УНИВЕРСТИТЕТСКИЙ ЛИЦЕЙ № 1523 ПРЕДУНИВЕРСИТАРИЯ НИЯУ МИФИ

СОГЛАСОВАНО УТВЕРЖДАЮ

Лаборант ИИКС НИЯУ МИФИ Лаборант ИИКС НИЯУ МИФИ

 Серебрякова Д.Р.  Серебрякова Д.Р.

подпись расшифровка подписи подпись расшифровка подписи

12.05.2025 12.05.2025

дата дата

ОТЧЕТ

О ПРОЕКТНОЙ РАБОТЕ

по теме:

РАЗРАБОТКА ТЕКСТОВОЙ КВЕСТ-ИГРЫ

Руководитель ПР,

лаборант ИИКС НИЯУ МИФИ  Серебрякова Д.Р. 12.05.2025

Москва 2025

## Содержание

[Содержание 2](#_Toc671028157)

[Введение 3](#_Toc386304882)

[Этапы реализации проекта 5](#_Toc2009136853)

[Теоретическая подготовка 5](#_Toc1181226771)

[Выбор технологий: 5](#_Toc269085037)

[Язык 5](#_Toc2142134233)

[Нейросеть 5](#_Toc1468245098)

[База данных 5](#_Toc1658929806)

[Архитектура Telegram-бота 8](#_Toc1810214828)

[Пример работы бота 10](#_Toc576668336)

[Генерация текста и сценария 12](#_Toc1978982760)

[Start\_command 12](#_Toc1437370943)

[Start\_story 13](#_Toc291509679)

[Handle\_dialog 13](#_Toc170240376)

[Parse\_response 14](#_Toc839099627)

[Process\_option 15](#_Toc1805636161)

[Call\_model 15](#_Toc856429994)

[Safe\_llm\_invoke 16](#_Toc1205646161)

[Тестирование 17](#_Toc942099843)

[Заключение 19](#_Toc1003907344)

[Список литературы 20](#_Toc825862404)

[Приложение 21](#_Toc1111003914)

# **Введение**

В последние годы технологии искусственного интеллекта всё активнее внедряются в различные сферы человеческой деятельности — от медицины до искусства.  
 Одним из направлений, где ИИ особенно заметен, является сфера развлечений и интерактивных медиа. Текстовые квест-игры — это жанр, который сочетает в себе элементы литературы, интерактивного повествования и гейм-дизайна.  
 Они позволяют игроку принимать решения, влияющие на развитие сюжета, и, таким образом, создают уникальный пользовательский опыт.

Традиционно такие игры создавались вручную: каждый возможный выбор пользователя прописывался сценаристами, что ограничивало количество сюжетных развилок и делало процесс разработки трудоёмким.  
 Однако с развитием языковых моделей, способных генерировать связный и логичный текст, появилась возможность создавать динамичные, нелинейные истории, которые реагируют на действия игрока в реальном времени.

Продуктом нашего проекта стал Telegram-бот, который способен вести текстовую квест-игру, используя возможности нейросетевой генерации.  
Такой бот должен не просто реагировать на команды, а выступать в роли рассказчика, генерируя сюжет, предлагать варианты выбора и запоминать действия игрока.  
Реализация подобной системы позволяет продемонстрировать практическое применение технологий машинного обучения и сделать взаимодействие с ИИ увлекательным и познавательным.

**Актуальность**

Актуальность проекта обусловлена растущим интересом к интерактивным форматам и нейросетевым технологиям. Наш текстовый квест, построенный на базе генеративного ИИ, сочетает в себе преимущества классических игр-книг с современными возможностями искусственного интеллекта. Это позволяет создавать уникальные игровые сценарии, где каждый выбор пользователя действительно влияет на развитие сюжета, а не ограничивается заранее прописанными ветками.

**Цель проекта:**

Исследовать способы интеграции языковых моделей в Telegram-среду на примере квест-игры с нелинейным сюжетом.

**Задачи:**

Для достижения поставленной цели мы выдвинули перед собой ряд задачь:

1. Изучить технологии, необходимые для создания Telegram-бота и работы с нейросетями.

2. Выбрать подходящие фреймворки, платформы и язык программирования.

3. Разработать структуру сценария и механику взаимодействия игрока с ботом.

4. Интегрировать модель ИИ для генерации сюжетных диалогов.

5. Настроить хранение прогресса игрока.

6. Реализовать и протестировать работу Telegram-бота.

# **Этапы реализации проекта**

## Теоретическая подготовка

На этом этапе мы изучили основы работы с Telegram Bot API, принципы использования фреймворков для Telegram-ботов , а также подходы к генерации текста с помощью нейросетей.

## **Выбор технологий:**

### **Язык**

При реализации нашего проекта мы остановились на языке программирования **Python**, и этот выбор был не случайным. Python — это один из самых популярных и универсальных языков, особенно в сфере **искусственного интеллекта, работы с данными и разработки Telegram-ботов**. Вот основные причины:

1. **Python** - высокоуровненый язык программирования
2. Большое количество библиотек
3. Поддержка современных технологий ИИ
4. Постоянные патчи и обновления
5. Кроссплатформенность и простота развёртывания

### **Нейросеть**

Mistral был выбран за его скорость генерации и способность поддерживать контекст в длинных диалогах, что идеально подходит для текстовых квестов. Модель обеспечивает плавное повествование при умеренных вычислительных затратах. Так же Mistral бесплатная и имеет возможность обращения к ней по API-ключу.

### **База данных**

Каждому пользователю соответствует запись в базе данных с текущим прогрессом, историей и идентификатором сессии. Для хранения прогресса игроков и других данных мы выбрали базу данных **SQLite**, потому что она идеально подходит для небольших Telegram-проектов с индивидуальным пользовательским взаимодействием. Вот ключевые причины этого выбора:

* **Простота и легкость интеграции**

SQLite встроена в Python и не требует отдельного сервера или сложной настройки. Это позволяет быстро подключить её к боту и приступить к работе.

* **Подходит для локальных и малых проектов**

Наш бот обрабатывает диалоги от одного пользователя за раз, и нет необходимости в масштабируемой многопользовательской СУБД, такой как PostgreSQL или MySQL. SQLite отлично справляется с хранением пользовательского прогресса и историй даже при увеличении количества пользователей.

* **Поддержка SQL-запросов**

Позволяет удобно работать с таблицами, фильтрацией, обновлением и извлечением данных — всем, что нужно для отслеживания выборов игрока и восстановления состояния игры.

* **Быстрота работы**

Для большинства квестов не требуется хранить большие объёмы информации. SQLite обеспечивает отличную скорость доступа и записи при минимальной нагрузке.

* **Кроссплатформенность**

Упрощает перенос проекта и делает его удобным для разработки и развёртывания.

Для обоснования выбора мы провели сравнительный анализ нескольких популярных СУБД: MySQL, SQLite, универсального SQL и NoSQL-подходов. В Таблице 1 приведены сводные характеристики.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Точка сравнения | MYSQL | SQLITE | SQL | NOSQLL |
| Тип базы данных | Относительный | Относительный | Относительный | Нереляционный |
| Пригодность применения | Средний и крупный масштаб | Малый и средний размер | Средний и крупный размер | Крупномасштабные и неструктурированные данные |
| Производительность и масштабируемость | Высокая производительность и масштабируемость | Легкое и простое развертывание | Высокая производительность и масштабируемость | Высокая масштабируемость и быстрая обработка данных |
| Требования к серверному процессу | Требуется отдельный серверный процесс | Не требует отдельного серверного процесса | Требуется отдельный серверный процесс | Не требует отдельного серверного процесса (зависит от реализации) |
| Расширенные функции и связи | Предлагает более продвинутые функции и сложные связи между таблицами | Предлагает более продвинутые функции и сложные связи между таблицами | Простота и портативность | Обеспечивает гибкость структур данных и эффективную обработку неструктурированных или полуструктурированных данных. |
| Гибкость запроса | Поддерживает SQL-запросы | Поддерживает SQL-запросы | Поддерживает SQL-запросы | Различается (некоторые поддерживают запросы, подобные SQL, другие имеют пользовательские языки запросов) |
| Гибкость структуры данных | Структурированный (табличный) | Структурированный (табличный) | Структурированный (табличный) | Гибкий |
| Расходы | Доступны бесплатная и коммерческая версии | Бесплатно и в общественном достоянии | Доступны бесплатная и коммерческая версии | Различается (некоторые с открытым исходным кодом, некоторые коммерческие) |
| Варианты резервного копирования и восстановления | Доступно несколько вариантов (например, двоичные журналы, полные резервные копии) | Ограниченные возможности (например, копии на уровне файлов) | Доступно несколько вариантов (например, полные резервные копии, журналы транзакций) | Различается (некоторые предоставляют встроенные механизмы репликации и резервного копирования, другие полагаются на внешние инструменты) |
| Примеры баз данных | Система управления персоналом, Интернет-магазин | Мобильные приложения, встроенные системы | MySQL, SQLite, MongoDB, Cassandra, Firebase | DynamoDB, Couchbase, Elasticsearch |

Таблица 1. Сравнение СУБД

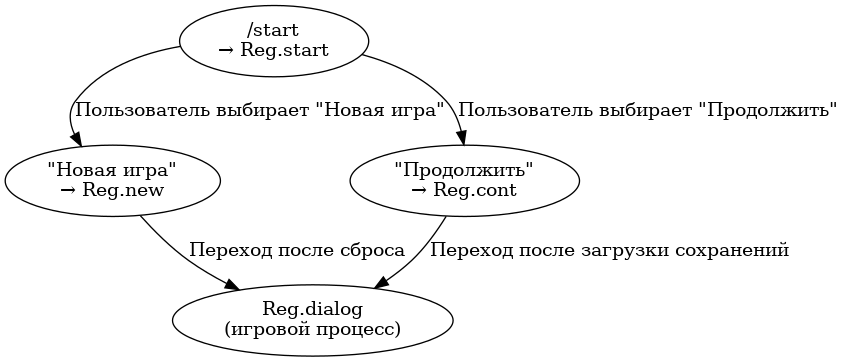
Анализ показал, что **SQLite наилучшим образом соответствует целям нашего проекта**. В отличие от тяжёлых серверных решений, она обеспечивает необходимую гибкость, высокую скорость при работе с небольшим объёмом данных и не требует дополнительной инфраструктуры. Благодаря встроенной поддержке SQL, работа с базой остаётся удобной, а простота подключения делает её идеальной для Telegram-ботов.

## **Архитектура Telegram-бота**

В разработке Telegram-бота мы использовали концепцию **конечного автомата состояний (FSM — Finite State Machine)** для управления процессами игры. FSM — это модель управления логикой приложения, где система может находиться только в одном из заранее определенных состояний(Reg.start, Reg.dialog, Reg.new, Reg.cont). Переход между состояниями происходит в ответ на события или действия пользователя. Это позволило структурировать логику взаимодействия с пользователем и сделать код более организованным и масштабируемым. Порядок переходов изображен на Рисунке 1.

Бот работает с тремя основными состояниями:

* **Начать квест —** пользователь выбирает, начать новую игру или продолжить старую. Отправляется приветственное сообщение с вариантами действий.
* **Продолжить** — если у пользователя есть сохранённый прогресс, бот восстанавливает его историю и предлагает продолжить игру с того момента, где она была прервана.
* **Новая игра** — обнуляются все предыдущие данные пользователя, создаётся новая сессия, и начинается новая история.

Рисунок 1. FSM

Для удобства взаимодействия была реализована **встроенная клавиатура**:

* Кнопки для выбора действий (" Начать квест ✅", " Продолжить ▶️", "Новая игра 🌍") упрощают навигацию и делают процесс игры интуитивно понятным.
* В диалогах предусмотрены inline-кнопки для выбора вариантов развития сюжета.

Inline-кнопки — это особый тип интерактивных элементов в Telegram, которые встраиваются прямо в текст сообщения и позволяют пользователю взаимодействовать с ботом, не покидая текущий чат. В отличие от обычных Reply-кнопок, которые заменяют клавиатуру, inline-кнопки отображаются под сообщением и сохраняют историю переписки.

Использование FSM (конечного автомата) в проекте дало три ключевых преимущества: во-первых, система теперь четко разделена на логические этапы (меню, игра, загрузка), что упрощает понимание кода. Во-вторых, добавление новых функций стало проще - можно модифицировать отдельные состояния, не затрагивая всю систему. В-третьих, исключены случайные ошибки перехода между этапами, так как все возможные сценарии взаимодействия теперь строго определены. Это сделало бота более стабильным и предсказуемым в работе. На Рисунке 2 представлена блок схема работы бота.

Рисунок 2. Блок-схема бота

#### **Пример работы бота**



Рисунок 3. Старт бота

На Рисунке 3 показан запуск Telegram-бота "Crystals of Fate". После команды /start, бот приветствует пользователя и предлагает три варианта действий через встроенную клавиатуру. Опции представлены в виде удобных кнопок, встроенных в интерфейс Telegram. Пользователь не вводит текст вручную — всё управление происходит нажатием кнопок.

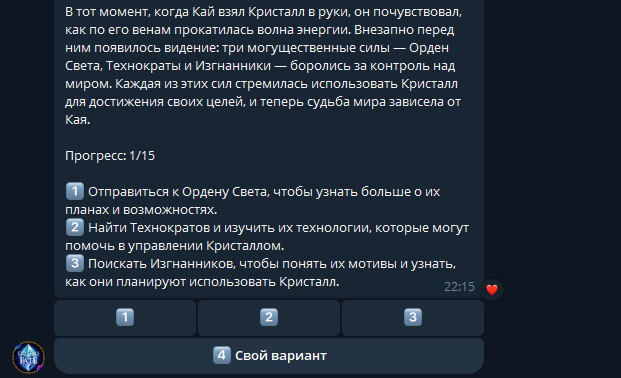
Каждая кнопка отправляет соответствующий сигнал боту, и он переходит в одно из трёх состояний (начало, продолжение, новая игра) с помощью FSM (машины состояний). Бот сразу записывает или обновляет данные пользователя в базе SQLite.

Рисунок 4. Выбор развития сюжета

На Рисунке 4 пользователь достиг сюжетной развилки — здесь игра предлагает выбор, от которого зависит дальнейшее развитие событий.

В тексте описана ситуация в квесте: главный герой Кай сталкивается с важным решением. Прогресс пользователя отображается в формате:( Прогресс: ?/15), что говорит о прохождении какого-либо этапа из пятнадцати. Ниже представлены три варианта выбора**.** Кнопка "Свой вариант" позволяет пользователю ввести нестандартный ответ — эта функция повышает интерактивность и гибкость квеста.

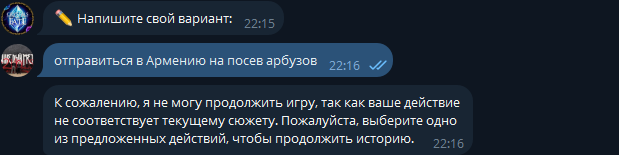


Рисунок 5. Логика выбора свойего варианта ответа

На рисунке 5 рассматривается ситуация, когда пользователь выбирает “Свой вариант” и его ответ никак не связывается с предыдущим развитием событий. В такой ситуации бот предложит выбрать один из уже сгенерированных вариантов ответа или попробовать написать что-то более связанное с контекстом.

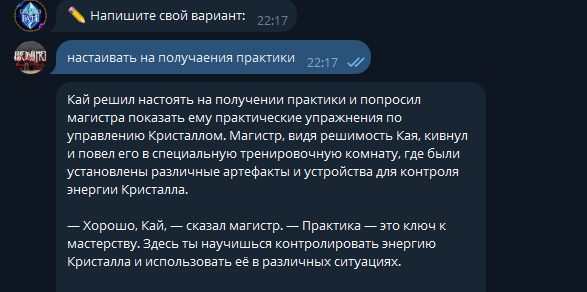


Рисунок 6. Принятый вариант своего ответа

Как видно на Рисунке 6 нейросеть, интегрировання в бота, проанализировала ответ и допустила развитие сюжета, опираясь на сообщение пользователя.

## **Генерация текста и сценария**

Сценарий игры хранится в виде документа. Он загружается и разбивается на части, создавая векторное пространство для поиска контекста. Используется MistralAI для генерации ответов с вариантами действий.

Рассмотрим основные функции, которые были использованы в программе. Исходный код приведен в приложении.

### **Start\_command**

**Назначение:** Обработка команды /start.

**Логика:**

* Переводит пользователя в состояние Reg.start.

Функция start\_command — это отправная точка взаимодействия пользователя с ботом, которая активируется при команде /start или первом запуске. Она выполняет ключевую роль в инициализации игрового процесса, переводя пользователя в стартовое состояние Reg.start. Здесь система ожидает от пользователя выбора одного из трёх базовых действий, представленных через интерактивную клавиатуру: начать новое приключение, продолжить сохранённую игру или полностью сбросить прогресс.

Технически функция решает три основные задачи. Во-первых, она явно устанавливает FSM-состояние Reg.start, создавая чёткую точку входа в логику бота и исключая обработку случайных сообщений. Во-вторых, формирует структурированное приветственное сообщение с объяснением возможных действий, что особенно важно для новых пользователей. В-третьих, отображает специально разработанную reply-клавиатуру с тремя крупными кнопками, которые визуально выделяют основные опции.

Особенность этого состояния в его переходном характере — оно существует исключительно для однозначного определения намерений пользователя перед погружением в основной игровой процесс. Как только выбор сделан, система мгновенно переключается в соответствующий режим работы: либо начинает новый сценарий (Reg.dialog), либо загружает сохранённые данные, либо выполняет полный сброс прогресса (Reg.new). Такой подход не только организует чёткую структуру взаимодействия, но и значительно упрощает обработку пользовательских запросов на последующих этапах, выступая своеобразным фильтром входящих команд.

* Отправляет приветственное сообщение с клавиатурой выбора действий (начать квест, продожить, новая игра).

Ключевыми элементами являются ReplyKeyboardMarkup для отображения кнопок, FSM для управления диалогом.

**Start\_story**

**Назначение**: Обработка выбора действия из стартового меню.

**Логика:**

* **«✨ Начать квест ✅»**:
* Удаляет старую запись пользователя в БД.
* Создает новую сессию (thread\_id) и сохраняет начальный прогресс (progress=1)
* Запускает handle\_dialog для генерации первого сообщения игры.
* **«📜 Продолжить ▶️»**:
* Проверяет наличие сохраненной игры в БД.
* Восстанавливает прогресс и историю сообщений.
* **«🔄 Новая игра 🌍»**:
* Удаляет данные пользователя из БД и сбрасывает состояние.

### **Handle\_dialog**

**Назначение**: Основной обработчик игрового диалога.  
 Это главная функция, которая управляет всей игрой. Когда игрок выбирает действие (например, "Начать квест" или кнопку с вариантом ответа), бот обрабатывает выбор именно здесь.

Сначала функция проверяет user\_id пользователя из Телеграма и заглядывает в базу данных, где записан текущий этап игры и предыдущие реплики.

Если только начало игры (progress=1), бот открывает файл со сценарием story.docx. Он выбирает оттуда подходящие кусочки текста и добавляет их в "правила игры" для нейросети.

Потом бот берёт последние несколько сообщений (обычно 3 реплики и 3 своих ответа) и отправляет их нейросети Mistral. Нейросеть, как хороший рассказчик, придумывает продолжение истории на основе этих данных.

Когда ответ приходит, бот разбирает его: отделяет саму историю от вариантов действий ( например "1 — Пойти налево, 2 — Пойти направо"). Если игрок прошёл 15 шагов — бот торжественно объявляет конец истории, хвалит тебя и предлагает начать заново.

Все выборы сохраняются в базе данных. Даже если закрыть Телеграм игра продолжится с того же места. А если что-то пойдёт не так (например, интернет заглючит), бот попросит попробовать ещё раз, не теряя прогресс игрока.

История меняется в зависимости от выборов юзера — нейросеть каждый раз немного по-новому рассказывает сюжет. И всё это работает благодаря тому, что разные функции кода передают друг другу информацию (работают кооперативно).

### **Parse\_response**

Функция служит ключевым интерфейсом между выводом нейросети и игровой механикой, преобразуя неструктурированный текст в игровые события.

Система использует многослойное регулярное выражение. Она не просто ищет цифры в тексте, а анализирует всю структуру сообщения по специальному шаблону: сначала захватывает номер варианта (может быть с точкой, скобкой или тире), затем извлекает связанный с ним текст, при этом умно определяя границы каждого варианта через механизм lookahead. Это позволяет корректно обрабатывать даже сложные случаи вроде вариантов, содержащих собственные цифры или пунктуацию.

После извлечения данных включается система валидации — каждый захваченный вариант проверяется на полноту данных (наличие и номера, и текста), при необходимости обрезаются слишком длинные формулировки (чтобы поместиться в кнопку Telegram), и строго ограничивается количество вариантов тремя наиболее подходящими. Здесь же происходит тонкая обработка текста — удаление лишних пробелов, нормализация кавычек и скобок, что особенно важно для корректного отображения в интерфейсе.

Финальный этап — упаковка результатов в две четкие структуры: основное повествование (которое пойдет в текстовое сообщение) и пронумерованные варианты действий. Особое внимание уделяется совместимости с Telegram API — тексты кнопок специально форматируются, а callback-данные формируются по строгому шаблону "option\_[номер]", что позволяет потом однозначно идентифицировать выбор пользователя. При этом система сохраняет гибкость — если вариантов действий нет (например, в концовке игры), функция корректно возвращает весь текст, позволяя плавно завершать игровые сессии.

**Process\_option**  
 Функция **process\_option** — это мост между нажатием кнопки и игровой логикой.

Когда игрок нажимает на инлайн-кнопку (например, "1️⃣"), функция берет цифру из callback\_data (скажем, "option\_1" → 1), создаёт фейковое сообщение с этим числом как текстом, и кидает его в handle\_dialog — будто игрок сам ввёл эту цифру. Это нужно чтобы не писать отдельную логику для кнопок и текста, а обрабатывать всё единообразно.

Технический секрет в строчке fake\_message = Message(text=option\_number, ...) — это "обманка", которая заставляет систему думать, что пользователь действительно отправил номер варианта, хотя он просто нажал кнопку.

### **Call\_model**

Функция call\_model выступает в роли надежного посредника между ботом и нейросетью MistralAI, обеспечивая стабильную генерацию ответов даже в неидеальных условиях. Когда боту нужно получить продолжение истории, он передает в эту функцию текущий контекст диалога — все предыдущие реплики и системные промпты, аккуратно упакованные в специальный формат.

Особенность функции в ее устойчивости к внештатным ситуациям. Она автоматически предпримет три попытки запроса с экспоненциально растущими паузами (сначала 2 секунды, потом 4, затем 8), если нейросеть временно недоступна или отвечает ошибкой. Это реализовано через декоратор @retry, который отслеживает сетевые сбои и специфические ошибки API.

Внутри происходит магия преобразования: функция берет необработанные сообщения, передает их нейросети, а затем упаковывает результат в стандартизированный формат, который понимает остальная система. Если после всех попыток запрос так и не прошел, функция вернет понятное сообщение об ошибке, а в логи запишет детали проблемы для последующего разбора.

Интеграция с LangGraph делает этот механизм еще мощнее — функция становится узлом в графе выполнения, что позволяет легко масштабировать систему. Например, при необходимости можно добавить проверку качества ответа или его фильтрацию, просто подключив дополнительные узлы к этому графу.

По сути, это надежный "движок" генерации контента, который в любом случае вернет какой-либо результат, но при этом не зависает и не ломает весь бот при временных проблемах с нейросетью. Все ошибки аккуратно обрабатываются, а система продолжает работать, даже если что-то пошло не так — либо предлагая пользователю повторить запрос, либо возвращая запасной вариант ответа.

### **Safe\_llm\_invoke**

**Назначение**: Безопасный вызов модели с экспоненциальной задержкой при ошибках.

Это умный механизм, который гарантирует, что ваш бот сможет получить ответ от нейросети даже при временных проблемах.

Какие задачи выполняет:

1. Борется с ненадежностью интернета

Если запрос с первого раза не прошел - система сделает ещё попытку.

1. Защищает от перегрузки API

Экспоненциальные паузы(удвоение времени) дают время серверам восстановиться.

Система распознаёт как ошибки только технические сбои: таймауты соединения, ошибку 429 (перегрузка API) и проблемы с API-ключом. При этом механизм не исправляет логические ошибки в запросах, не модифицирует параметры запроса при повторах и строго ограничен тремя попытками - он не будет пытаться выполнить запрос бесконечно. Это обеспечивает баланс между устойчивостью работы и рациональным использованием ресурсов. Схема взаимодействия функций в боте представлена на Рисунке 7.

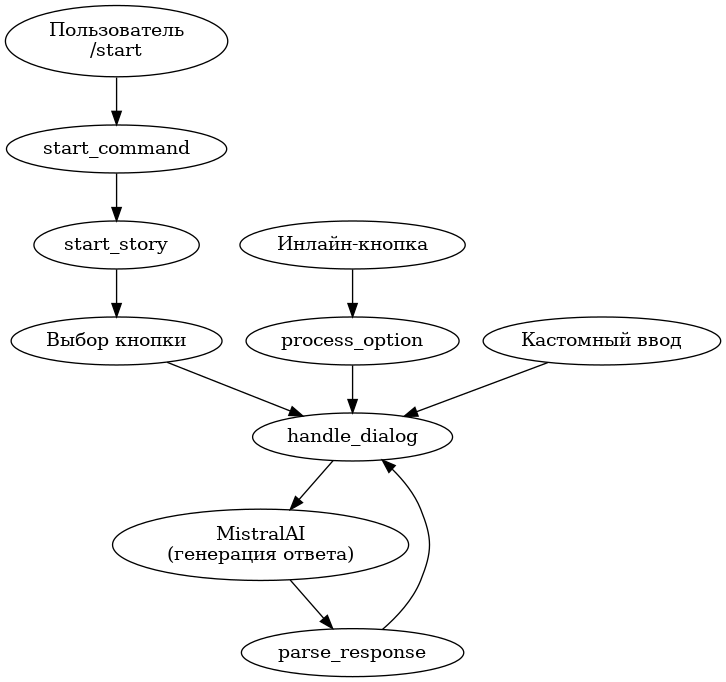


Рисунок 7. схема взаимодействия функций

Ключевые зависимости:

* **LangChain**: Построение цепочек и работа с документами.
* **LangGraph**: Управление состоянием диалога.
* **aiogram**: Взаимодействие с Telegram API.

## **Тестирование**

Для проверки корректной работы Telegram-бота в рамках проекта использовалось юнит-тестирование. Оно позволило протестировать отдельные функции программы без запуска всей системы целиком, что удобно и эффективно на этапе разработки.

Тестированию подвергались ключевые элементы логики бота: команда /start, выбор игрока с помощью кнопок, ручной ввод текста, функции start\_command, handle\_dialog, process\_option и генерация текста с помощью нейросети. Проверялось, как бот реагирует на ввод пользователя, корректно ли он переходит между состояниями (например, из начального меню в активную игру), и правильно ли обрабатывает сценарии.

Тестировалось все с помощью стандартной библиотеки unittest, встроенной в Python, а также инструменов, помогающих проверять работу конечного автомата состояний и взаимодействие с Telegram API. Чтобы не делать реальные запросы к языковой модели во время тестов.

Также тестировались функции, работающие с базой SQLite: добавление новых пользователей, сохранение и восстановление прогресса. Благодаря юнит-тестированию удалось заранее выявить ошибки и упростить процесс доработки кода.

Tестирование повысило надёжность проекта и позволило избежать сбоев при взаимодействии с пользователем.

# **Заключение**

В ходе реализации проекта был разработан интерактивный Telegram-бот с использованием современных технологий генеративного искусственного интеллекта. Ключевые достижения проекта включают:

1. Создание игрового движка на базе нейросетевой модели MistralAI, способного генерировать адаптивные сюжетные линии в реальном времени.
2. Разработку системы сохранения прогресса с использованием SQLite, обеспечивающей непрерывность игрового процесса между сеансами.
3. Реализацию интуитивного интерфейса с интерактивными элементами управления (кнопки выбора действий), значительно улучшающего пользовательский опыт.
4. Внедрение надежного механизма обработки запросов с автоматическими повторами при временных сбоях, что повысило стабильность работы системы.

Проект демонстрирует значительный потенциал применения генеративного ИИ в области интерактивных образовательных и развлекательных систем. Особую ценность представляет реализованный механизм ветвления сюжета, где каждый выбор пользователя оказывает существенное влияние на развитие истории, создавая уникальный игровой опыт для каждого пользователя.

Полученные результаты открывают перспективы для дальнейшего развития системы, включая возможность добавления мультимедийных элементов, расширения сюжетных линий и интеграции с другими платформами. Проект успешно решает поставленные задачи и служит доказательством эффективности использования современных ИИ-технологий в создании интерактивных нарративных систем.

В дальнейшем мы планируем существенно расширить функциональность и возможности нашего бота. Мы уверены, что эти улучшения сделают наш проект еще более увлекательным и востребованным среди пользователей!

## **Список литературы**

1. Telegram Bot API. Официальная документация [Электронный ресурс]. — Режим доступа: <https://core.telegram.org/bots/api>
2. Aiogram Documentation. Документация по библиотеке Telegram-ботов для Python [Электронный ресурс]. — Режим доступа: <https://docs.aiogram.dev>
3. SQLite. Официальная документация [Электронный ресурс]. — Режим доступа: <https://sqlite.org/docs.html>
4. LangChain. Документация по фреймворку генерации и обработки текста с LLM [Электронный ресурс]. — Режим доступа: <https://docs.langchain.com>
5. Mistral AI. API Reference [Электронный ресурс]. — Режим доступа: <https://docs.mistral.ai>
6. Unittest — Unit testing framework [Электронный ресурс]. — Режим доступа: <https://docs.python.org/3/library/unittest.html>
7. Хабр. Статьи по созданию Telegram-ботов на Python [Электронный ресурс]. — Режим доступа: <https://habr.com/ru/search/?q=telegram+бот+python>
8. Real Python. How to Make a Telegram Bot With Python [Электронный ресурс]. — Режим доступа: <https://realpython.com/how-to-make-a-telegram-bot-python/>
9. SQLite Python Tutorial [Электронный ресурс]. — Режим доступа: <https://www.sqlitetutorial.net/sqlite-python/>

## **Приложение**

|  |  |
| --- | --- |
| **Сокращение** | **Расшифровка** |
| ИИ | Искусственный интеллект |
| AI | Artificial Intelligence — искусственный интеллект (английский аналог ИИ) |
| API | Application Programming Interface — интерфейс программирования приложений |
| FSM | Finite State Machine — конечный автомат состояний |
| SQL | Structured Query Language — язык структурированных запросов |
| SQLite | Встраиваемая легковесная реляционная база данных, реализующая стандарт SQL |
| NoSQL | Not Only SQL — система управления базами данных, не основанная на таблицах |
| СУБД | Система управления базами данных |
| LLM | Large Language Model — большая языковая модель |
| NLP | Natural Language Processing — обработка естественного языка |
| UI | User Interface — пользовательский интерфейс |
| JSON | JavaScript Object Notation — формат обмена данными |
| ID | Identifier — уникальный идентификатор |
| Bot API | Программный интерфейс для создания и управления ботами в Telegram |
| Mistral | Языковая нейросетевая модель для генерации текста |
| LangChain | Фреймворк для построения цепочек взаимодействий с языковыми моделями |
| LangGraph | Библиотека для управления состояниями и логикой в диалоговых системах |
| Thread ID | Уникальный идентификатор сессии/диалога пользователя |
| CLI | Command Line Interface — интерфейс командной строки |

Таблица 2. Список сокращений

Ссылка на гитхаб с кодом — <https://github.com/MeatPine/Project-1523>