java