Санкт-Петербургский государственный университет

**КОВАЛЬ Ангелина Александровна**

**Выпускная квалификационная работа**

**Исследование методик тестирования разговорных интерфейсов**

Уровень образования: бакалавриат

Направление 45.03.02 «Лингвистика»

Основная образовательная программа СВ.5106. «Прикладная, компьютерная и математическая лингвистика (английский язык)»

Профиль «Прикладная, компьютерная и математическая лингвистика (английский язык)»

Научный руководитель:

доцент, Кафедра математической лингвистики,

Митренина Ольга Владимировна

Рецензент:

профессор,

 Кафедра информационных систем в искусстве и гуманитарных науках,

Ягунова Елена Викторовна

Санкт-Петербург

2022

**Аннотация**

В данной ВКР исследуется проблема тестирования разговорных интерфейсов. Целью работы является исследование методик тестирования ПО, которые можно использовать для определения качества чат-ботов, и оценка их работы. В тексте рассматриваются теоретические вопросы тестирования программных продуктов, разработки и тестирования диалоговых интерфейсов, а также устройство платформы JAICP. В практической части данного исследования описывается создание диалогового агента, выполняющего роль консультанта по правилам игры DnD, процесс и результаты его тестирования.

*Ключевые слова:* диалоговая система, разговорный интерфейс, чат-бот, тестирование ПО, тестирование чат-ботов, методика тестирования.

The graduation qualification paper deals with the issue of conversational agents testing and is aimed at determining the methods of software testing which are applicable to evaluating a chat-bot’s quality. The paper addresses theoretical questions of software testing, the design and testing of dialogue interfaces, and the structure of JAICP – a platform for chat-bots’ development. In the second part of the paper you can find the description of a conversational agent which acts as a consultant for the rules of the game DnD, and the process and the results of its testing.

*Keywords:* dialogue system, conversational agent, chat-bot, software testing, chat-bot testing, testing methods.

**Оглавление**

Введение…………………………………………………………….……....5

**1. Тестирование и диалоговые интерфейсы**

**1.1 Тестирование программных продуктов**

1.1.1 Определение тестирования……...…………………………………...7

1.1.2 Подходы к тестированию…………………………………………….9

1.1.3 Процесс тестирования………………………….………………...…11

1.1.4 Подготовка к тестированию………………………….….……….…12

1.1.5 Тест-дизайн……………………………………………….…………14

1.1.6 Методики тестирования…………………………………………….17

1.1.7 Автоматизация тестирования………………………………………22

1.1.8 Принципы хорошего тестирования……………….……….……….25

**1.2 Диалоговые системы**

1.2.1 Определение диалоговой системы и ее актуальность в современном мире………………………………….……….………….…………...29

1.2.2 Функции диалоговых систем…………………………….…………31

1.2.3 Архитектура диалоговой системы и инструменты для ее

разработки………………………………………………….……..….32

1.2.4 Рекомендации по разработке диалоговой системы……………….35

1.2.5 Компания JUST AI и платформа JAICP………………………….…37

1.2.6 Тестирование диалоговых систем………...……….…………….…40

**2. Проверка методик тестирования на диалоговой системе**

**2.1 Описание чат-бота**

2.1.1 Выбор тематической области для чат-бота………..…….…………44

2.1.2 Бот-справочник о правилах DnD……….……….………….………45

2.2 Устройство чат-бота…………….……….…….……………..……….46

**2.3 Статические методы**

2.3.1 Правила составления паттернов……….….……….…….…………48

2.3.2 Статическое тестирование паттернов………….…….…….………49

2.3.3 Статическое тестирование именованных паттернов..…….………51

**2.4 Тестирование с помощью тест-кейсов**

2.4.1 Ошибки в тестах…………………………………………….………56

2.4.2 Ошибки в паттернах и именованных паттернах……….…………57

2.4.3 Пересечение паттернов…………….…………….…………………58

2.4.4 Ошибки в сценарии………………………….….…….……….……62

2.4.5 Другие ошибки………………………………………………………63

2.4.6 Общие выводы………………….…….…….………………….……64

**2.5 Тестирование с привлечением пользователей**

2.5.1 Загрузка чат-бота в канал…………………..…….…………………65

2.5.2 Общение бота с пользователями……………………………………66

Заключение………………………………………….………………….…70 Список литературы……………………………………………………......73

Приложения……………………………………………………………….…..…76

**ВВЕДЕНИЕ**

Проблема тестирования важна и актуальна для разработчиков любых продуктов, будь то компьютерные программы, инструменты, блюда или что-либо другое. Создатели стремятся сделать продукт максимально качественным, отвечающим как требованиям заказчика (если таковой имеется), так и общепринятым стандартам. Для разработчиков чат-ботов эта задача, безусловно, также является актуальной.

Несмотря на то, что направление создания разговорных интерфейсов развито довольно хорошо, оно еще очень молодо и пока не обладает достаточным набором методов и подходов к тестированию разработок. Вместе с тем, при проектировании разговорных интерфейсов на текущий момент разработчики ориентируются в основном на гипотезы, не имея возможности исследовать и проверять эффективность выдвигаемых им предположений. Иных строго регламентированных методик не существует, либо они находятся в других областях знаний и требуют адаптации под реалии практики создания разговорных интерфейсов.

Поэтому **целью** моей работы является проведение сравнительного исследования и подбор методов для оценки качества ботов и проверки состоятельности продуктовых гипотез.

**Объект работы** – методики тестирования программных продуктов.

**Предмет работы** – применимость и эффективность методик тестирования для тестирования диалоговых систем.

**Задачи работы:**

1. Изучить принципы и методы тестирования программных продуктов;
2. Найти информацию о методах тестирования чат-ботов;
3. Самостоятельно определить набор методов, позволяющих наиболее полно протестировать диалоговую систему;
4. Изучить платформу JAICP;
5. Придумать чат-ботa для проверки эффективности методов тестирования;
6. Создать этого бота с помощью инструментов платформы JAICP;
7. Протестировать бота с помощью выбранных методов;
8. Сделать вывод об эффективности методов для оценки качества чат-ботов.

При выполнении работы применялись следующие **методы**: теоретический анализ литературных источников по проблеме исследования, анализ, синтез информации, моделирование, программирование, тестирование.

**1. Тестирование диалоговых интерфейсов**

**1.1. Тестирование программных продуктов**

**1.1.1 Определение и историческое развитие тестирования**

**Тестирование** – это процесс оценки конечного продукта, целью которого является определение его качества, выявление и исправление дефектов. Технически тестирование представляет собой выполнение приложения на некотором множестве данных; полученные результаты сверяются с эталонными, что позволяет установить соответствие различных свойств и характеристик приложения тем, которые требуются заказчику или прописаны в соответствующей документации.

Тестирование как неотъемлемая часть разработки программного продукта развивалось на протяжении нескольких десятилетий. В 50-60 годах прошлого века была широко распространена концепция «исчерпывающего тестирования»: тестировщики стремились проверить все возможные пути выполнения кода на всех возможных данных. Рабочий в теории, на практике этот подход оказался практически неосуществим; разработчикам пришлось обратиться к задаче создания и проведения исчерпывающего набора тестов, достаточного для адекватной оценки работоспособности программы.

В 70-е годы появились новые задачи тестирования. Теперь оно заключалось не только в проверке кода, но и в проверке соответствия продукта заявленным свойствам (которые определяются заказчиком и/или технической документацией, устанавливающей общепринятые стандарты). Кроме того, с помощью тестирования начали выявлять условия, в которых программа работает некорректно (другими словами, стали проверять не только «положительные», но и «отрицательные» сценарии).

В 80-х годах тестирование из отдельного, изолированного процесса разработки превратилось в процесс, интегрированный во все циклы создания продукта. Разработчики стали быстрее находить проблемы и оперативнее устранять их, а в некоторых случаях – и предугадывать неполадки еще до их появления. 90-е года ознаменовали начало автоматизации тестирования, а также переход от простого тестирования к более общему процессу, который называется «обеспечение качества».

**Обеспечение качества** - это совокупность мероприятий, охватывающих все технологические этапы разработки, выпуска и эксплуатации ПО и информационных систем, предпринимаемых на разных стадиях жизненного цикла ПО, для обеспечения требуемого уровня качества выпускаемого продукта [6].Этот подход вывел тестирование на качественно новый уровень и дал импульс для появления множества новых технологий и методик тестирования, о некоторых из которых пойдет речь в следующих разделах.

В нулевых годах двадцать первого века автоматизированное тестирование получило уже повсеместное распространение; кроме того, многие разработчики начали ориентироваться в большей степени на пользователей, нежели на технические требования к программам. Более удачными стали считаться продукты, которые позволяли пользователям более быстро и эффективно решать свои задачи. Среди особенностей современного тестирования называют гибкие методологии и гибкое тестирование, глубокую интеграцию с процессом разработки, широкое использование автоматизации, колоссальный набор технологий и инструментальных средств и кросс-функциональность команды [12].

С учетом современных подходов, тестирование ПО можно определить как технику контроля качества продукта, в которую входит комплекс мероприятий по разработке тестов, выполнение тестирования и анализ его результатов. Важно помнить, что этот процесс неотделим от процесса создания продукта; даже роли в современной команде разработчиков зачастую распределяются так, что тестированием в той или иной мере занимается каждый из участников, а собственно разработчики продумывают план и координируют работу всего коллектива. По утверждению исследователей [6],тестирование занимает примерно 40% трудоемкости при разработке программного продукта, из чего следует, что наибольший эффект в снижении трудоемкости может быть получен именно в этой фазе. Как этого добиться? Ответ на этот вопрос я постараюсь дать в конце этого раздела.

**1.1.2 Различные подходы к тестированию**

Перед тем, как приступить к работе, тестировщику необходимо ответить для на себя на несколько вопросов: на каком этапе разработки продукта необходимо начинать тестирование? К каким функциям следует обраться в первую очередь, какие проверить необходимо, а какими можно пренебречь? В каком объеме? Ответы могут быть самыми различными. И с этой точки зрения можно выделить ряд различных подходов к тестированию.

**Реактивный подход** (validation process) заключается в выявлении дефектов. При таком подходе тесты выполняются уже после того, как программное обеспечение было произведено, и их главная функция – нахождение ошибок в коде. С точки зрения исторического развития это самый ранний из подходов; некоторые команды придерживаются его до сих пор, другие же приняли решение подойти к этому вопросу с другой стороны. Альтернативный подход носит название **превентивного или профилактического** (verification process) и заключается в предотвращении дефектов. В этом случае тесты разрабатываются еще до написания программного кода. Эту методику используют, например, тестировщики компании Google. Джеймс Уиттакер, бывший технический директор Google, утверждает, что настоящая цель тестирования – «создать процесс, в котором сложно допустить ошибку» [23]. Избавиться от ошибок полностью невозможно, но можно организовать работу таким образом, чтобы разработчику было сложнее ошибиться.

Таким же образом построена технология RUP (Rational Unified Process): написание тестов начинается на самом раннем этапе проектирования продукта, с момента, когда были определены требования к нему. Тесты проводятся на каждом этапе создания, их результаты анализируются и при необходимости вносятся изменения в саму структуру будущей программы. В настоящее время превентивный подход является более широко распространенным и популярным.

Второй вопрос касается расстановки приоритетов. Что важнее, найти максимально возможное количество ошибок или найти меньше, но более важных для пользователя? Некоторые тестировщики в большей степени ориентируются на «**разрушение**» - то есть разработку такого кода, который проверит в первую очередь негативные сценарии. Программист, как правило, не учитывает возможность возникновения внештатных ситуаций (например, неправильного формата данных, которые вводит пользователей, или каких-либо технических ошибок на сервере) либо учитывает не все. Поэтому проверка подобных «слепых зон» действительно является нелишней. Этот подход иначе называют **исчерпывающим тестированием**.

С другой стороны, негативные сценарии случаются заметно реже, чем позитивные; во всяком случае, сам пользователь, как правило, не настроен ломать программу - напротив, он готов приложить усилия для того, чтобы максимально быстро достичь необходимого ему результата. Программисты, которые при проведении тестирования фокусируются на запросах пользователей, применяют «**созидательный**» подход (**разумное тестирование**): они начинают работу с самых базовых сценариев и самых важных функций. Они определяют эффективность своей работы не по количеству найденных ошибок, а по количеству пропущенных: чем меньше проблем в приоритетных для пользователя сценариях ускользнуло от глаза тестировщика, тем успешнее тестирование [17].

Последний вопрос заключается в степени **вовлеченности пользователей** в процесс тестирования. Некоторые команды ценят роль «живого» тестирования не так высоко; другие стремятся выпустить проект как можно раньше, чтобы дорабатывать и корректировать его согласно отзывам клиентов. Последнего подхода придерживаются, например, в Google. «Мы строим минимально жизнеспособный продукт и выпускаем его в тот момент, когда он может стать полезным как можно большему количеству людей, получаем обратную связь и дальше развиваем продукт итеративно» - пишет Уиттакер в книге «Как тестируют в Google». Данной точки зрения придерживаются и российские специалисты. Максим Рокатанский, сотрудник компании OTUS, в одной из своих статей [16]подчеркивает важность постоянной обратной связь для снижения бизнес-рисков и потенциальных сбоев в работе. Возможность тестировать меньше, но чаще может помочь обеспечить быструю обратную связь и повысить ценность продукта для клиентов.

**1.1.3 Процесс тестирования**

Тестирование – сложный процесс, состоящий из нескольких этапов. Некоторые специалисты представляют его как конечный «маршрут», пройдя который, можно получить исчерпывающую информацию о качестве продукта; другие описывают как цикл, повторяющийся на каждом этапе разработки приложения до тех пор, пока оно не будет выпущено в свет [10]. Подходов к построению процесса тестирования существует немало; я предлагаю рассмотреть лишь несколько, чтобы определить этапы, актуальные для любой задачи, которая может встать перед тестировщиком.

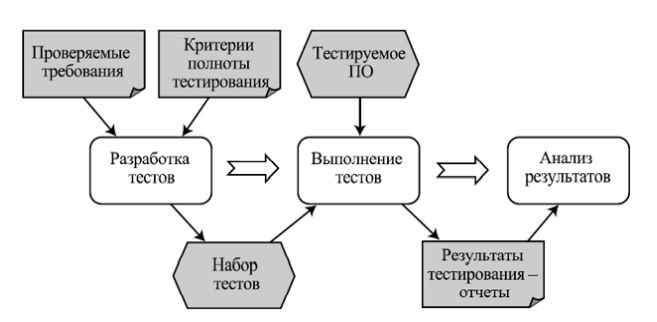
Процесс тестирования можно представить, например, как на *рисунке 1*. Это простая схема, в которой всего три этапа – разработка тестов, выполнение тестов и анализ результатов. Этап планирования здесь как бы вынесен за скобки; частично он реализуется в первом этапе, где тесты пишутся с учетом требований, которые необходимо проверить, и критериев полноты тестирования, то есть условия, что тесты должны покрывать все требования.

Рисунок 1

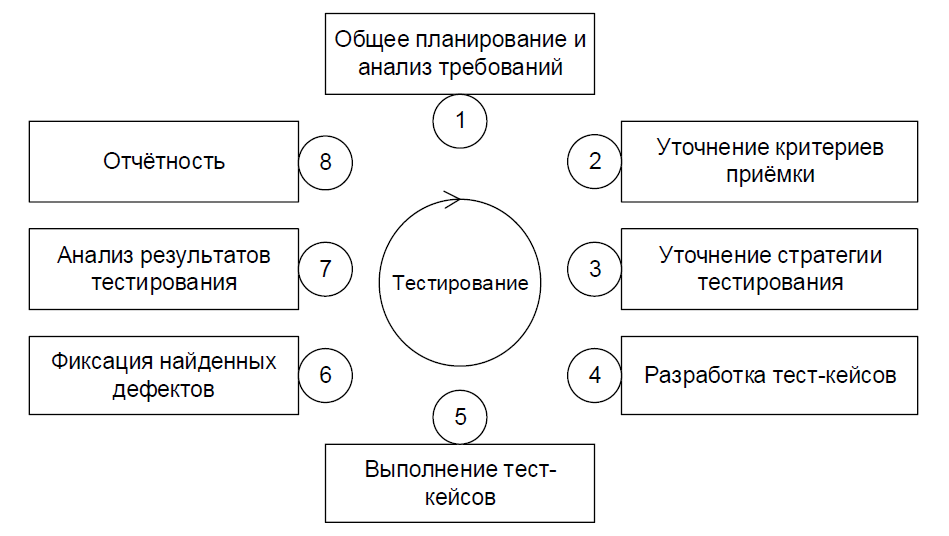
Другие авторы представляют тестирование как более сложный процесс (*рисунок 2*) [12]*.* Здесь первые три стадии посвящены планированию: оценивается цель и объем работы, производится анализ требований, определяются инструменты, формулируются метрики для оценки качества тестирования, а также условия начала и окончания тестирования. На третьей стадии происходит уточнение стратегий тестирования, которые будут применяться на каждом конкретном этапе. Разработка тест-кейсов и тестовых сценариев начинается только на этапе номер четыре, и на каждой итерации тестовый набор подвергается пересмотру и корректировке в соответствии с текущими задачами. Стадии пять и шесть выполняются одновременно: дефекты обнаруживаются и фиксируются прямо по ходу проведения тестов. Последние два этапа также тесно связаны: анализ результатов тестирования производится в письменном виде, и получившийся документ является отчетным документом. Выводы, которые тестировщик получает после прохождения всех восьми этапов, служат основой для стадий 1, 2 и 3 следующей итерации тестирования.

Рисунок 2

**1.1.4 Подготовка к тестированию**

Итак, первым этапом, без которого не обходится ни один проект, является планирование. Формально **планирование** можно определить как непрерывный процесс принятия управленческих решений и методической организации усилий по их реализации с целью обеспечения качества некоторого процесса на протяжении длительного периода времени [12].

Причина, по которой планирование необходимо, довольно проста: команда без плана не способна эффективно функционировать. При составлении плана распределяются обязанности внутри команды; если каждый из ее участников не получил конкретного задания, работа будет выполняться плохо. Плохая работа не позволит выявить все ошибки приложения и найти причины их возникновения. Без этого невозможно будет сделать правильные выводы о том, как их устранить, а значит, нельзя будет составить и качественный отчет о результатах, который укажет направление дальнейшей работы. Таким образом, цикл, который мы рассмотрели в предыдущем разделе, замкнется, не принеся полезных результатов и не имея возможности продолжаться.

Что именно следует учесть при планировании, чтобы избежать подобных проблем? Программные продукты могут быть самыми разными: масштабными или простыми, локальными или ориентированными на международную аудиторию, коммерческими или любительскими. В зависимости от параметров конкретного проекта будут актуальны разные детали; их количество и качество должен определить сам тестировщик. Но есть и несколько общих вопросов:

* Кто будет тестировать? На каких этапах?
* Какие компоненты должны быть протестированы – все или только те, которые угрожают наибольшими потерями для всего проекта?
* Когда и в какие сроки будет проходить тестирование? Будет ли это непрерывный процесс, параллельный созданию продукта, или оно будет выполняться на завершающей стадии разработки?
* Как и что надо проверять – правильность кода, работу функций, удовлетворенность пользователей продуктом?
* В каком объеме нужно тестировать? Ограничены ли ресурсы?

Кроме этого, можно обдумать несколько технических моментов:

* Где у продукта слабые места?
* Есть ли проблемы с надежностью, конфиденциальностью, совместимостью, безопасностью, производительностью, пригодностью использования?
* Работают ли первичные пользовательские сценарии как следует? Подходят ли они для международной аудитории?
* Взаимодействует ли продукт с другими продуктами? Успешно ли?
* Хорошо ли работают средства диагностики проблем?

Только когда определены все особенности и потенциальные проблемы продукта, можно переходить к следующим этапам – в частности, к тест-дизайну.

**1.1.5 Тест-дизайн**

Основным инструментом тестирования программных продуктов являются тест-кейсы. **Тест-кейс** (test case) – это совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или её части [29]. Тест-кейсы объединяются в тестовые наборы (test suit) по какому-либо логическому принципу, например, по функции, которую они проверяют. Важно отметить, что написание тест-кейсов в процессе тест-дизайна происходит в последнюю очередь; до этого тестировщик должен создать несколько проектных документов, которые будут служить для него «путеводной звездой», не дадут отклониться от выбранного подхода или забыть какие-то функции.

Первым из таких документов является **стратегия тестирования**. Это относительно небольшой документ, который описывает базовые подходы к проведению тестирования. Как правило, по ходу работы он не претерпевает изменений, хотя тестировщик может принять решение о переходе к другой стратегии, если посчитает полученные результаты неудовлетворительными.

**Тест-план** (план тестирования) – документ, в котором описывается весь объем работ по тестированию: описание объекта, расписание, критерии начала и окончания тестирования, необходимое в процессе работы оборудования, специальные знания, необходимые тестировщикам, а также производится оценка рисков с вариантами их разрешения. Этот документ является чрезвычайно важным в начале тестирования, однако в дальнейшем довольно быстро теряет актуальность. При разработке продукта могут появляться новые функции или исчезать старые; заказчик может выдвинуть новые требования; могут поступить отзывы от пользователей. Только если своевременно вносить все эти изменения в тест-план, он будет оставаться полезным для команды. Он также должен отвечать ряду требований:

* Он должен быть выполнимым;
* Он должен быть построен таким образом, чтобы при возникновении непредвиденных обстоятельств допускать быстрое изменение в любой из частей без нарушения взаимосвязи с другими;
* Он должен согласовываться с общим проектным планом.

Кроме тест-плана, пишут также **тестовый сценарий**. Он представляет собой последовательность действий над продуктом и сообразных им проверок корректности поведения продукта. Фактически при успешном прохождении всего тестового сценария мы можем сделать вывод о том, что та или иная функция продукта работает корректно.

Еще одним полезным документом является **чек-лист**. Он более компактен и удобен в использовании, чем тест-план, и представляет собой набор идей: по тестированию, по разработке, по планированию и управлению. Чек-листы также можно использовать для самоконтроля: какие функции уже протестированы, к каким нужно обратиться в дальнейшем, с какими возникли трудности. Чек-листы можно создавать, например, для:

* отдельных частей (модулей и подмодулей) приложения;
* отдельных требований, групп требований, уровней и типов требований;
* частей или функций приложения, наиболее подверженных рискам;
* различных уровней функционального тестирования;
* типичных пользовательских сценариев.

После такой всеобъемлющей подготовительной работы можно приступать к написанию тест-кейсов. Здесь тоже есть несколько правил; тест кейс должен быть:

* Показательным, то есть проверять ситуацию, в которой высока вероятность возникновения ошибки;
* Неизбыточным по отношению к другим тест-кейсам – не совершать ту же проверку, что и другой;
* Демонстративным – отклонение от ожидаемых результатов должно быть очевидным;
* Лаконичным, то есть не содержать лишней или очевидной информации;
* Повторяемым – при повторении он должен показывать один и тот же результат;
* Прослеживаемым – информация, содержащаяся в нем, должна прямо указывать на то, что он проверяет.

Стоит также поговорить о техниках тест-дизайна. Их существует довольно много, и выбирать их следует исходя из особенностей проекта и тех деталей, которые необходимо протестировать.

На самом общем уровне различают статическое и динамическое тестирование. При **статическом тестировании** код не выполняется, а проверяется вручную вместе с документами требований и проектными документами на наличие ошибок. Основная цель этого тестирования - повысить качество программных продуктов путем выявления ошибок на ранних этапах цикла разработки. При **динамическом тестировании** же выполняется код. Этот подход призван подтвердить, что программный продукт работает в соответствии с требованиями.

Один из статических методов носит название **предугадывание ошибки** – это метод, при котором тест-аналитик использует свои знания системы и спецификации для того, чтобы «предугадать», при каких входных условиях система может выдать ошибку. Среди динамических выделяют, например, метод **«причина-следствие»**: его смысл заключается в построении графа логических зависимостей между причинами (вводимыми условиями) и следствиями (ответами программы). Такой граф можно затем преобразовать в Таблицу решений (Decision Table), чтобы отслеживать каждую комбинацию причин, приводящих к определенному эффекту.

**Тестирование переходов между состояниями** – еще один динамический метод. С его помощью тестировщик анализирует поведение приложения в зависимости от различных входных условий в последовательности. Метод **Use Case** описывает сценарий взаимодействия двух и более участников (пользователя и системы или двух систем между собой). Целью этого тестирования является нахождение дефектов, которые трудно найти при тестировании модулей по отдельности.

 Для тестирования различных значений, которые может принимать переменная, используются такие техники разбиение на классы эквивалентности, анализ граничных значений и попарный перебор [9].

**1.1.6 Методики тестирования**

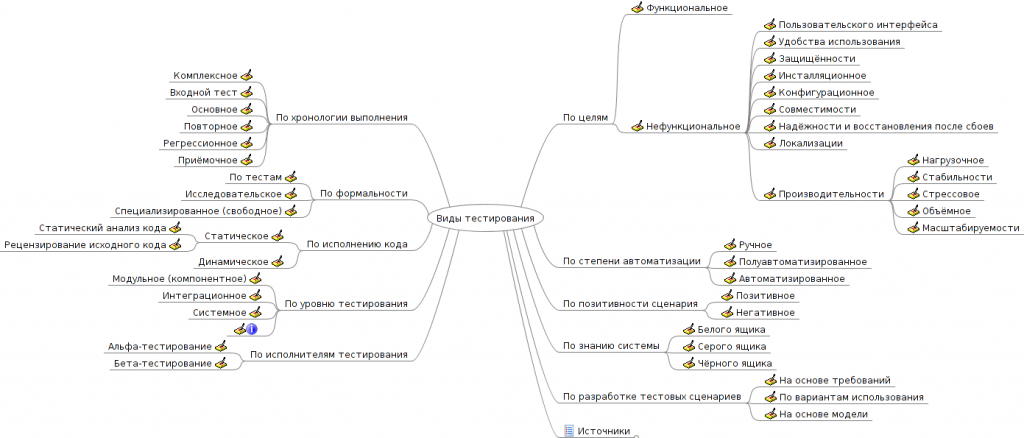
В ходе исторического развития тестирования было создано большое количество методик, позволяющих проверить те или иные свойства продукта. Их настолько много, что охватить все нам не удастся *(рисунок 3)*; я коснусь лишь некоторых, чтобы дать общее представление об инструментарии, которым обладает тестировщик.

Рисунок 3

**Методики по уровню тестирования**

**Модульное тестирование** — уровень тестирования, на котором тестируется минимально возможный для тестирования компонент, например, отдельный класс или функция. Основной недостаток модульного тестирования состоит в том, что проводить его можно, только когда проверяемый элемент программы уже разработан. Снизить влияние этого ограничения можно, подготавливая тесты (а это — наиболее трудоемкая часть тестирования) на основе требований заранее, когда исходного кода еще нет [29].

**Интеграционное тестирование** – уровень тестирования, на котором отдельные программные модули объединяются и тестируются в группе. Интеграционное тестирование в качестве входных данных использует модули, над которыми было проведено модульное тестирование, группирует их в более крупные множества, выполняет тесты, определённые в плане тестирования для этих множеств, и представляет их в качестве выходных данных и входных для последующего системного тестирования.

**Системное тестирование** – это тестирование программного обеспечения, выполняемое на полной, интегрированной системе, с целью проверки соответствия системы исходным требованиям.

**Методики по степени автоматизации**

**Ручное тестирование** — тестирование, в котором тест-кейсы выполняются человеком вручную без использования средств автоматизации. Несмотря на то что это звучит очень просто, от тестировщика в те или иные моменты времени требуются такие качества, как терпеливость, наблюдательность, креативность, умение ставить нестандартные эксперименты, а также умение видеть и понимать, что происходит «внутри системы», т.е. как внешние воздействия на приложение трансформируются в его внутренние процессы.

**Автоматизированное тестирование** - это процесс верификации программного обеспечения, при котором основные функции и шаги теста, такие как запуск, инициализация, выполнение, анализ и выдача результата, выполняются автоматически при помощи инструментов для автоматизированного тестирования [33]*.*

**Методики по позитивности сценария**

**Позитивное тестирование** направлено на исследование приложения в ситуации, когда все действия выполняются строго по инструкции без каких бы то ни было ошибок, отклонений, ввода неверных данных и т.д. Если позитивные тест-кейсы завершаются ошибками, это тревожный признак — приложение работает неверно даже в идеальных условиях (и можно предположить, что в неидеальных условиях оно работает ещё хуже). Для ускорения тестирования несколько позитивных тест-кейсов можно объединять (например, перед отправкой заполнить все поля формы верными значениями) — иногда это может усложнить диагностику ошибки, но существенная экономия времени компенсирует этот риск.

**Отрицательное тестирование** - тип тестирования ПО для поиска точек отказа в программном обеспечении, который проверяет систему на обработку исключительных ситуаций, а также проверяет, что вызываемая приложением функция не выполняется при срабатывании валидатора. [25]

**Методики по знанию системы**

Тестирование методом «**черного ящика**», также известное как тестирование, основанное на спецификации, или тестирование поведения – техника тестирования, основанная на работе исключительно с внешними интерфейсами тестируемой системы. Это тестирование, не предполагающее знания внутреннего устройства компонента или системы.

Целью этой техники является поиск ошибок в таких категориях как:

* неправильно реализованные или недостающие функции;
* ошибки интерфейса;
* ошибки в структурах данных или организации доступа к внешним базам данных;
* ошибки поведения или недостаточная производительности системы

Тестирование методом **белого ящика** (также: прозрачного, открытого, стеклянного ящика; основанное на коде или структурное тестирование) – метод тестирования ПО, который предполагает, что внутренняя структура/устройство/реализация системы известны тестировщику. Знание всех особенностей тестируемой программы и ее реализации обязательны для этой техники [6].

Тестирование методом **серого ящика** – метод тестирования ПО, который предполагает комбинацию White Box и Black Box подходов. То есть, внутреннее устройство программы нам известно лишь частично. Предполагается, например, доступ к внутренней структуре и алгоритмам работы ПО для написания максимально эффективных тест-кейсов, но само тестирование проводится с помощью техники черного ящика, то есть, с позиции пользователя.

**Методики по уровню подготовки к тестированию**

**Тестирование на основе тест-кейсов** — формализованный подход, в котором тестирование производится на основе заранее подготовленных тест-кейсов, наборов тест-кейсов и иной документации. Это самый распространённый способ тестирования, который также позволяет достичь максимальной полноты исследования приложения за счёт строгой систематизации процесса, удобства применения метрик и широкого набора выработанных за десятилетия и проверенных на практике рекомендаций [12].

**Исследовательское тестирование** — частично формализованный подход, в рамках которого тестировщик выполняет работу с приложением по выбранному сценарию, который, в свою очередь, дорабатывается в процессе выполнения с целью более полного исследования приложения. Ключевым фактором успеха при выполнении исследовательского тестирования является именно работа по сценарию, а не выполнение разрозненных бездумных операций. Существует даже специальный сценарный подход, называемый сессионным тестированием. В качестве альтернативы сценариям при выборе действий с приложением иногда могут использоваться чек-листы, и тогда этот вид тестирования называют тестированием на основе чек-листов [12].

**Свободное тестирование** – это вид тестирования, который выполняется без подготовки к тестированию продукта, без определения ожидаемых результатов, проектирования тестовых сценариев. Это неформальное, импровизационное тестирование. Оно не требует никакой документации, планирования, процессов, которых следует придерживаться при выполнении тестирования [6].

**Методики по целям тестирования**

**Функциональное тестирование** — вид тестирования, направленный на проверку корректности работы функциональности приложения (корректность реализации функциональных требований). Часто функциональное тестирование ассоциируют с тестированием по методу чёрного ящика, однако и по методу белого ящика вполне можно проверять корректность реализации функциональности.

**Нефункциональное тестирование** — вид тестирования, направленный на проверку нефункциональных особенностей приложения (корректность реализации нефункциональных требований), таких как удобство использования, совместимость, производительность, безопасность и т.д. [12]

**Тестирование производительности** - тестирование, которое проводится с целью определения, как быстро работает система или её часть под определённой нагрузкой.

**Стрессовое тестирование** - обычно используется для понимания пределов пропускной способности приложения. Этот тип тестирования проводится для определения надёжности системы во время экстремальных или диспропорциональных нагрузок и отвечает на вопросы о достаточной производительности системы в случае, если текущая нагрузка сильно превысит ожидаемый максимум.

**Нагрузочное тестирование** - это простейшая форма тестирования производительности. Нагрузочное тестирование обычно проводится для того, чтобы оценить поведение приложения под заданной ожидаемой нагрузкой [25]. Его **цели**:

* + оценка производительности и работоспособности приложения на этапе разработки и передачи в эксплуатацию;
  + оценка производительности и работоспособности приложения на этапе выпуска новых релизов, патчей;
  + оптимизация производительности приложения, включая настройки серверов и оптимизацию кода;
  + подбор соответствующей для данного приложения аппаратной (программной платформы) и конфигурации сервера.

**Тестирование стабильности** проводится с целью убедиться в том, что приложение выдерживает ожидаемую нагрузку в течение длительного времени.

**Тестирование безопасности** - оценка уязвимости программного обеспечения к различным атакам. Тестирование безопасности проверяет фактическую реакцию защитных механизмов, встроенных в систему, на проникновение.

**Тестирование локализации** - многогранная вещь, подразумевающая проверку множества аспектов, связанных с адаптацией продукта для пользователей из других стран.

**Тестирование совместимости** — вид нефункционального тестирования, основной целью которого является проверка корректной работы продукта в определенном окружении.

**1.1.7 Автоматизация тестирования**

В процессе тестирования практически любого продукта, особенно если он велик по объему и многофункционален, встает вопрос об автоматизации. **Автоматизированное тестирование ПО** – это процесс тестирования, при котором запуск, инициализация и выполнение теста, а также анализ и выдача результата выполняются автоматически, с помощью программных инструментов [21].

Автоматизация тестирования – это полноценная разработка ПО. Как и любая другая разработка, она начинается с плана. В нем тестировщик должен прописать, какие средства ему понадобятся (подставные объекты, фреймворки), как будет выглядеть и осуществляться отчетность и мониторинг результатов, а также, тестирование каких функции и в какой последовательности будет переведено в автоматический режим. Планирование сквозного тестирования, то есть проверки последовательной работы всех компонентов программы, стоит проводить в последнюю очередь; в противном случае разработчик окажется привязан к конкретной архитектуре проекта и не сможет изменить стратегию тестирования.

Разумеется, повсеместное применение автоматизации не имеет смысла и даже вредно для проекта, так как приводит к лишним затратам. Однако в некоторых случаях оно, наоборот, позволяет сэкономить значительное количество ресурсов, а также избавиться от рисков, связанных с человеческим фактором. К таким случаям можно отнести:

1. Рутинные операции – к примеру, заполнение форм с большим количеством полей;
2. Длинные операции (интеграционное или системное тестирование);
3. Проверка данных, требующая сложных математических расчетов;
4. Автоматическая подстановка разных значений для проверки валидационных сообщений;
5. Проверка часто используемых функций (сбои работе которых крайне нежелательны);
6. Проверка труднодоступных мест в системе.

Теперь рассмотрим подробнее, как автоматизация может повлиять на ход тестирования. Здесь можно отметить и положительные, и отрицательные стороны. Сначала обратимся к преимуществам автоматизации:

* Быстрота выполнения тестов – в отличие от человека, компьютеру не нужно сверяться с документациями, все это уже сделано до него;
* Исключение «человеческого фактора» - компьютер не пропустит ни один тест и не перепутает результаты;
* Автоматизация отчетов – их не нужно составлять и рассылать вручную;
* Уменьшение затрат на анализ и поддержку – уже написанные скрипты быстрее обрабатывать автоматически, чем вручную;
* Экономия усилий тестировщика и мощностей компьютера (тесты могут выполняться в то время, пока специалист занят другими полезными делами или даже в нерабочее время, когда локальные сети менее загружены).

Впрочем, в некоторых случаях эти преимущества могут обернуться недостатками:

* «Человеческий фактор» иногда бывает полезен. Тестировщик при выполнении теста может обратить внимание на его индивидуальные особенности и провести дополнительные операции, не предусмотренные планом. Программа же не анализирует тесты, поэтому некоторые ошибки останутся ею не замечены;
* Программе также свойственно пропускать мелкие ошибки, особенно связанные с пользовательским интерфейсом, если тест-дизайнер не предусмотрел возможность их возникновения;
* Затраты на разработку довольно велики, так как речь идет фактически о создании программного продукта;
* Затраты на поддержку – хоть они и меньше, чем для ручного тестирования, они будут расти с каждым изменением программы;
* Стоимость самих инструментов автоматизации может быть довольно высока (а бесплатные инструменты зачастую имеют менее богатый функционал).

**1.1.8 Принципы хорошего тестирования**

Количество разнообразных подходов к тестированию велико. Специалисты, решая разные задачи, обращаются то к одному, то к другому в поисках наиболее оптимального варианта. И хотя тестирование – вещь довольно субъективная, в литературе можно найти несколько принципов, актуальных почти для любой задачи.

Так как исчерпывающее тестирование невозможно, задача специалиста в самом общем виде состоит в том, чтобы найти правильную стратегию тестирования. Круг всех возможных функций и вариантов их работы нужно свести к ограниченному количеству, которое человек способен обработать – вручную или автоматически.

Начать мероприятия по тестированию нужно как можно раньше – желательно, еще на стадии проектирования программного продукта. Стратегия предотвращения ошибок во многих случаях работает лучше, чем стратегия проверки «постфактум».

Практика показывает, что большая часть ошибок сосредоточена в небольшом количестве модулей. Из этого следует, что при возникновении ошибки, природу которой не удается понять сразу, следует в первую очередь проверить тот модуль, в котором обнаружилось больше всего дефектов (или тот, в котором встретились похожие дефекты).

Еще одним статическим наблюдением является то, что по мере того, как растет количество обнаруженных ошибок, растет и относительная вероятность того, что в программе есть необнаруженные [6].

В приоритете для разработчика стоят потребности заказчика, а не формальные требования к продукту, поэтому и тестировщик должен оценивать продукт именно с этой позиции.

Стоит также сказать о критерии тестирования, то есть том аспекте работы программы, который мы хотим проверить. Хороший критерий тестирования отвечает следующим требованиям: он достаточен (показывает, что данное множество тестов достаточно для тестирования данного аспекта), полон (для каждой ошибки существует тест, который может ее выявить), надежен (два множества тестов на одних и тех же данных дают непротиворечивые результаты) и легко проверяем [6].

Разделение тестирования на этапы значительно упрощает работу тестировщика. Выполнение задачи по частям, а не целиком, удобнее наблюдать и контролировать. У каждого этапа есть своя отдельная задача; для каждого готовится отдельный набор тестов, который можно выполнять в той среде, которая больше ему подходит. В целом, при таком подходе производительность и эффективность тестирования вырастает.

Важно учитывать интересы потенциальных и реальных пользователей программного обеспечения. Его форма и структура должны согласовываться со свойственными человеку особенностями восприятия, мышления и поведения. Этот аспект приложения специалисту по тестированию также следует подвергнуть оценке [2].

Важную роль в тестировании играет автоматизация. Сотрудники крупных компаний придерживаются мнения, что автоматизировать следует все этапы тестирования, когда это возможно. Только если задача требует человеческого внимания и интуиции, ее нужно поручать человеку. В перспективе перед специалистами стоит задача создать такие инструменты тестирования, которые позволили бы максимально приблизить автоматизацию к человеческим возможностям [23].

Залогом эффективного тестирования является тщательное предварительное планирование. Чем больше экспериментов, прикидок и набросков планов разработчик сделал до начала реализации проекта, тем сильнее будет его уверенность в успехе мероприятий по тестированию. Два самых важных аспекта плана – начало и конец тестирования. Под «началом» я подразумеваю те цели, которые разработчик ставит перед собой; без этого дальнейшее планирование и выполнение тестов невозможно. А для того, чтобы тестирование завершилось, необходимо с самого начала установить критерии, выполнение которых будет означать удовлетворительную работу программы (поскольку исчерпывающее тестирование невозможно и поиск ошибок может продолжаться бесконечно).

В процессе тестирования важно вести подробные записи [15]. Хороший тестировщик ведет заметки о ходе сессии, фиксируя свои наблюдения и описывая свои действия. Детальные заметки помогают ему выявить причинно-следственные связи, сообразить, какие вопросы надо задавать, и наметить план будущих сессий [28]. Если тестировщик был не слишком внимателен или скрупулезен при ведении записей, велика вероятность того, что ему придется проводить одни и те же тесты дважды, чтобы понять природу ошибки.

Последним принципом хорошего тестирования, который я хотела бы упомянуть, является применение креативного или критического мышления на разных этапах тестирования. Этот вопрос поднимает Джон Стивенсон в одной из своих статей [30]; он утверждает, что оба эти вида мышления очень важны для специалиста по тестированию, но необходимы в разной степени в зависимости от задачи, которую он в данный момент решает.

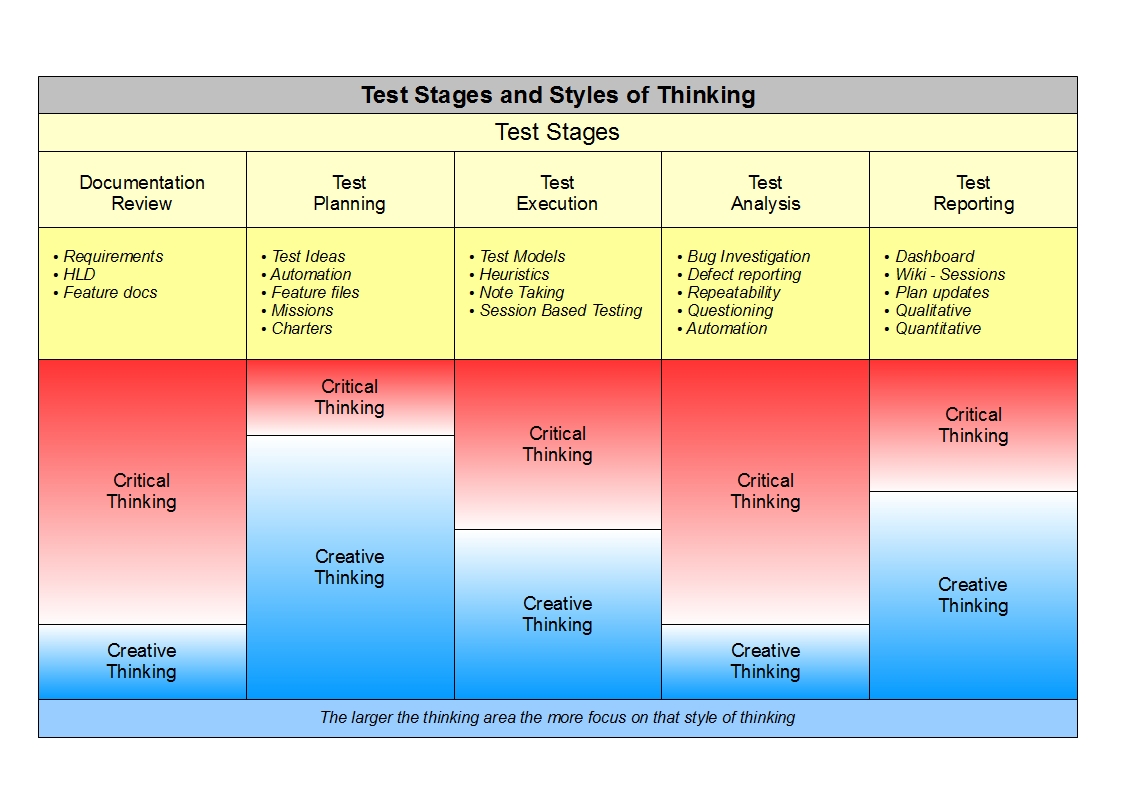
**Критическое мышление** можно определить как интеллектуальный процесс концептуализации, применения, анализа, синтеза и/или оценки информации, полученной из или с помощью наблюдения, опыта, размышления, рассуждения или общения. Проще говоря, это способ мышления, который помогает вам решать, является ли некое утверждение верным, частично верным или неверным, позволяет подвергнуть его сомнению. **Креативное мышление** направлено на генерацию новых идей, способов, подходов, продуктов, – чего-либо, что изменяет или трансформирует существующую область. Как правило, наибольшее внимание уделяется первому способу, но второй не менее важен. Как вы можете видеть из *рисунка 4*, критическое мышление наиболее важно при анализе документации и результатов тестов, а на этапах планирования тестов и подведения итогов тестирования (который включает в себя также представление результатов коллегам и разработку новых стратегий) более важную роль играет креативное мышление. Выбор правильной стратегии в каждом из случаев позволяет тестировщику повысить качество и эффективность своей работы.

Рисунок 4

**1.2 Диалоговые системы**

**1.2.1 Определение диалоговой системы и ее актуальность в современном мире**

Диалоговые системы известны под разными названиями: диалоговые интерфейсы, интерфейсы естественного языка, интеллектуальные виртуальные агенты, виртуальные помощники, чат-боты. Данные понятия не являются полностью взаимозаменяемыми, а программные продукты, к которым они относятся, различаются сложностью, объемом задач и способами взаимодействия с пользователями.

В самом общем смысле **диалоговой системой** называется программное приложение, имитирующее поведение человека и предназначенное для общения с пользователем с помощью текста или голоса; кроме собственно коммуникации, чат-бот может выполнять такие задачи, как поиск и обработка информации, вызов такси, запись на прием к врачу, а также помогать в изучении иностранных языков и даже играть в игры. Распознавание запросов может производиться по шаблонам или с помощью модулей распознавания речи, а ответные реплики выбираются ботом из готового набора либо генерируются. Некоторые боты умеют «самообучаться», запоминая ответы пользователя и используя их при генерации речи [8].

Интерес к чат-ботам неуклонно растет. Ученые начали заниматься их разработкой еще в 60-х годах XX века, но именно в последнее десятилетие диалоговые интерфейсы по-настоящему захватили умы человечества. Срини Джанарсанам, крупный специалист в данной области и автор учебного пособия «Разработка чат-ботов и разговорных интерфейсов», отмечает всплеск интересов и огромные инвестиции, вкладываемые в создание цифровых сущностей, способных разговаривать на естественном языке. Технологии диалогового пользовательского интерфейса стали одной из главных тенденций в технологическом бизнесе; многие крупные бренды, заинтересованные в автоматизированном решении повторяющихся или однотипных бизнес-задач, делают выбор в пользу чат-ботов [3].

Данные утверждения подтверждаются на практике. По данным компании Gartner, доход брендов, внедривших в свои веб-сайты возможность голосового и визуального поиска, за несколько лет вырос на 30%. Кроме того, Gartner прогнозирует, что больше половины всех существующих компаний будут вкладывать больше средств в разработку чат-ботов, нежели традиционных мобильных приложений.

Почему же представители бизнеса так заинтересованы в диалоговых системах? За этим кроется две причины. Первая – **рост популярности ботов у пользователей**. Чат-боты, как правило, не существуют в виде отдельных приложений, а внедряются в мессенджеры. Последние по количеству пользователей уже обогнали социальные сети.По статистике, сегодня ими пользуются 2,5 млрд человек [11]. Возможно, это связано с тем, что люди устали от количества контента, который предлагают и даже навязывают соцсети, и сделали выбор в пользу более минималистичного интерфейса [14]. Классические приложения (которые, например, сообщают прогноз погоды или сравнивают цены в разных магазинах) скачиваются все реже: они требуют трафика для скачивания и времени для установки, занимают место в памяти и на экране смартфона. На их фоне мессенджеры и внедренные в них чат-боты выгодно выделяются. Один из разработчиков ботов для Telegram утверждает, что «боты могут и должны заменить функционал 80% существующих мобильных приложений» [20].Боты популярны и в бизнес-задачах: согласно опросам клиентов, для связи с предприятиями люди чаще предпочитают каналы веб-чата, чем другие традиционные каналы (электронную почту или телефон) [3].

Вторая причина заключается в **выгодах**, которые чат-боты предлагают компаниям. Диалоговые агенты, выполняющие роль консультанта из службы поддержки, доступны пользователям в любое время дня и ночи, предоставляют персонализированный подход в зависимости от того, какая информация о пользователе содержится в базе, работают непрерывно и стабильно, не заставляют пользователя ждать ответа, а также стоят гораздо меньше, чем штат сотрудников, выполняющих ту же задачу. Более того, разработка чат-ботов в последние годы стала проще и дешевле в силу того, что появились новые инструменты для управления диалогами, распознавания и синтеза речи, аналитики и других важных задач [3].

Все это обуславливает пристальное внимание, направленное сейчас на диалоговые интерфейсы, и актуальность исследований в этой области.

**1.2.2 Функции диалоговых систем**

Функции, которые может выполнять диалоговая система, крайне разнообразны. Единой классификации чат-ботов по задачам не существует. В самом общем виде их можно разделить на две категории: корпоративные помощники и персональные помощники [3]*.*

**Корпоративные помощники** выполняют функции представителей службы поддержки и продавцов-консультантов. Большинство ботов, развернутых в мессенджерах, принадлежат именно к этой категории. **Персональные помощники** (например, Siri от Google или Алиса от Яндекс) выполняют такие задачи, как прием вызовов и отправка текстов, внесение напоминаний в календарь, воспроизведение музыки и видео, вызов такси и многие другие. Они при общении подстраиваются под конкретного пользователя, а пользователь, в свою очередь, может настроить диалоговый интерфейс так, как удобно ему, создавая для него новые навыки и развивая существующие.

При более подробной классификации можно выделить четыре типа чат-ботов [22]:

* Боты, выполняющие простые, **рутинные операции**;
* Боты для **сбора данных**, аналитики, а также распространения корпоративной информации среди сотрудников;
* Боты-**помощники**, выступающие в качестве консультантов, ассистентов, работников службы поддержки;
* Боты для **развлечения**, общения, игр.

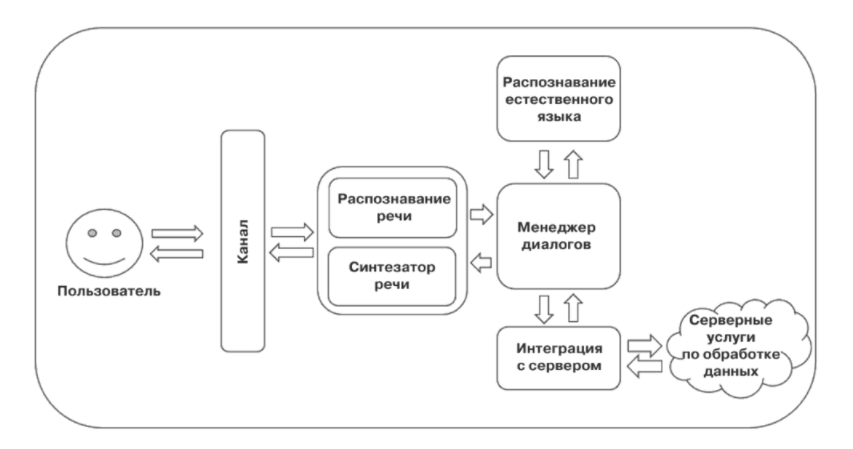
Б. Шауар и Э. Этвел, специалисты в области диалоговых систем, выделяют другие **направления использования ботов** [24]:

1. В сфере **бизнеса** и коммерции чат-боты могут решать такие задачи как онлайн-консультирование, продажи, рекламные коммуникации и т.п.
2. В сфере **получения информации** чат-боты используются для предоставления пользователям интересной им информации. Например, диалоговый агент может сообщать новости по темам, которые интересны человеку, или рекомендовать ему кино в соответствии с его предпочтениями.
3. Боты также используются в сфере **обучения иностранным языкам**. Они предоставляют учебный материал (тексты или слова для заучивания) по дисциплине и оценивают усвоение знаний.
4. В сфере **развлечений** возможности виртуальных помощников также велики. Чат-боты могут вести диалог с пользователем, играть с ним, включать музыку.

Возможности диалоговых агентов продолжают расти по мере появления новых технических возможностей [19]. Они приобретают не только те функции, которые выполняются другими существующими приложениями, но и уникальные, казавшиеся невозможными ранее. Например, в сфере СМИ сейчас активно разрабатываются боты, которые могли бы анализировать массивы данных и писать полноценные журналистские тексты. Появляется новая сфера – автоматизированная журналистика, задачей которой является сбор информации, ее качественная интерпретация и передача [7].

**1.2.3 Архитектура диалоговой системы и инструменты для ее разработки**

Диалоговая система состоит из нескольких модулей *(рисунок 5)*. Главным является **менеджер диалогов** – модуль, который контролирует поток разговора. Он получает запрос пользователя, интерпретирует его и решает, каким должен быть ответ системы.

*****Рисунок 5*

Сам менеджер диалогов состоит из трех компонентов: **сценария** или пути разговора (Dialog Flow), **модели классификации сообщений** (ML Classification model), которая определяет намерения пользователей, и **модели для извлечения данных** из сообщений (ML Information Extraction model) [18]. Для написания сценария существует множество инструментов, различающихся по функционалу и назначению. Chatfuel и ManyChat позволяют визуализировать диалог; для текстового представления контента существуют языки сценариев, такие как AIML, ChatScript и RiveScript. Если разработчик планирует добавить в чат-бот модуль распознавания естественного языка, ему подойдет приложение Dialogflow. В качестве альтернативы существуют библиотеки SDK, которые можно использовать для создания чатов: MS Bot Builder, BotKit, BotFuel и т. д.

Если диалоговому агенту необходимо брать информацию из сторонних источников, в сценарий добавляется код для взаимодействия с **серверными модулями** – базами данных, сайтами. В качестве серверного модуля может также выступать онлайн-служба для выполнения инструкций пользователя (например, для бронирования билетов).

В процессе разработки могут пригодиться инструменты **прототипирования**: они позволяют продемонстрировать клиентам графический интерфейс и поведение бота. Такие платформы как BotSociety.io и BotMock.com дают возможность визуализировать беседу между программой и пользователем еще на стадии проектирования, чтобы разработчик мог получить отзывы потенциальных клиентов и при необходимости внести правки [3].

Еще одним важным элементом архитектуры бота является **канал** – среда, в которой пользователь может с ним общаться. Существуют разные виды каналов, причем диалоговую систему можно подключать сразу к нескольким:

* Службы обмена сообщениями (Telegram, WhatsApp);
* Социальные сети (ВКонтакте, Facebook);
* Офисные и командные чаты (Microsoft Teams);
* Традиционные каналы – веб-чат, голосовые звонки, SMS;
* Умные колонки (Google Home, Яндекс Станция).

Выбор платформы зависит от целевой аудитории чат-бота. Молодые люди часто и охотно пользуются социальными сетями и мессенджерами. Согласно опросу ВЦИОМ, 91% людей в возрасте от 18 до 24 лет ежедневно заходят в соцсети; у граждан старше 60 лет службы обмена сообщениями гораздо менее популярны – только 15% опрошенных регулярно общаются с помощью чатов [14]. Исходя из этого, в качестве платформы для бота, ориентированного на молодых людей, стоит использовать мессенджер или соцсеть, а для общения с пожилыми людьми хорошим решением будут голосовые звонки.

Для улучшения качества диалогового интерфейса используют **инструменты аналитики**. Они сохраняют и анализируют входящие и исходящие сообщения, сообщают о типах разговоров, которые происходят между программой и пользователями, и проблемах, с которыми последние сталкиваются. Информация о том, какие ветки диалога являются самыми популярными, какие высказывания пользователей бот не смог распознать, а какие распознал неверно, и в каких местах диалог прерывается, очень полезна для разработчика, который хочет продолжать совершенствовать свою диалоговую систему и после релиза.

Последним важным элементом, который уже не относится ни к непосредственно архитектуре чат-бота, ни к его разработке, является **каталог**, который призван помочь пользователю найти то, что ему нужно. Такие каталоги как BotFinder, BotPages и ChatBottle делят диалоговые интерфейсы на категории по нескольким параметрам: назначению, языку, стране-разработчику и другим. В мессенджерах Facebook и Telegram также есть каталоги ботов, размещенных в них [3].

**1.2.4 Рекомендации по разработке диалоговой системы**

Центральным принципом работы любого разговорного агента является имитация живого диалога. В связи с этим авторы многих пособий по созданию чат-ботов отмечают, что помимо технических требований такая программа должна удовлетворять ожиданиям пользователя в том, что касается естественности общения. При создании диалогового агента следует обратить внимание на несколько аспектов [8]:

* **Порядок обмена репликами**: человек, как правило, интуитивно понимает, когда ему следует вступить в диалог; для бота же (особенно если он развернут в голосовом канале связи) нужно прописать правила, когда и сколько следует ожидать ответа от пользователя, когда нужно повторить реплику и т.д.
* **Общий контекст собеседников**: участники диалога всегда имеют некие общие знания о мире (то, что в английском языке обозначается термином common ground), а также они запоминают информацию, которой обмениваются в процессе общения. Чат-бот тоже должен уметь запоминать данные, полученные от пользователя, а кроме того (что гораздо сложнее в реализации), обладать «кругозором», хотя бы в той сфере, в которой он специализируется.
* **Тема и контекст беседы**: люди при общении всегда понимают, о чем идет речь и когда тема диалога меняется; для бота также очень важно это отслеживать, чтобы правильно интерпретировать реплики собеседника. Это достигается с помощью прописывания стейтов (состояний), в которых бот может находиться, и правил перехода из одного стейта в другой.
* **Структура диалога**: все естественные диалоги строятся по определенным схемам и правилам, которые люди интуитивно знают и применяют в соответствии с ситуацией, в которой находятся. Самым простым примером такого правила является то, что любой диалог принято начинать с приветствия, а заканчивать прощанием. Сценарий для чат-бота следует прописывать в соответствии с этими правилами.
* **Инициатива в диалоге**: в некоторых ситуациях направление диалога выбирает один человек, в других инициатива переходит от одного к другому. Создатель чат-бота сам выбирает, машина или человек будет задавать вектор диалога, но более простым в реализации является сценарий, при котором ведущим является бот.

Кроме естественности диалога, разработчику следует позаботиться и о **дружественности интерфейса**: пользователю должно быть легко и приятно общаться с чат-ботом [1]. Здесь тоже есть несколько факторов:

* **Гибкость диалога**, то есть способность системы подстраиваться под различные особенности, потребности и уровень квалификации пользователей;
* **Ясность поведения системы** – пользователь должен понимать, о чем бот говорит, чего от него хочет и что может предложить. Будет хорошо, если бот в начале беседы расскажет человеку о своих возможностях (текстом или в виде кнопок);
* **Простота пользования** – человеку должно быть интуитивно понятно, как с помощью бота добиться желаемого результата;
* **Соблюдение мер вежливости** и других норм человеческого общения;
* **Доступность системы** в любой момент времени;
* **Обеспечение идентификации и защиты данных**;
* Способность адекватно **действовать в нештатных ситуациях**.

Если все эти рекомендации будут учтены, а разработчик сценария не забудет поставить себя на место пользователя, у клиентов не возникнет проблем при общении с ботом.

**1.2.5 Компания Just AI и платформа JAICP**

Тема данной дипломной работы была предложена компанией **Just AI**, поэтому практическая часть – написание и тестирование чат-бота – будет выполняться на одной из разработанных ею платформ, JAICP. Компания занимается разработкой диалоговых интерфейсов для различных задач, выпускает программные продукты для создания чат-ботов и навыков для умных устройств и ассистентов, а также организует образовательные вебинары и конференции, чтобы развивать данную отрасль.

Среди инструментов, разработанных в Just AI [26]:

* JAICF – открытый бесплатный фреймворк для разработки голосовых навыков и чатботов на базе Kotlin;
* Aimylogic – no-code конструктор сценариев;
* JAICP – платформа для реализации больших и сложных проектов;
* SDK Aimybox – бесплатная платформа с открытым кодом для создания голосового ассистента для приложений и устройств;
* CAILA – сервис понимания естественного языка, используемый во всех разработках компании.

**Just AI Conversational Platform Ultimate** (JAICP Ultimate) – платформа для разработки, внедрения, эксплуатации и сопровождения диалоговых систем и голосовых ассистентов. Для распознавания запросов пользователей используются как подходы, основанные на правилах, так и алгоритмы машинного обучения. Платформа доступна по нескольким тарифам, в том числе и для некоммерческого использования, поэтому подходит для учебных целей [4].

JAICP подходит для проектов любой сложности; с ее помощью можно создать такие продукты как:

* Голосовой ассистент для контакт-центров;
* Боты для звонков внутри компании;
* Бот-рекрутер для отбора персонала;
* Голосовой помощник на сайте;
* Разговорные навыки для умных колонок;
* Бот-консультант для мессенджера;
* Голосовые игры для различных устройств.

JAICP состоит из нескольких элементов. Основной из них – **редактор**, который содержит в себе директорию со всеми файлами проекта и окно для их отображения и редактуры. Минимальный проект чат-бота состоит из трех файлов: конфигурационного файла, который по умолчанию называется «chatbot.yaml», файла с основным сценарием работы бота («main.sc») и файла с тестами. В проекте может быть несколько сценариев для разных тем, несколько тестовых файлов, а также файлы с кодом на JavaScript, справочники именованных сущностей в формате csv и справочники примеров запросов (для машинного обучения). Все файлы «подключаются» к проекту через основной файл сценария [27].

**Сценарий** чат-бота пишется на языке **JAICP DSL**, также разработанном в компании. Файл сценария имеет древовидную структуру, вложенность стейтов задается с помощью отступов. Для описания намерений собеседника используются **теги NLU**, структура бота и правила перехода между состояниями задаются с помощью **декларативных тегов**, поведение бота описывается **тегами реакций**. Помимо текста, диалоговый агент может общаться с пользователем с помощью кнопок-подсказок, включить ему аудиозапись или показать картинку. Есть возможность добавлять и свои собственные теги.

Язык JAICP DSL использует вставки JavaScript-кода: функции-условия, скрипты инициализации, а также некоторые встроенные переменные, функции и библиотеки. Фрагменты кода можно прописывать непосредственно в сценарии, а можно вынести в отдельный файл.

**Механизмы NLU** – понимания естественного языка – содержатся в другом разделе платформы. NLU-ядро **CAILA** подключается в настройках проекта; доступно несколько алгоритмов классификатора – с помощью функции похожести данных (STS), с помощью классического машинного обучения (Classic ML) или на основе глубоких нейронных сетей (Deep Learning). В качестве обучающего материала используются **интенты** (намерения пользователей) и **сущности** (имена, названия, номера телефонов и т.п.), встроенные по умолчанию или добавленные разработчиком. Задача классификатора состоит в том, чтобы вычленить из речи пользователя эти элементы. Распознавание запросов можно осуществлять и без CAILA, для этого нужно прописать паттерны и сущности в отдельных файлах и подключить их к проекту.

Еще один раздел платформы – **каналы** – посвящен подключению диалоговых систем к различным каналам. Каналы бывают четырех типов:

* входящие
* телефонные
* операторские
* интеграции.

К **входящим** относятся текстовые и гибридные каналы, где клиенты могут общаться с ботом: голосовые ассистенты, мессенджеры, социальные сети, различные платформы и сервисы - всего более 20 каналов. Также доступен API для интеграции с приложениями и чат-виджет для размещения на сайте. В **телефонных** каналах пользователи могут общаться с ботами, а в **операторских** – с живыми операторами; если активны оба этих канала, можно осуществлять переключение между ботом и операторами. Из внешних систем для **интеграции** доступны только Google Таблицы. С помощью бота в них можно читать и записывать информацию.

На платформе JAICP можно проводить тестирование бота в ручном или автоматическом режиме. Для ручного тестирования есть **чатвиджет**; автоматическое происходит при запуске бота при условии, что к конфигурационному файлу подключен файл с прописанными тест-кейсами. Публикация бота в канал возможна только после успешного прохождения всех тест-кейсов.

В окне редактора есть возможность посмотреть **логи** – записи диалогов, произошедших в процессе тестирования или работы диалогового ассистента с клиентами. Из логов можно извлекать информацию, оценивать качество работы бота и вычленять нераспознанные фразы. На них диалогового агента можно дообучать, чтобы уменьшить количество ошибок в будущем.

Несмотря на то, что платформа устроена достаточно сложно, научиться ею пользоваться можно довольно быстро благодаря тому, что создатели написали к ней подробную документацию с примерами и возможностью поиска по сайту, а также записали видеокурс, в котором пошагово рассказали обо всех этапах создания чат-бота.

**1.2.6 Тестирование диалоговых систем**

Диалоговые системы, использующие алгоритмы искусственного интеллекта, стали разрабатываться не так давно, поэтому общепринятых методов и международных стандартов для оценки качества их работы на сегодняшний день не существует. Создатели чат-ботов вынуждены самостоятельно подбирать методики, которые соответствовали бы всем функциям и параметрам их продукта. Однако разговорные агенты имеют ряд общих особенностей, которые можно учесть при тестировании.

Чат-бот представляет собой не просто программу; он нередко используется в коммерческих целях, заменяя продавцов-консультантов и сотрудник колл-центров, а значит, к нему применимы KPI (Key Performance Indicator) метрики, которые используются для оценки деловой активности сотрудников организаций. При этом, поскольку чат-бот – программный продукт, стандартные методы оценки ПО для него так же актуальны [18].

Оценка бизнес-показателей важна в первую очередь владельцам компаний, поскольку помогает определить, как быстро окупятся затраты на разработку бота и выполняет ли он все заявленные задачи. К KPI метрикам относятся:

* Изменение количества продаж
* Уменьшение нагрузок на колл-центры
* Количество новых и вернувшихся пользователей
* NPS – оценка потребительской лояльности (чат-бот – альтернативный канал для проведения опросов)

Что касается специфических технических показателей чат-бота, специалисты по тестированию предлагают два показателя:

* Оценка Dialog Flow: какой процент ответов чат-бота оказался правильным (уместным);
* Оценка качества извлечения данных: какой процент ценных данных боту удалось выделить в речи пользователя [18].

На практике оба эти показатели можно определить с помощью тест-кейсов и живого пользовательского тестирования, поскольку прохождение теста и оценка пользователя напрямую зависят от того, были ли ответы бота правильными и смог ли он воспользоваться информацией, полученной от собеседника.

Изучив существующие методики оценки программных продуктов, я собрала набор, подходящий для тестирования диалоговых интерфейсов *(рисунок 6)*.

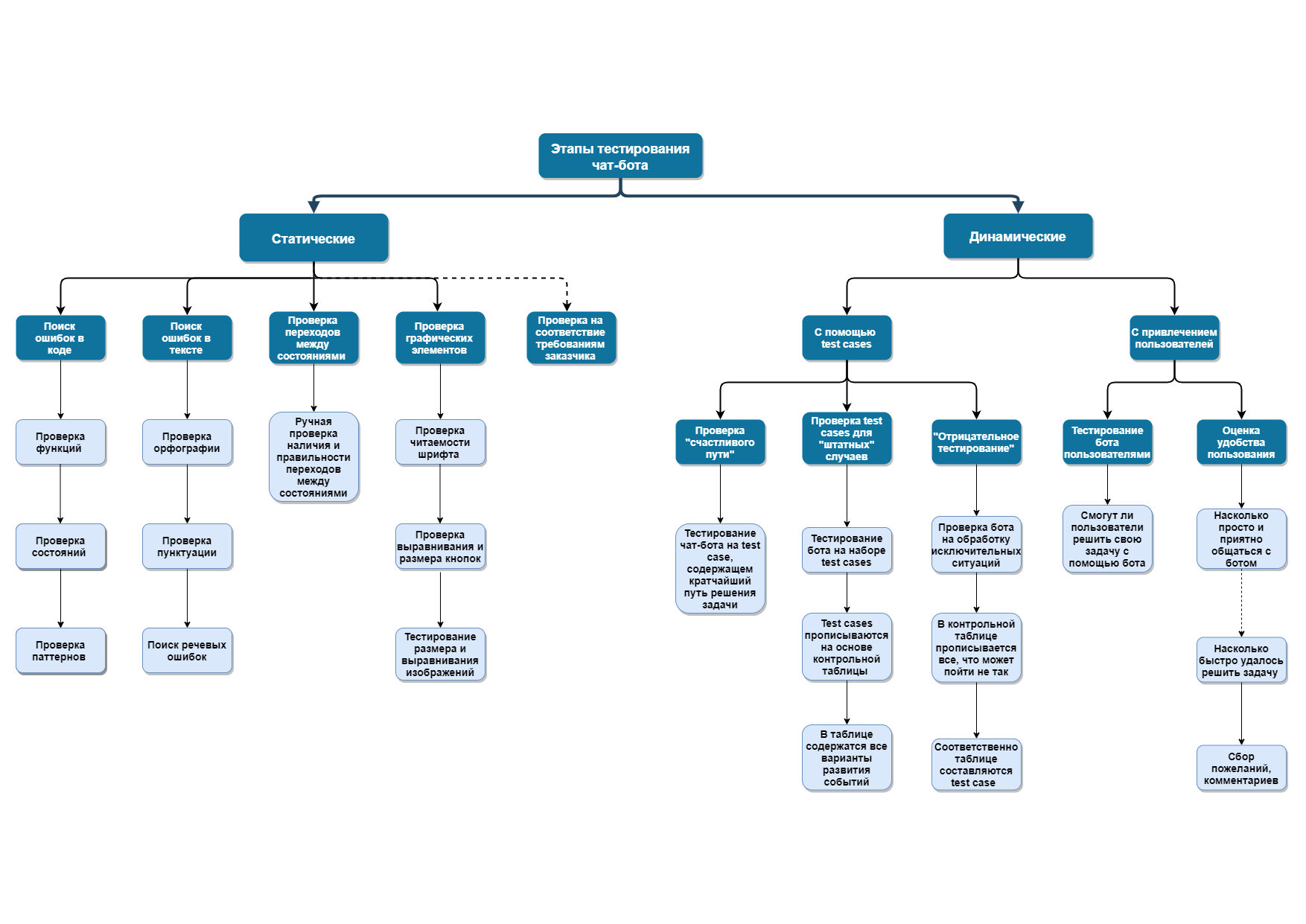


Рисунок 6

Первая группа тестов – **статическая** – выполняется вручную. Код и текстовые фрагменты проверяются на наличие ошибок; особое внимание должно быть уделено переходам между состояниями, так как именно в этой части бота зачастую кроется причина большей части неполадок. Важно также оценить продукт «глазами пользователя»: читаем ли шрифт, отображаются ли кнопки, какого размера картинки и выровнены ли они по ширине текста. Все эти действия призваны сократить количество ошибок, которые будут обнаружены непосредственно при живом тестировании бота, чтобы тестировщик смог сосредоточиться на структуре диалоговой системы, а не на технических моментах ее устройства. Если у чат-бота есть заказчик, нужно дополнительно оценить соответствие продукта его требованиям.

**Динамическое** тестирование выполняется с помощью test cases. Тесты прописываются так, чтобы покрыть максимальное количество возможных случаев. Отдельный набор нужно составить из тестовых диалогов, которые проверяют бот на обработку исключительных ситуаций: например, когда пользователь дает ему неверную информацию или произносит фразу, которую бот не может понять. Последним этапом является тестирование бота пользователями. Возможно, остались пути развития диалога, которые мы не учли; поэтому на этом этапе проверки также могут возникнуть ошибки. Наконец, важно собрать отзывы пользователей о боте: насколько просто, удобно и комфортно было с ним общаться, удалось ли им получить какой-то результат, есть ли у них какие-либо замечания или пожелания для разработчиков.

Следующий раздел будет посвящен созданию чат-бота и проверке на нем данных методик тестирования.

**2. Проверка методик тестирования на диалоговой системе**

**2.1 Описание чат-бота**

**2.1.1 Выбор тематической области для чат-бота**

Для проведения эксперимента я решила создать диалоговую систему, которая выполняла бы роль справочника – отвечала бы на вопросы пользователей по какой-либо теме. Это решение продиктовано в первую очередь моим желанием разработать чат-бот, который был бы полезен людям. Разговорный агент, который играет роль оператора в кафе или банке, возможно, был бы проще в исполнении, но совершенно лишен перспективы быть доработанным и найти настоящих клиентов. Моим намерением было создать разработку, которая, даже будучи созданной в учебных целях, могла бы приносить пользу людям.

Кроме того, архитектура вопросно-ответного бота довольно проста. Он не предполагает длинных разветвленных сценариев, большинство состояний находятся на одном уровне и лишь иногда «углубляются» наводящими вопросами. В его создании таится другая сложность: для того, чтобы охватить предметную область, даже очень узкую, необходимо прописать довольно много состояний. С другой стороны, в посвящении чат-бота конкретной теме есть и положительный аспект: для него не нужно прописывать много стейтов «не по делу», для поддержания диалога – все-таки у оператора колл-центра нечасто спрашивают, как дела, и интересуются его личностью.

Выбор предметной области для диалоговой системы был обусловлен несколькими факторами. Эта должна была быть область, в которой я обладаю достаточным количеством знаний; информация по которой находится в открытом доступе; и которая была бы интересна достаточно большому кругу людей. Исходя из этого, в качестве темы для чат-бота я выбрала правила настольной ролевой игры Dungeons&Dragons (DnD, D&D, «Подземелья и Драконы»). Эта игра довольно популярна в России, поэтому бот сможет найти своих пользователей; кроме того, я довольно хорошо знакома с ее правилами, поэтому могу учесть большинство разделов, нюансов и сложностей, которые они затрагивают.

**2.1.2 Справочник о правилах DnD**

Dungeons&Dragons появилась в 1974 году в США и довольно быстро распространилась по миру. В DnD нельзя выиграть или проиграть; целью игры является хорошо провести время с друзьями, попутно придумав и рассказав историю. Группа игроков (party) во главе с ведущим (гейм-мастер, Dungeon Master, DM) придумывают себе персонажей и действуют от их лица в заданных условиях. Ведет повествование и отыгрывает всех неигровых персонажей гейм-мастер. Сеттинг, в котором происходит действие – как правило, фентезийный мир, но игроки могут выбрать любую локацию и временной период. Истории, которые «проживают» игроки, мастер может выбрать из архива готовых или придумать самостоятельно. Игра в Dungeons&Dragons вдохновила нескольких авторов на написание полноценных литературных произведений – так появились, например, романы цикла «Сага о копье» Маргарет Уэйс и Трейси Хикман и несколько циклов Роберта Сальваторе.

Поскольку игрокам дана полная свобода действий в ведении истории и отыгрыше персонажей, создателям DnD было необходимо ввести обширную и разветвленную систему правил. Герои могут сражаться с противниками, помогать или мешать друг другу, колдовать, мастерить вещи, вскрывать замки, пугать, очаровывать, обманывать, покупать и продавать, решать головоломки, ориентироваться на местности, путешествовать, вспоминать факты об истории мира, приручать животных, молиться богам, и многое другое. Каждое из действий описывается правилами, которые содержатся в Книге Игрока (Player’s Handbook) и Руководстве Мастера (Dungeon Master’s Guide). В каждом из этих пособий более трехсот страниц. Даже для опытного игрока в этой огромной системе правил есть белые пятна, а новичка она и вовсе может отпугнуть.

Хорошим подспорьем для начинающего игрока являются более опытные товарищи по команде и гейм-мастер, которые в ходе истории могут подсказать, что и как нужно делать. Однако может случиться и другая ситуация: представим, что группа друзей без какого-либо игрового опыта решила освоить DnD. Даже прочитав книгу правил от корки до корки, они не смогут приступить к игре, поскольку запомнить столько информации за один раз невозможно. Во время партии им придется обращаться к книге раз за разом, что может испортить впечатления от игры, превратить ее из развлечения в рутину. Для таких игроков всезнающий бот может стать отличным решением – он всегда под рукой, содержит ту же информацию, что и Книга Игрока (а может, и больше), и мгновенно отвечает на вопросы. Кроме того, бот не умеет укоризненно смотреть, вздыхать и иронизировать, а значит, ему нестрашно задать любой, даже самый глупый вопрос.

**2.2 Устройство чат-бота**

Для того, чтобы пользователям было приятнее общаться с чат-ботом, я дала ему имя – Ион. Бот начинает диалог с приветствия и описания своих возможностей, а затем предлагает задать вопрос. Он обращается к собеседнику на «ты», поскольку мне показалось уместным создать неформальную обстановку, атмосферу дружеской компании, в которой бот является одним из членов команды. Диалоговый агент не отвечает на вопросы личного характера, хотя я рассматриваю возможность в будущем придумать для него полноценный образ – манеру речи, историю создания, мнение по каким-либо вопросам, – чтобы игроки чувствовали себя с ним еще более комфортно.

Поскольку охватить все правила ДнД – задача очень сложная и трудоемкая, мой чат-бот отвечает только на вопросы о действиях, которые можно совершать во время сражения. Эти вопросы разбиты на десять категорий, в каждая из которых содержит несколько более узких тем. Список категорий вы можете видеть ниже; список всех тем содержится в *приложении 2* (их общее количество - 48).

1. В целом о сражении
2. Атака
3. Другие виды действий
4. Бонусное действие
5. Перемещение
6. Укрытие
7. Урон
8. Лечение
9. Очередность ходов
10. Состояния

В каждой из тем содержится как минимум один запрос (состояние бота, стейт). Всего получилось 148 различных запросов, для каждого из которых есть как минимум два примера вопросов пользователей. Они носят цифро-буквенные названия (например, D4.2) где заглавная буква отсылает к разделу, первая цифра – к теме, а вторая маркирует конкретный запрос. Некоторые стейты содержат запросы, уточняющие ответ бота, и являются дочерними, то есть такими, попасть в которые можно только из их родительского стейта. Их названия наследуют названия родительских и дополняются строчными буквами латинского алфавита (например, D4.2-a).

Кроме тематических вопросно-ответных стейтов есть несколько стейтов для поддержания диалога: они обрабатывают такие фразы пользователя как приветствие, прощание, выражение благодарности, согласия или несогласия с ботом; кроме того, есть стейт CatchAll, в который попадают все нераспознанные запросы. Таблица, в которой содержится вся информация о вопросах и ответах, представлена в *приложении 3*.

Распознавание запросов осуществляется с помощью паттернов. **Паттерны** – это формальные правила, описывающие ключевые понятия, выражения и структуру фразы. Чат-бот анализирует фразу пользователя, сравнивая ее с разными паттернами, и затем относит к тому, которому она соответствует больше всего. Паттерны для каждого стейта прописаны так, чтобы покрыть все ключевые слова во всех вопросах, которые к нему относятся. Для слов, встречающихся в нескольких стейтах, являющихся синонимами и занимающих одну и ту же позицию в предложении, введены **именованные паттерны**. С ними вы можете ознакомиться в *приложении 4*.

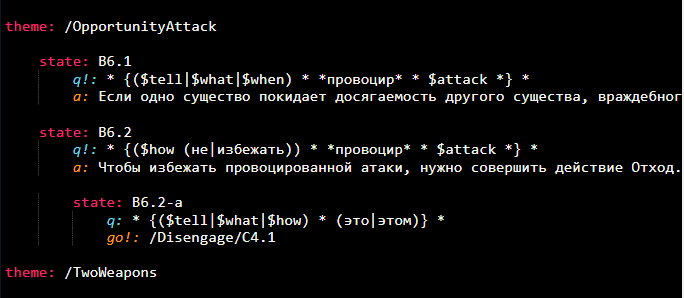
Поскольку написание **сценария** работы данного диалогового интерфейса вручную заняло бы много времени, я оптимизировала этот процесс, создав программу на языке Python. Программа считывает данные из файла с базой вопросов (который я предварительно перевела в xlsx-формат) и записывает их в том виде, который требуется для работы на платформе JAICP. Кроме этого, она обращается к txt-файлу с названиями тем, переведенных на английский язык, и добавляет их в сценарий. Пример структуры сценария вы можете видеть на *рисунке 7*. Код программы содержится в *приложении 5*.

Рисунок 7

**2.3 Тестирование чат-бота: статические методы**

После того, как была готова база вопросов, паттерны и программа для написания сценария, можно было приступать к тестированию. Следуя намеченному плану *(приложение 1)*, я начала со статических методов – тех, для выполнения которых не требовалось запускать код.

В результате **проверки орфографии и пунктуации** не было выявлено много ошибок. В нескольких местах понадобилось исправить несколько местоимений, добавить или убрать лишние отступы, а также расставить несколько запятых. Орфографических ошибок, по всей видимости, позволила избежать автоматическая проверка орфографии, встроенная в Word.

**2.3.1 Правила составления паттернов**

Прежде, чем обратиться к результатам тестирования паттернов, следует поговорить о том, как составляются паттерны и какой синтаксис в них используется. Язык паттернов почти идентичен языку регулярных выражений. Любые символы в любом количестве обозначаются символом «звездочка» («\*»). Звездочка перед каким-либо словом или после него обозначает, что в этом месте может быть любое количество других слов; звездочка, примыкающая к слову вплотную, обозначает суффикс, префикс или окончание. Например, в паттерн *\*ход\** попадут слова «ходить», «выход», «выходить».

Слова-альтернативы, занимающие в паттерне одну позицию, разделяются вертикальной чертой («|»). Слова группируются с помощью скобок. Круглые скобки служат лишь для объединения слов; квадратные скобки означают, что слово или группа слов, заключенные в них, являются опциональными; элементы внутри фигурных скобок могут встречаться в реплике пользователя в любом порядке.

Кроме этого, используется символ «тильда» («~»). Для слова, перед которым стоит этот символ, будут рассмотрены все его словоформы. Название каждого именованного паттерна начинается с символа «доллар» («$»). Именованные паттерны прописываются в отдельном файле сценария (с расширением «sc.»).

**2.3.2 Статическое тестирование паттернов**

Статическое **тестирование** **кода** дало весьма интересные результаты. В первую очередь я решила **сопоставить паттерны с примерами запросов**, чтобы убедиться в том, что все значимые слова были учтены. Упущения обнаружились, но в небольшом количестве и были в основном связаны с тем, что я не учла одно из слов в именованном паттерне. Так, например, для запроса B1.3, который посвящен определению результата атаки, одним из примеров было предложение *«Как понять, попал ли я?»*. Для глаголов со значением «узнавать», «определять» я создала именованный паттерн $learn, но слово «понять/понимать» в нем не было учтено. Другим примером «упущенного» предложения является вопрос из стейта A1.1 *«С чего начинается сражение?».* Он представляется мне довольно редким, однако его тоже нельзя было оставить без внимания; для него пришлось создать отдельный паттерн, содержащий разные формы глагола «начинать».

Нашлось несколько «избыточных» запросов – таких, в которых можно было убрать одно из слов, не потеряв при этом смысловой нагрузки. Например, для стейта G1.2, в котором идет речь об определении вида урона от атаки, паттерн выглядел следующим образом: \* {$how \* $learn \* $options \* $damage} \*. Паттерн такой длины, с семью перестановками, требует много времени на обработку; кроме того, элемент $how здесь явно не несет большой смысловой нагрузки, а только «дублирует» $learn – очевидно, что если пользователь спрашивает об определении вида урона, он хочет узнать, ***как*** это сделать. Возможен также случай, когда пользователь решит оформить свою реплику не в виде вопроса, а в виде запроса для поисковой системы: *«Определить вид урона»*. Все свидетельствует о том, что именованный паттерн $how является здесь лишним и может быть удален без потерь.

Встретились и противоположные случаи, когда нужно было добавить элемент в паттерн, чтобы избежать совпадения с другим запросом. Состояния C5.3 и C5.4 очень похожи: в первом говориться о подготовке действия в бою, а во втором – о подготовке заклинания. Паттерн для первого изначально содержал всего два элемента - $how («как» и аналоги) и *подготов\**, а паттерн для второго, кроме этих двух - $spell («заклинание» и аналоги). Чтобы запросы были распознаны верно, я решила дополнить паттерн для C5.3 именованным паттерном $action («действие» и аналоги).

В ряде случаев я сочла необходимым, не меняя содержания паттернов, объединить несколько в один или, наоборот, разбить один на два. Объединение призвано оптимизировать сценарий, сделать его короче, а разбиение – ускорить работу диалоговой системы, так как длинные запросы обрабатываются долго. Разработчики платформы JAICP советуют использовать в перестановках не более пяти-шести правил, чтобы не замедлять бот [5].

Примером объединения является паттерн состояния B8.6: \* {($tell|$possibility) \* ($attack|$action) \* $out-of-rules} \*. Он посвящен возможности совершения импровизированных действий и атак и ранее состоял из двух частей – для действий в общем и для атак в частности. Поскольку ответ бота для обоих вариантов запроса одинаков, а общий паттерн получается не слишком длинным, я сочла возможным их объединить.

В процессе анализа паттернов я придумала несколько новых примеров для запросов и добавила их в базу. Для каждого из них пришлось добавлять новые слова в паттерн или даже вводить новый именованный паттерн. Так, для запроса E2.1, касающегося прерывания перемещения во время хода персонажа, появился новый пример вопроса – *«Можно ли тратить перемещение по частям?»*. Это обусловило появление в паттерне нового элемента – (<…>|((по частям)|частями)). В запрос C7.1, посвященный действию «помощь», был добавлен новый пример – *«Могу ли я помогать друзьям?»*, и новый паттерн - \* {$possibility \* (~помогать|~помочь|помощ\*)}\*.

**2.3.3 Статическое тестирование именованных паттернов**

Претерпели изменения и именованные паттерны. Самую большую сложность для обработки представляют **вопросительные местоимения и союзы**: они часто бывают полезны для определения намерений пользователя, однако в разных контекстах могут иметь разное значение.

Слово **«что»** выполняет роль вопросительного местоимения, относительного местоимения или союза и встречается во многих запросах. Намерения пользователей, выраженные в этих запросах, нельзя свести к общему знаменателю. Сравните, к примеру, предложения «Что такое уклонение», «Что дает уклонение» и «Что нужно сделать, чтобы уклониться»: в первом случае собеседник чат-бота хочет получить определение действия, во втором – узнать о его преимуществах, а в третьем – получить инструкцию. Из-за такого обилия значений я приняла решение включить слово «что» в несколько именованных паттернов, причем в составе более сложных конструкций:

* В паттерне **$how**, маркирующем запросы со значением «как», содержится элемент (что \* чтобы) *(что нужно сделать, чтобы…)*
* В паттерне **$what** – ((что (такое|значит|это))
* В паттерне **$what-to-do**, отвечающем за просьбы пользователя о совете, – ((что \* если)|(что, если))
* В паттерне **$why**, содержающем слова со значением «зачем» – (что \* (дает|даёт))

В первоначальном варианте паттерна **$how** содержался единичный союз «чтобы», однако я сочла нужным заменить его на конструкцию (что \* чтобы) во избежание отнесения к этому паттерну лишних запросов. К примеру, автор запроса *«Где я должен стоять,* ***чтобы*** *совершить рукопашную атаку?»* желает узнать не о способе совершения действия, а о позиции, в которой его можно совершить. Соответственно, чат-бот должен проигнорировать союз «чтобы» и выделить вопросительное слово «где».

С именованным паттерном **$how** связана и другая сложность. В него входит местоимение «как», которое не всегда необходимо учитывать. В некоторых случаях без него бот не сможет правильно интерпретировать запрос: так происходит, например, в стейтах B6.1 и B6.2, которым соответствуют вопросы *«Что такое провоцированная атака»* и *«Как спровоцировать атаку».* Для бота они различаются только местоимениями. С другой стороны, если действие, о котором спрашивает игрок, в явном виде прописано в паттерне или в именованном паттерне, и нет запросов с похожей структурой, вопросительное слово «как» вполне можно опустить (как в паттерне G1.2, описанном выше). В каждом отдельном случае приходится принимать решение о необходимости его указания в паттерне.

Небольшое изменение претерпели паттерны **$when** и **$why**. Первоначально слово «почему» входило в последний, поскольку это казалось логичным: в него же входит слово «зачем», а оно имеет довольно близкое значение. Однако, пересмотрев содержание паттерна **$why**, я пришла к выводу, что он имеет несколько другое значение: слова в нем указывают не на причину какого-либо действия, а на преимущества, которое оно может дать, и влияние, которое может оказать. Таким образом, слово «почему» по смыслу оказалось ближе к паттерну «when», в котором собраны слова не только с временным, но и обстоятельственным оттенком (*когда, в каком случае*).

Тут стоит отметить, что слово «почему» встречается только в одном запросе: *«Почему я должен совершать бросок с помехой?»*. Здесь оно действительно очень близко к «когда», поскольку присутствие или отсутствие помехи при броске кубика связано с моментом времени, когда это происходит (персонаж совершает действие с помехой, *когда* на него действуют какие-либо негативные внешние силы). Если в будущем появятся другие стейты, в которых речь будет идти непосредственно о причинах какого-либо события, содержание именованных паттернов можно будет изменить.

Еще один **сложный случай** представляют собой запросы, в которых встречается какое-либо слово, входящее в состав одного из именованных паттернов, но другие слова этого паттерна для этого запроса не подходят. Например, состояние B5.1-a, которое можно описать предложением *«Какое оружие считается импровизированным?»*,содержит в себе слово «импровизированный», являющееся частью именованного паттерна $out-of-rules. Что писать в паттерне – слово «импровизированный» или именованный паттерн? С одной стороны, чтобы избежать путаницы, кажется более логичным написать именованный паттерн. С другой стороны, «импровизированное оружие» - это термин, и если пользователь захочет что-то узнать об оружии, которое не указано в правилах, он сформулирует свой запрос именно так, а не иначе. Другие же слова паттерна $out-of-rules – «другой», «альтернативный», «нет в правилах» – в этом контексте он совершенно точно использовать не будет. Как принять решение?

Следует посмотреть, не изменят ли другие слова именованного паттерна смысл предложения. Если нет таких вопросов, которые попадут в паттерн по ошибке из-за подстановки данного именованного паттерна на место требуемого слова, можно осуществлять замену. Если же есть вероятность ошибок, лучше оставить паттерн без изменений.

В ходе тестирования обнаружилось несколько случаев **пересечения паттернов**. Так, в именованном паттерне $action встретился элемент *ход\** (подразумевается существительное), а в $go – *~ходить*. Очевидно, что глагол «ходить» и все его формы, начинающиеся на ход-, попадут в оба эти паттерна. Чтобы избежать этого, элемент *ход\** потребовалось заменить на ~*ход*.

Другой пример – паттерн \*счита\* в $formula, который частично совпадает со словом «считается», являющимся частью $what. Решением этой проблемы стала замена элемента \*счита\* на два других – ~считать и ~посчитать.

Сразу несколько общих слов оказалось в паттернах $position и $occupy. Пришлось немного переосмыслить их значение: в $position остались существительные со значением «место», «позиция» и глаголы «нахождения в пространстве», а $occupy был переименован в $control и наделен словами «занимать» и «контролировать». Кроме того, встретилось несколько случаев «неявного» пересечения, когда потенциально может возникнуть путаница между двумя именованными паттернами, но на практике этого, скорее всего, не случиться. Эти случаи проще и эффективнее проверять с помощью динамического тестирования.

После данной проверки я вывела несколько **общих положений**, касающихся составления паттернов:

* Главной проблемой при составлении паттернов является отсутствие явным образом сформулированных критериев, по которым можно «на глаз» определить, является ли паттерн достаточным, недостаточным или избыточным для того, чтобы определить какой-либо запрос. Разработчик, прописывающий паттерны, может полагаться только на собственные предположения и догадки, подтвердить или опровергнуть которые может только запуск кода.
* В процессе составления паттернов легко забыть, какие именованные паттерны уже созданы и какие слова в них включены, особенно если объем стейтов большой. Ошибки, связанные с этим, проще искать и исправлять в процессе разработки, поскольку к моменту тестирования может накопиться достаточно большое количество.
* При введении нового именованного паттерна необходимо убедиться в том, что слова, входящие в него, не встречались в паттернах ранее. Если встречались, нужно определить, можно ли заменить их на именованный паттерн (описание этой проблемы было выше).

В целом, можно сделать вывод, что статическое тестирование является очень важным и полезным шагом, позволяющим не только найти ошибки и несоответствия в запросах и паттернах, но и дополнить базу вопросами и стейтами, не предусмотренными разработчиком. Этот шаг определенно поможет уменьшить количество ошибок, которые будут обнаружены с помощью тест-кейсов.

К статическим методам также относятся тестирование графического интерфейса и проверка бота на соответствование требованиям заказчика, но эти пункты не актуальны для моего чат-бота. В качестве канала я планирую использовать Телеграм (подробнее об этом – в разделе 2.5.1); бот общается только текстом, в нем не используются кнопки и картинки, поэтому никаких проблем с графическим интерфейсом быть не должно. Заказчика же у моего диалогового интерфейса нет.

**2.4 Тестирование с помощью тест-кейсов**

Автоматическое тестирование диалоговых систем производится с помощью тест-кейсов – примеров диалогов, которые могут произойти между пользователем и ботом. Платформа подает на вход чат-боту реплики пользователя из сценария и сравнивает ответ бота или то состояние, в которое он перешел, с «эталонным» - тем, которое прописано в тест-кейсе.

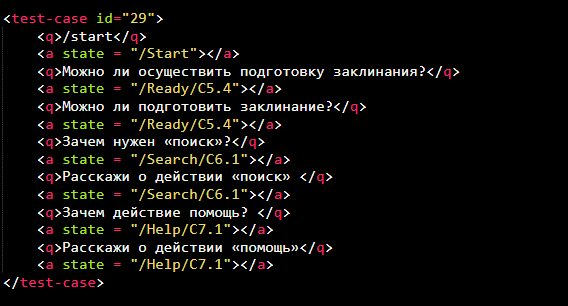
Поскольку в моем диалоговом агенте очень много состояний и примеров запросов, было необходимо автоматизировать и процесс написания тестов. Для этого я создала программу (*приложение 6*). Суть ее работы заключается в том, что она берет данные из специально созданного excel-документа и записывает в текстовый файл в виде тест-кейсов, так, как требует платформа JAICP. В каждый тест-кейс она записывает по шесть вопросов пользователя; пример такого тест-кейса вы можете видеть на *рисунке 8*. Общее количество тестов – 75.

Рисунок 8

В схеме тестирования диалогового интерфейса (*приложение 1*) тест-кейсы разбиты на три категории: те, которые проверяют кратчайший путь достижения пользователем цели (тесты «счастливого пути»), тесты, моделирующие другие пути развития диалога и «отрицательные» тесты, призванные оценить работу программы в нештатных ситуациях. Для моего бота «счастливый путь» определить нельзя, так как вопросы, с которыми пользователи приходят к нему, уникальны и могут относиться к разным областям. «Отрицательное тестирование» покрывается одним стейтом /NoMatch, обрабатывающим все нераспознанные фразы. Таким образом, моей главной задачей было протестировать максимальное количество вариантов диалога, не пропустив ни один вопрос из базы.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| запуск тестов | ошибки в паттернах | ошибки в именов. паттернах | пересече-ние паттернов | ошибки в сценарии | ошибки в тестах | особенно-сти платформы | опечатки | всего |
| 1 | 6 | 13 (11) | 6 | 4 | 25 (16) | 4 | 2 | 60 |
| 2 | 7 | 7 | 5 | - | 13 (6) | 1 | 3 | 36 |
| 3 | 1 | 4 | 3 | - | 8 (3) | 2 | 1 | 19 |
| 4 | 4 | 1 | 1 | 1 | 4 |  | 1 | 12 |
| 5 | 1 |  | 2 | 1 | 2 |  |  | 6 |
| 6 |  | 1 | 1 |  |  |  |  | 2 |
| 7 |  |  |  | 1 |  | 1 |  | 2 |

Результаты автоматического тестирования вы можете видеть в *таблице 1*. При первой компиляции обнаружилось 60 ошибок; на то, чтобы выявить их причину и устранить их, потребовалось восемь итераций. Я разбила ошибки на семь категорий; предлагаю рассмотреть каждую из них подробнее.

Таблица 1

**2.4.1 Ошибки в тестах**

Значительную часть составляют ошибки не в архитектуре бота, а в самих в тестах; причина заключается в том, что мне не сразу удалось разобраться в требованиях платформы. Сложности вызвали случаи переходов из одного состояния в другое с помощью тега «go!», который обеспечивает мгновенное перемещение с выполнением всех реакций нового стейта. Если в стейте, в который запрос попал изначально, есть какая-либо реакция, в тесте следует прописать и его, и стейт перехода (*рисунок 9*). Если же в начальном стейте реплик бота нет, в тесте указывается только то состояние, в котором бот оказался после перемещения (*рисунок 10*). Большая часть стейтов из десятой темы, посвященной различным состояниям персонажа, устроена именно так; отсюда так много ошибок. В таблице цифры без скобок обозначают общее количество ошибок, а цифры в скобках – количество уникальных. Как вы можете видеть, на первой итерации десять ошибок из двадцати пяти произошли именно по этой причине. На то, чтобы определить правильный вид тест-кейсов и устранить все неполадки, связанные с этим, потребовалось три итерации.

Вторая группа ошибок была связана с неправильно прописанным путем к стейту. Для вложенных стейтов нужно прописывать путь целиком, начиная с темы, к которой они относятся, и заканчивая названием их родительского стейта. Я при создании программы упустила этот момент, поэтому, например, вместо "/UnarmedAttack/B5.1/B5.1-a" в тесте было указано "/UnarmedAttack/B5.1-a" – а стейт с таким путем платформа найти не смогла.

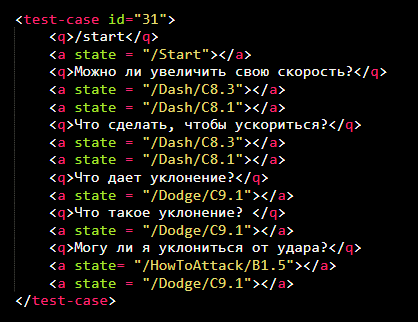


Рисунок 9

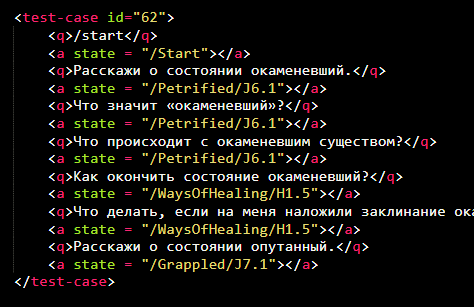


Рисунок 10

Кроме того, в некоторых тестах стейты оказались неправильно указаны или перепутаны (вероятно, эти ошибки перешли в бот из базы вопросов). Так, вопрос «Что такое «критпровал»?» бот совершенно справедливо отнес к стейту B4.2, но в тесте было написано B4.1. Встретилось несколько случаев, когда стейт в тест-кейсе был прописан правильно, но тема, к которой он относится – нет. Например, вместо "AdvantagesOfHiding/F3.1" было указано "/HideDefinition/F3.1".

**2.4.2 Ошибки в паттернах и именованных паттернах**

На втором месте по распространенности – ошибки в паттернах. Их суть заключается в том, что некоторые слова и выражения, которые человек может использовать в запросе, не были учтены. В ходе исправления этих ошибок было внесено достаточно много изменений в именованные паттерны:

* Слово «порядок» было добавлено в паттерн **$steps** (встречается во фразе «порядок сражения»);
* Слово «нести» – в паттерн **$move**;
* Слово «способ» – в **$options**;
* Слово «невраждебный» – в **$friend**;
* Слово «фут» – в **$space**;
* В фразу из паттерна **$knock\_down** была добавлена звездочка (сбить \* с ног) – для того, чтобы учесть варианты типа «сбить врага с ног»
* В паттерне **$tell** добавлен предлог «об» как разновидность «о» и звездочка между словами (расскажи \* о), чтобы учесть такие варианты как «расскажи мне о»;
* и ряд других.

Исправление ошибок в паттернах запросов так же требовало внесения в них новых слов и словосочетаний, но в некоторых случаях было необходимо менять их структуру или даже добавлять новые. Так, в стейте B7.1 («*Можно ли сражаться двумя оружиями?*») не был прописан именованный паттерн **$more\_than\_one**, содержащий слова «два», «несколько» и подобные (причем в списке именованных паттернов он был!). Для исправления этой ошибки потребовалось удалить последний элемент паттерна для B7.1 и заменить его на $**more\_than\_one**.

В паттерне для B8.6 оказалось не учтено предложение «*Можно ли придумать свой вид атаки?*»*.* Вписать его элементы в существующий паттерн оказалось невозможно, поэтому пришлось прописывать дополнительный: \* {$options \* ($attack|$action) \* (придумать|изобрести)} \*. Аналогичная ситуация произошла с вопросом «*Что происходит, когда хиты опускаются до нуля?*» из стейта H2.1. Именованный паттерн **$what\_to\_do**, который маркирует запросы о причинно-следственных связах и советы, которые пользователь спрашивает у бота, не учитывает фразы, соответствующие шаблону (что \* когда). Добавление этого шаблона именованный паттерн могло повлечь ошибки в других паттернах, в которых он используется, поэтому я приняла решение, не изменяя **$what\_to\_do,** написать для стейта H2.1 дополнительный паттерн, описывающий это предложение.

**2.4.3 Пересечения паттернов**

Попадание запроса в неправильный паттерн тоже оказалось распространенной ошибкой, а ее решение – довольно сложной задачей. В каждом отдельном случае требовалось сравнить искомый паттерн и тот, в который запрос попал по ошибке, определить причину произошедшего и найти способ устранить ее. Универсального решения у таких задач нет; я предлагаю рассмотреть несколько примеров пересекающихся паттернов, которые встретились в структуре моего бота, и изменения, которые потребовалось в них внести.

Вопрос «*Какие действия можно подготовить?*» вместо стейта C5.2 попал в C5.3. Паттерны этих стейтов подходят ему в равной степени: с каждым из них у него есть по три совпадения ({(какое|какие|что) \* $possibility \* подготов\*} и {($how|$possibility) \* подготов\* \* $action}). Для того, чтобы исключить возможность ошибки, нужно было сделать так, чтобы с паттерном для C5.2 у данного предложения было больше совпадений. Добиться этого удалось с помощью разбиения данного паттерна на два и добавления одного элемента в последний:

q!: \* {что \* $possibility \* подготов\*} \*

q!: \* {(какое|какие) $action \* $possibility подготов\*} \*

Теперь со вторым паттерном совпадает сразу четыре (по сути, все) слова предложения; при этом перегруженности паттерна удалось избежать, убрав возможность перестановки некоторых элементов.

Запрос «*Расскажи о действии «поиск»*» вместо C6.1 попал A2.1 («*Что можно делать в бою?»*). Посмотрим на паттерны этих стейтов:

С6.1: \* {($what|$why|$tell) \* поиск\*} \*

A2.1: \* {($steps|$tell|$possibility|$options) \* $action} \*

Данный запрос имеет одинаковое количество совпадений с обоими. При этом расширить паттерн для С6.1, на первый взгляд, невозможно, так как при добавлении элемента **$action** (действие) перестанут учитываться такие предложения как «*Расскажи о поиске*». Удалить какой-либо элемент из паттерна для A2.1 тоже нельзя. Решением этой проблемы стало добавление опционального элемента [$action] в первый паттерн.

Запрос «*Можно ли перемещаться ползком?*» вместо E5.3 попал в E3.2. Сравним:

E5.3: \* {($possibility|скорост\*) \* $crawl} \*

E3.2: \* {(сочетать|совмещать) \* $options \* $go} \*

\* {($on-foot|$go) \* ($fly|$swim|$crawl)} \* (и еще три запроса, призванных учесть разные комбинации способов перемещения)

В первый паттерн попадает два слова – «можно» и «ползком»; в третий – «перемещаться» и «ползком». Добавление в E5.3 именованного паттерна $go ничего бы не решило: во-первых, он исключил бы распознавание таких предложений как «Можно ли ползать?», а во-вторых, даже если сделать его опциональным, это повлечет перетягивание запросов из E3.2 (таких как, например, «***Можно*** *ли часть пути* ***пройти****, а часть* ***проползти****?*»). Для того, чтобы избавиться от этой ошибки, было необходимо полностью переписать паттерны в E3.2:

q!: \* {(сочетать|совмещать) \* $options \* $go} \*

q!: \* {($on\_foot|$go|$swim) \* (частично|часть|половину|потом)} \*

q!: \* {($fly|$crawl) \* (частично|часть|половину|потом)} \*

Теперь «*Можно ли перемещаться ползком?*» совпадает с ними только одним элементом.

Нашлось две пары почти идентичных паттернов. Первая - B8.2 («*Что такое захват?*») и J14.1 («*Что значит «схваченный»?*»):

B8.2: \* {($what|$how|$possibility) \* $capture} \*

J14.1: \* {($what|$tell|происходит\*) \* $capture} \*

Очевидно, что такая высокая степень сходства неминуемо приведет к ошибкам. Поскольку именованный паттерн **$capture** содержит и формы существительного «захват», и формы глагола «схватить», пришлось частично заменить его на другие элементы. Другим решением могло бы быть разделение всех слов смысловой группы **$capture** на два именованных паттерна – с существительными и с глаголами и глагольными формами. Однако этот вариант представляется более сложным, поскольку потребует пересмотра всех паттернов, в которых используется **$capture**, и замены его на один из двух новых именованных паттернов или на оба.

Итоговый вид паттернов:

B8.2: \* {($how|$possibility) \* $capture} \*

\* {$what \* захват} \*

J14.1: \* {($what|$tell|происходит\*) \* (схвачен\*|схватил\*)} \*

Вторая пара похожих стейтов - C9.2 и B1.5. Они оказались не только похожи, но и идентичны по смыслу: в обоих речь идет о возможности уклонения от удара. Я сочла, что наличие двух одинаковых стейтов нецелесообразно и удалила C9.2.

**2.4.4 Ошибки в сценарии**

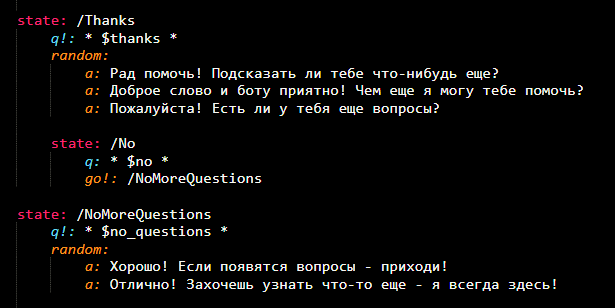
Ошибок в сценарии обнаружилось немного. В основном они были связаны с тем, что в процессе автоматического написания сценария было пропущено несколько названий тем. Так, запросы группы G1 относятся к теме «DamageTypes», однако из-за того, что эта тема не была указана в сценарии, бот отнес эти запросы к предыдущей, «AdvantagesOfHiding». Отсюда – ошибки при тестировании, когда бот попадает в правильное состояние, но название, указанное в тесте, не совпадает с тем, что написано в сценарии.

Рисунок 11

Нашлись в сценарии и лишние элементы. Например, в дочернем стейте «No», вложенном в стейт «Thanks» («*Есть ли у тебя еще вопросы?*» - «*Нет*») оказался прописан именованный паттерн $no\_questions (для таких фраз как «*Вопросов больше нет*»). В этом стейте он не нужен, так как существует отдельный стейт для подобных запросов *(рисунок 11).*

Одна из ошибок в сценарии произошла по невнимательности: при написании стейта «GotIt» я забыла добавить элемент «random» перед списком реплик бота (как на *рисунке 11*), и при его выполнении чат-бот вместо одной случайной реплики выдал все. Но такую ошибку очень легко идентифицировать и исправить.

**2.4.5 Другие ошибки**

Поскольку это был мой первый опыт работы на платформе JAICP, некоторые ошибки произошли по причине незнания мною некоторых ее особенностей. Кроме правил написания сценарий, о которых я написала выше, сложности вызвал еще один момент. В правилах написания паттернов, которые указаны в электронной документации к JAICP [27], говорится о символе «тильда», который можно добавить к слову, чтобы бот учитывал все его формы. На практике оказалось, что этот символ работает не всегда. По всей видимости, это связано с тем, что словарь, встроенный в платформу, ограничен по объему, и некоторых слов, необходимых мне, в нем нет.

Так, платформа не смогла обработать слова «попасть», «попадание», «уклониться», «рывок», «вылечить», «ослепить», «ошеломить» и «схватить». Для обозначения словоформ вместо «тильды» пришлось использовать «звездочки» (например, (ослепи\*|ослепл\*) вместо ~ослепить).

Кроме этого, обнаружилось по одной опечатке в именованных паттернах, названиях тем и тестах.

**2.4.6 Общие выводы**

Большая часть ошибок, которые были обнаружены в результате автоматического тестирования, являются результатом скорее моей неопытности, чем сложности задачи, которая передо мной стояла. Знание правил написания тест-кейсов и «подводных камней», которые могут встретиться, могло бы значительно уменьшить количество багов и ускорить процесс отладки чат-бота.

В целом же, при написании диалоговой системы, работа которой основана на паттернах, и исправлении ошибок в ее работе, есть две главные сложности. Во-первых, поскольку паттерны стейтов включают в себя именованные паттерны, они сильно от них зависят. Внесение изменений в именованный паттерн для улучшения распознавания одного запроса может повлечь ухудшение распознавания других. Кроме того, сами по себе паттерны сильно зависят друг от друга. Один паттерн может очень хорошо работать, но если появится другой, с похожей структурой, или с бóльшим количеством элементов, ситуация может измениться. Чат-бот, определяя, в какое состояние перейти, сравнивает запрос со всеми существующими паттернами и вычисляет коэффициент соответствия, поэтому любые изменения в одном из паттернов могут повлиять на «вес» другого. Этот факт – неизбежное пересечение паттернов – является основным источником ошибок.

Во-вторых, при написании сценария бота, особенно длинного, есть вероятность прописать какое-либо состояние дважды или написать два стейта, очень похожих по содержанию. Допустить такую ошибку легко, а обнаружить – сложно. В тест-кейсе прописано одно состояние, запрос попадает в другое, но это другое тоже ей подходит, и тестировщик в первую очередь решает, что первое состояние написано в тесте или сценарии по ошибке. Было бы полезно иметь встроенный в платформу инструмент, который сравнивал бы паттерны разных стейтов и при слишком большом проценте совпадений предупреждал бы об этом разработчика.

Самые сложные для обработки случаи – те, когда причину возникновения ошибки не сразу удается установить. Чтобы они возникали реже, необходимо более внимательно подходить к написанию сценария и тест-кейсов. Если тестировщик уверен в том, что инструменты тестирования работают исправно, он будет больше на них полагаться и в сложных ситуациях в первую очередь предполагать причиной ошибки пересечение паттернов, а не невнимательность разработчика тестов.

**2.5 Тестирование с привлечением пользователей**

**2.5.1 Загрузка чат-бота в канал**

Для того, чтобы пользователи могли пообщаться с ботом, нужно загрузить его в какой-либо канал. В качестве канала может выступать мессенджер, социальная сеть, голосовой ассистент, какая-либо платформа, сервис или телефонная линия. Для загрузки своего диалогового агента я выбрала мессенджер Telegram.

Telegram — бесплатный мессенджер, позволяющий обмениваться текстовыми сообщениями и файлами различных форматов, доступный для различных устройств и операционных систем [13]. Согласно статистическим исследованиям, это наиболее популярная и универсальная платформа, ее выбирают пользователи разных возрастов, в том числе и молодые люди, на которых в первую очередь рассчитан мой диалоговый агент [20].

Боты в Telegram по сути являются специальными аккаунтами, выполняющими роль внешнего интерфейса для сервиса, который работает на удаленном сервере. Они запускаются со стороны пользователя и отправляют запросы к Telegram Bot API [22]. Для того, чтобы подключить своего бота к этому приложению, необходимо получить специальный токен. Регистрирует боты и выдает токены специальный бот, которого зовут BotFather. Он просит придумать для чат-бота имя и никнейм, по которому пользователи смогут его найти; никнейм обязательно должен оканчиваться на «-bot», чтобы его нельзя было спутать с живым человеком.

После получения токена его необходимо указать на платформе JAICP в разделе «Каналы». Как только канал будет подключен, с ботом можно начинать общение. Если при подключении в настройках указать «публиковать автоматически», при каждом изменении в структуре бота и удачном прохождении тестов в канале будет публиковаться новая версия бота.

**2.5.2 Общение с пользователями**

На платформе JAICP можно посмотреть записи диалогов пользователей с ботом. Для этого нужно зайти в раздел «Аналитика» и открыть вкладку «Диалоги».

К тестированию бота было привлечено десять человек. Шесть из них знакомы с правилами игры DnD, четыре – нет. Целевой аудиторией бота являются люди, знающие, как выглядит партия в Dungeons$Dragons и (в идеальном случае) имеющие опыт игры, на котором они могли бы основывать свои вопросы. Однако не исключена возможность попадания бота в руки людей, незнакомых с игрой и желающих полностью освоить ее с помощью бота, поэтому посмотреть на общение чат-бота с «новичками» тоже было полезно.

Прежде чем перейти к анализу диалогов, я бы хотела сделать одно замечание. Бот разрабатывался в первую очередь как помощник игрока *во время игры*, поэтому заточен на более частные вопросы, которые потенциально могут возникнуть у игрока в процессе сражения. Участники же моего эксперимента задавали вопросы «из головы», поэтому они в основном носят общий характер. Проверить работу бота «в полевых условиях» в рамках данной дипломной работы не представляется возможным, однако мне кажется, что те данные, которые были получены от пользователей, являются достаточно информативными.

Первое наблюдение касается манеры общения пользователей: они склонны задавать **короткие вопросы**, часто даже в форме утверждений. Это, вероятно, является отражением того, как молодые люди привыкли общаться в мессенджерах. Вот несколько примеров таких запросов:

* бонусное действие
* степени укрытия
* действия в бою
* сколько раундов
* инициатива
* зелья лечения
* атака

В некоторых случаях нельзя достоверно определить, какую именно информацию пользователь хочет получить. Например, запрос «инициатива» может обозначать «Что такое инициатива?», «На что влияет инициатива?», «Как посчитать инициативу?». Для обработки таких коротких запросов нужно вводить новые паттерны в стейты и новые стейты в сценарий – такие, в которых бот уточнял бы у пользователя, что именно он хочет узнать о названном им объекте.

Пользователи, совсем не знакомые с игрой, задавали боту очень **общие вопросы**, часто выходящие за рамки темы «сражение», или **«глупые» вопросы**, ответы на которые я посчитала очевидными и не стала добавлять в сценарий. Среди таких вопросов, например:

* Что такое атака
* Какие зелья бывают?
* Кто такой мастер?
* Сколько участников может быть?
* Какие предметы можно использовать?
* Где брать деньги?
* Как покупать оружие?

Несмотря на то, что чат-бот рассчитан на людей, ориентирующихся в правилах, стоит учесть и возможность таких вопросов.

Опытные игроки задали несколько **вопросов по теме «сражение»**, которые мною **не были учтены**:

* Можно ли биться двумя мечами? (Частный случай запроса B7.1 *«Можно ли сражаться двумя оружиями?»*, который я не прописала);
* Можно ли отвлечь врага? (Частный случай запроса C7.1 *«Зачем нужно действие помощь?»*);
* Можно ли использовать заклинание, если не хватает компонентов? (Частный случай запроса C3.1 *«Можно ли наложить заклинание?»*)
* Какой модификатор урона у меча?
* Как щит влияет на Класс Доспеха?
* Как определить Класс Доспеха?

Эти запросы нужно обязательно добавить в базу, чтобы бот мог помогать не только начинающим, но и опытным игрокам. Кроме этого, люди, знакомые с правилами, задали несколько вопросов, на которые чат-бот мог бы ответить, но не смог распознать:

* Какие есть действия в бою? (должен был попасть в A2.1)
* Как работает инициатива? (I1.1)
* Как работает подготовка? (C5.1)
* Как работает помощь? (C7.1)
* Рывок что это (C8.1)
* Где брать лечащие зелья? (H1.3)

Один из участников попытался **узнать о возможностях бота**, задав вопрос «Что ты умеешь?». Ответ на такой вопрос не был мной предусмотрен; бот в начале диалога сообщает, на какие темы умеет говорить, и я сочла, что этого достаточно. С другой стороны, если пользователя не удовлетворил этот общий список и он хочет узнать о том, как бот может ему помощь, больше, такой стейт может быть полезен.

Некоторые «промахи» бота были связаны с тем, что он не умеет обрабатывать слова **с** **орфографическими ошибками**. Так, один из пользователей попытался узнать у бота об одной из степеней укрытия («наполовину»), но написал слово неправильно («на половину»), и реплика не была распознана.

Смысл двух вопросов из заданных пользователями, формально относящихся к теме чат-бота, не удалось установить. Это вопросы «Как тратится перемещение?» и «Назови зелье лечения». Поскольку не известно, какую именно информацию хотели получить авторы, внести их в базу не представляется возможным.

В целом хочется отметить, что бот, обладая достаточно большой базой вопросов, при общении с пользователями довольно часто заходит в тупик. Это происходит не только из-за недостатка знаний, но в следствие довольно слабо развитых «коммуникативных навыков»: если пользователь хочет уточнить какую-то деталь или значение какого-то слова, бот не всегда понимает, о чем идет речь. Это можно исправить добавлением большего количества вложенных стейтов и переходов между состояниями, - тогда диалог будет казаться более «живым».

На те темы, которые в него заложены, бот отвечает достаточно хорошо, хотя и здесь ошибок не удалось избежать. Пользователи отметили обстоятельность ответов бота, грамотность и связность его речи, вежливость. В целом отзывы скорее положительные. Информация, полученная в результате тестирования, может быть использована для улучшения работы диалоговой системы в будущем.

**ЗАКЛЮЧЕНИЕ**

В рамках данной дипломной работы было проведено исследование теории и практики тестирования программных продуктов в целом и диалоговых систем в частности. В первой главе подробно рассмотрены различные подходы к тестированию, описаны методики, обозначены основные этапы тестирования. Также уделено внимание процессу тест-дизайна, положительным и отрицательным аспектам автоматизации тестирования, и на основе литературы по данной теме прописаны рекомендации по улучшению качества тестирования.

Вторая половина первой главы посвящена теории разработки и тестирования диалоговых интерфейсов. Было дано определение диалоговой системы, обозначены возможности современных разговорных интерфейсов и перспективы их развития; описаны принципы, на основе которых осуществляется их работа, и инструменты для их создания. Из источников были выделены требования, которым должен удовлетворять чат-бот, и рекомендации по их выполнению. Помимо этого, была изучена и описана платформа JAICP от компании JUST AI и выбран ряд методик, применимых для тестирования диалоговых систем.

Был описан следующий набор методов, позволяющих наиболее полно протестировать диалоговую систему:

* Статические методы:
  + Поиск ошибок в коде;
  + Поиск ошибок в тесте;
  + Проверка переходов между состояниями;
  + Тестирование графического интерфейса;
  + Проверка соответствия требованиям заказчика;
* Динамические методы:
  + Тестирование с помощью тест-кейсов;
  + Тестирование «счастливого пути»;
  + «Отрицательное тестирование»;
  + Тестирование с привлечением пользователей;
  + Оценка удобства пользования.

Практическая часть работы заключалась в проектировании, создании и тестировании диалогового интерфейса. Во второй главе описано устройство созданного мной чат-бота, а также обоснован выбор тематической области, которой он посвящен. Сценарий для бота был написан автоматически, с помощью созданной в рамках данной работы программы на языке Python; устройство программы обозначено в тексте главы, а ее код содержится в приложении 4.

Тестирование было разделено на три этапа: статическое, с помощью тест-кейсов и с привлечением пользователей. В ходе статического тестирования база данных и сценарий чат-бота были проверены на наличие орфографических, пунктуационных и речевых ошибок, а также ошибок в паттернах для стейтов и именованных паттернах. Выявленные ошибки приведены в тексте главы.

Для написания тест-кейсов была создана еще одна программа; ее код вы можете найти в приложении 5. Ошибки, выявленные в ходе автоматического тестирования, были классифицированы, а их количество после каждого набора внесенных в устройство бота изменений, занесено в сводную таблицу. Каждый класс ошибок подробно рассмотрен и подкреплен примерами; описаны причины возникновения ошибок и способы их исправления. Сделаны выводы об особенностях устройства и тестирования чат-ботов и сложностях, с которыми может столкнуться разработчик.

Последний этап тестирования чат-бота проводился с привлечением пользователей. Им было предложено пообщаться с диалоговым агентом, подключенным к мессенджеру Telegram, и дать характеристику его работы. Записи диалогов были обработаны, и на их основе сделаны выводы о манере общения собеседников чат-бота, достаточности объема его базы данных и недочетах в его устройстве. Пользователи назвали работу диалоговой системы удовлетворительной, но выявили в ней несколько недостатков.

В результате проведенного эксперимента были сделаны выводы об эффективности методов тестирования чат-ботов:

* **Статические методы** полезны при проектировании и разработке чат-бота. В процессе написания сценария разработчику следует просматривать готовый текст и паттерны на предмет ошибок, но по окончании разработки целесообразнее сразу переходить к динамическим методам. «Ручная» проверка бота занимает гораздо больше времени, а количество найденных ошибок оказывается в разы меньше, чем при автоматической.
* **Проверка с помощью тест-кейсов** – высокоэффективный метод, позволяющий за небольшое количество времени найти и отладить большое количество ошибок. С помощью автоматизации процесса написания тестов его скорость можно сделать еще выше. Эффективность этого метода зависит от количества вариантов, которые тестировщик смог собрать для каждого из прописанных в сценарии запросов. Чем больше вариантов, тем меньше шанс, что при эксплуатации бот не сможет распознать какую-либо фразу.
* **Тестирование с привлечением пользователей** также можно назвать эффективным методом, поскольку оно позволяет найти «белые пятна» в сценарии: запросы и слова, которые не были учтены при разработке. Оно также дает возможность получить информацию о реальных потребностях людей, о том, какие из путей диалога наиболее популярны, и о манере речи пользователей, на которых ориентирован бот.

Использование тест-кейсов в связке с «живым» тестированием бота пользователями позволяет получить наиболее полную картину о достоинствах и недостатках бота и разработать стратегию улучшения качества его работы.

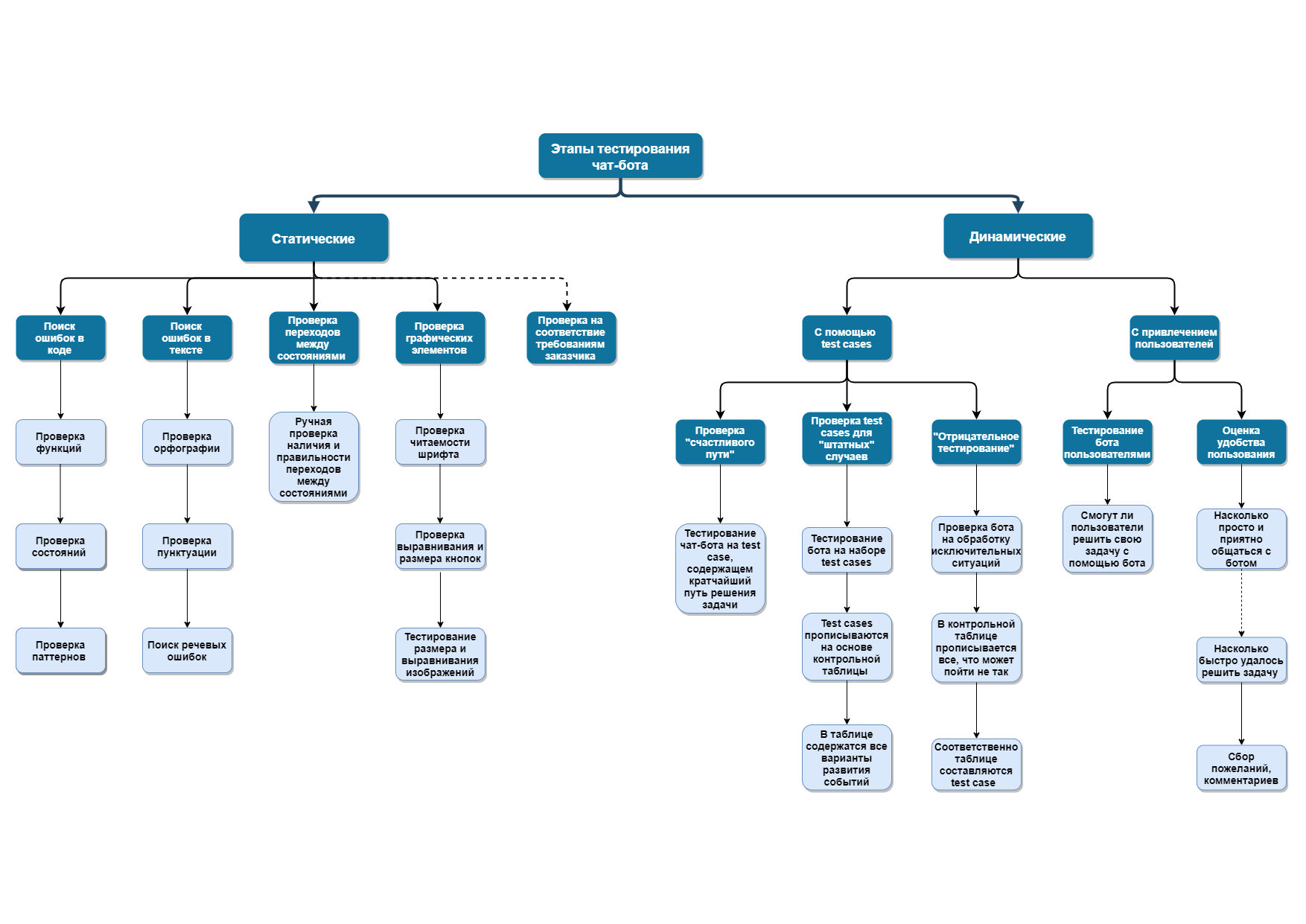
Тема данного исследования была предложена компанией Just AI.

**Литература и электронные ресурсы:**

1. Власов М. П. Моделирование экономических процессов / М. П. Власов, П. Д. Шимко. — Ростов н/Д : Феникс, 2005. – 409 с
2. Глазков, С.В. Разработка интерактивных приложений с многомодальным интерфейсом для гетерогенных мобильных устройств. // Четвертый междисциплинарный семинар «Анализ разговорной русской речи». – 2010. – С. 51-56
3. Джанарсанам, С. Разработка чат-ботов и разговорных интерфейсов. – М.: ДМК Пресс, 2019. – 340 с.
4. Документация JAICP. [электронный ресурс]. – <https://help.just-ai.com/docs/ru/>
5. Документация JAICP. Базовые элементы паттернов. [электронный ресурс]. – https://help.just-ai.com/docs/ru/Patterns/base\_patterns
6. Еремеев, Владислав. Библия QA v. 2.0 [электронный ресурс] – https://vladislaveremeev.gitbook.io/qa\_bible/
7. Иванов А. Д. Чат-боты в Telegram и в контакте как новый канал распространения новостей // Вестник Волжского университета им. В. Н. Татищева, Том 1, № 3, 2018 г., 126-132 стр.
8. Кадеева, О.Е., Сирицына. В.Н. Чат-боты и особенности их использования в образовании. // журнал «Информатика в школе», номер 10 (163), 2020, стр. 45-53.
9. Каплун, А. Тестирование областей определения или нечто большее, чем анализ граничных значений [электронный ресурс]. – https://habr.com/ru/company/infopulse/blog/270909/ - статья в интернете
10. Кормушина, Д. Как мы тестируем фичу от ТЗ до пост-продакшена и сохраняем дружеские отношения внутри команды [электронный ресурс]. – https://habr.com/ru/company/2gis/blog/449942/ - статья в интернете.
11. Кузнецов В. В. Перспективы развития чат-ботов//Успехи современной науки. – 2018. – № 12, 16–19
12. Куликов, С. C. Тестирование программного обеспечения. Базовый курс: учебное пособие. С.С. Куликов. — Минск: Четыре четверти, 2020. — 312 с.
13. Лутфуллин А. Умные боты для Telegram. [электронный ресурс]. – <http://android.mobile-review.com/articles/3621> - статья в интернете
14. Мохова, М. К. Применение технологии чат-бот для автоматизации коммуникаций с клиентами // Весенние дни науки. – 2020. – С. 702-704.
15. Потапова, Н. 10 правил хорошего тона при описании багов [электронный ресурс]. – https://habr.com/ru/company/docsvision/blog/264163/ - статья в интернете
16. Рокатанский, М. Меньше «сложного» тестирования, больше — «умного» тестирования [электронный ресурс]. – https://habr.com/ru/company/otus/blog/554638/ - статья в интернете.
17. Рукол, Н. Тестирование — это не поиск ошибок! [электронный ресурс]. – https://www.software-testing.ru/library/testing/general-testing/1713-2012-08-23-10-03-18 - статья в интернете
18. Смирнов С.Ю. Методы и метрики для оценки качества чат-бота. Журнал «Моя профессиональная карьера». Том 1, номер 10, 2020. c. 136-140.
19. Смыслова Л. В. Чат-бот как современное средство интернет-коммуникаций // Молодой ученый. — 2018. — № 9 (195). — С. 36-39.
20. Соколова, А. Чат-революция: почему боты убьют мобильные приложения. [электронный ресурс]. – <https://rb.ru/longread/bots-are-the-new-apps/> - статья в интернете
21. Тестирование и качество ПО. [Электронный ресурс]. – http://software-testing.ru/
22. Тугушева Н.А. Использование чат-ботов в различных сферах повседневной жизни. // Молодой ученый 21 (155), 2017, с. 36-39
23. Уиттакер, Дж. Как тестируют в Google / Дж. Арбон, Дж. Каролло. – Спб.: Питер, 2014. – 320 с.
24. Шауар Б. и Этвел Э. Chatbots: они действительно полезны? // Форум LDV. — 2007. — Т. 22. — №. 1. — С. 29–49/
25. Copland, L. A Practitioner’s Guide to Software Test Design: учебное пособие. – Boston, Artech House Publishers, 2004. – 358 c.
26. Just AI: о компании. [Электронный ресурс]. – <https://justai.ru/o-kompanii>
27. Just AI: платформа JAICP. [Электронный ресурс]. – <https://justai.ru/platforma-jaicp>
28. Lambert, R. Explaining Exploratory Testing Relies on Good Notes [электронный ресурс]. - http://thesocialtester.co.uk/explaining-exploratory-testing-relies-on-good-notes/ - статья в интернете
29. Manh, L. Ultimate glossary of software testing terms for beginner testers [электронный ресурс]. - https://www.asktester.com/smoke-test-vs-sanity-test-vs-retest-vs-regression-test/ - статья в интернете
30. Stephenson, J. Creative and Critical Thinking and Testing Part 1 [электронный ресурс]. - http://steveo1967.blogspot.com/2013/03/creative-and-critical-thinking-and.html - статья в интернете

**ПРИЛОЖЕНИЯ**

***Приложение 1***



***Приложение 2***

**Механики боя в DnD**

Тематические разделы

1. В целом о сражении
2. Атака
3. Другие виды действий
4. Бонусное действие
5. Перемещение
6. Укрытие
7. Урон
8. Лечение
9. Очередность ходов
10. Состояния

Тематические разделы (подробнее)

1. В целом о сражении
   1. Как выглядит сражение
   2. Из чего состоит ход
2. Атака
   1. Совершение атаки
   2. Ближняя и дальняя атака
   3. Помеха и преимущество
   4. Критуспех и критпровал
   5. Безоружная атака
   6. Провоцированная атака
   7. Сражение двумя оружиями
   8. Виды атаки
   9. Определение атаки
3. Другие виды действий
   1. Засада
   2. Взаимодействие с предметами
   3. Наложение заклинаний
   4. Отход
   5. Подготовка
   6. Поиск
   7. Помощь
   8. Рывок
   9. Уклонение
4. Бонусное действие
   1. Бонусные действия
   2. Реакции
5. Перемещение
   1. Скорость
   2. Прерывание перемещения
   3. Виды перемещения
   4. Труднопроходимая местность
   5. Падение ничком
   6. Чужое пространство
6. Укрытие
   1. Определение укрытия
   2. Механизм укрытия
   3. Преимущества укрытия
7. Урон
   1. Виды урона
   2. Уязвимость
   3. Сопротивление
   4. Иммунитет
8. Лечение
   1. Механики лечения
   2. Ноль хитов
   3. Воскрешение
9. Очередность ходов
   1. Инициатива
   2. Элемент неожиданности
10. Состояния
    1. Бессознательный
    2. Испуганный
    3. Невидимый
    4. Недееспособный
    5. Оглохший
    6. Окаменевший
    7. Опутанный
    8. Ослепший
    9. Отравленный
    10. Очарованный
    11. Ошеломленный
    12. Парализованный
    13. Сбитый с ног
    14. Схваченный

***Приложение 3***

|  |  |  |  |
| --- | --- | --- | --- |
| Состо-яние | Паттерны | Вопросы | Ответ |
| A1.1 | \* {($what|$how|$steps|$tell) \* $fight \* [$dnd]} \*  \* {(~начать|~начинать|~начинаться|  ~начаться) \* $fight \* [$dnd]} \* | *Как выглядит сражение в ДнД?* | Сражения в ДнД происходят раундами. Каждый участник сражения совершает один ход за раунд. Порядок ходов определяется в начале боевой сцены с помощью броска на инициативу. Если кого-то из участников застали врасплох, он пропускает первый ход. Ход состоит из перемещения и действия, например, атаки. Сражение продолжается до победы одной из сторон. |
| *Сражение в ДнД пошагово* |
| *С чего начинается сражение?* |
| *Расскажи о сражении в ДнД* |
| *Как сражаться с врагами в ДнД?* |
| *Как происходит сражение в ДнД* |
| *Порядок сражения в игре* |
| A2.1 | \* {($steps|$tell|$possibility|$options) \* $action} \* | *Что я могу сделать в свой ход?* | В свой ход ты можешь переместиться на расстояние, не превышающее твою скорость, и совершить одно действие. Также ты можешь совершить одно бонусное действие, если твой персонаж обладает соответствующим умением.  Среди возможных действий – атака, засада, использование предмета, накладывание заклинания, отход, подготовка, помощь, поиск, рывок, уклонение. Ты можешь обсудить с Мастером и другие варианты. Желаешь узнать об одном действий подробнее? |
| *Какие действия я могу совершать во время хода?* |
| *Из чего состоит ход?* |
| *Расскажи о видах действий в сражении* |
| *Что я могу делать во время хода?* |
| B1.1 | \* {($how|$steps) \* $attack} \* | *Как совершить атаку?*  *Как атаковать?*  *Как атаковать противника?*  *Порядок совершения атаки*  *Из каких этапов состоит атака?*  *Из чего состоит атака?*  *Что сделать, чтобы нанести удар?*  *Что сделать, чтобы атаковать мечом?* | Для того, чтобы совершить атаку, нужно:  1. Определить цель;  2. Определить модификаторы (есть ли у кого-то помеха, преимущество, умения, влияющие на результат атаки);  3. Совершить бросок атаки;  4. Если атака попала по противнику, определить урон.  Также есть и другие виды атаки, например, Захват и Толкание. |
| B1.2 | \* {$formula \* ($success|$attack)} \* | *Как посчитать попадание?*  *Какой кубик нужно кинуть на попадание?*  *Какая формула для попадания?*  *Какая формула для броска атаки?* | Бросок атаки: d20 + модификатор + бонус мастерства.  Для атаки рукопашным оружием используется модификатор Силы; для атаки дальнобойным оружием – модификатор Ловкости. Для оружия с пометкой «фехтовальное» или «метательное» можно выбрать любой из этих модификаторов.  ! Бонус мастерства прибавляется только в том случае, если ты владеешь оружием, которое используешь. |
| B1.3 | \* {$success \* (КД|кд|(Класс\*|класс\*) (Доспеха|доспехa))} \*  \* {$success \* ($attack|~цель)} \*  \* {результат\* \* $attack} \*  \* {$success \* $learn} \* | *Как понять, попал ли я?*  *Как определить, была ли атака успешной?*  *Как определить результат атаки?*  *Атака должна быть больше КД или равна КД, чтобы попасть?*  *В каком случае удается попасть в цель?*  *Если мое попадание равно КД, я попал?* | Если сумма результата броска и модификаторов не меньше Класса Доспеха (КД) цели, атака попадает. |
| B1.4 | \* {($formula|$how-much) \* $damage} \* | *Как посчитать урон?*  *Как вычислить урон?*  *Как понять, сколько хитов отняла атака?*  *Формула подсчета урона* | Урон от атаки: кость оружия + модификатор  Для атаки рукопашным оружием используется модификатор Силы; для атаки дальнобойным оружием – модификатор Ловкости. Для оружия с пометкой «фехтовальное» или «метательное» можно выбрать любой из этих модификаторов. |
| B1.5 | \* {($possibility|$how) \* (~уклониться|~избежать)} \* | *Можно ли уклониться от атаки?*  *Как уклониться от удара?* | В свой ход ты можешь сосредоточиться на уклонении от атак. Переход в состояние C9.1 |
| B2.1 | \* {$what \* $long-range \* $attack} \* | *Что такое дальняя атака?*  *Какая атака считается дальнобойной?* | Если ты стреляешь из лука или арбалета, кидаешь ручной топор или другой снаряд во врага с расстояния, ты совершаешь дальнобойную атаку. |
| B2.2 | \* {$what \* $hand-to-hand \* $attack} \* | *Что такое ближняя атака?*  *Какая атака считается рукопашной?* | Если ты сражаешься мечом, боевым молотом, топором или подобным оружием, требующим нахождения в пределах 5 футов от противника, ты совершаешь рукопашную атаку. |
| B2.3 | \* {$possibility \* $hand-to-hand \* $long-range} \* | *Могу ли я совершить дальнобойную атаку в ближнем бою?*  *Я могу выстрелить из лука, стоя вплотную к врагу?* | Если ты находишься в пределах 5 футов от враждебного не недееспособного существа, бросок атаки дальнобойным оружием совершается с помехой. |
| B2.4 | \* {($position|$distance) \* $hand-to-hand} \* | *Из какой позиции я могу атаковать рукопашным оружием?*  *С какого расстояния я могу совершить рукопашную атаку?*  *Где я должен стоять, чтобы совершить рукопашную атаку?*  *Мне нужно стоять к врагу вплотную, чтобы совершить ближнюю атаку?* | Рукопашная атака совершается с расстояния 5 футов. |
| B2.5 | \* {($position|$distance) \* $long-range} \* | *Из какой позиции я могу атаковать стрелковым оружием?*  *С какого расстояния я могу совершить дальнобойную атаку?*  *Где я должен стоять, чтобы совершить дальнюю атаку?*  *Стрелять из лука нужно с расстояния?* | Дистанция, в пределах которой можно совершить атаку дальнобойным оружием, индивидуальна для каждого оружия. |
| B2.6 | \* {($what|$tell|$difference) \* $hand-to-hand \* $long-range} \* | *В чем разница между ближней и дальнобойной атакой?*  *Расскажи о рукопашной и дальнобойной атаках*  *Какая атака называется дальнобойной, а какая рукопашной?* | Если ты стреляешь из лука или арбалета, кидаешь ручной топор или другой снаряд во врага с расстояния, ты совершаешь дальнобойную атаку. Если твое оружие требует стоять к врагу вплотную, ты совершаешь рукопашную атаку. |
| B2.7 | \* {[$action] \* $long-range \* перезаря\*} \*  \* {$why \* \* перезаря\*} \* | *Нужно ли тратить действие на перезарядку лука?*  *На что влияет свойство оружия «перезарядка»?*  *Нужно ли перезаряжать арбалет?* | Если у оружия есть свойство «перезарядка», ты можешь выстрелить из него только один боеприпас одним действием, бонусным действием или реакцией вне зависимости от количества положенных атак. Тратить действие на перезарядку не нужно. |
| B3.1 | \* {($learn|$when|$why|$how) \* (помех\*|преимуществ\*)} \* | *Как узнать, есть ли у кого-то помеха?*  *Как определить, есть ли у меня преимущество?*  *Когда появляется преимущество?*  *Как получить преимущество при атаке?*  *Почему я должен совершить бросок с помехой?* | Есть ли у кого-то из игроков помеха или преимущество при броске, определяет Мастер. Обычно помеха или преимущество появляется из-за действия каких-либо эффектов от заклинаний, удачного/неудачного положения существа в пространстве, правильно/неправильно подобранного оружия и брони и других факторов. |
| B3.2 | \* {$what \* помех\*} \* | *Что значит иметь помеху?*  *Что такое помеха?*  *Что такое помеха при броске?*  *Что значит бросить кубик с помехой?*  *Что значит атаковать с помехой?* | Если ты совершаешь бросок атаки или проверку характеристики с помехой, кидай кубик дважды и выбирай меньшее значение. |
| B3.3 | \* {$what \* преимуществ\*} \* | *Что значит иметь преимущество?*  *Что такое преимущество?*  *Что такое преимущество при броске?*  *Что значит бросить кубик с преимуществом?*  *Что значит атаковать с преимуществом?* | Если ты совершаешь бросок атаки или проверку характеристики с преимуществом, кидай кубик дважды и выбирай большее значение. |
| B4.1 | \* {($what|$what-to-do|$why) \* $crit-success} \* | *Что будет, если выпадет «20»?*  *Что такое «критуспех»?*  *Что дает критическое попадание?*  *Что дает критическое попадание?* | Если при броске атаки выпало «20», атака попадает вне зависимости от всех модификаторов и КД цели. Кроме того, в этом случае кости урона бросаются два раза и их результаты складываются.  Если ты выкинул «20» при проверке характеристики, она считается успешной независимо от ее Сложности. |
| B4.2 | \* {($what|$what-to-do|$why) \* $crit-failure} \* | *Что, если выпало «1»?*  *Что такое «критпровал»?*  *Что дает критический провал?* | Если при броске атаки выпало «1», атака промахивается вне зависимости от всех модификаторов и КД цели.  Если ты выкинул «1» при проверке характеристики, она считается проваленной независимо от всех модификаторов. |
| B5.1 | \* {($possibility|$how|$what-to-do) \* (((нет|без|кроме) оружи\*)|безоружн\*)} \* | *Могу ли я атаковать без оружия?*  *Что делать, если у меня нет оружия?*  *Как провести безоружную атаку?*  *Как провести атаку без оружия? Можно ли драться чем-то кроме оружия?* | Ты можешь использовать безоружный удар, например, ударить противника кулаком. Такой удар причиняет дробящий урон 1 + твой модификатор Силы. Кроме того, ты можешь использовать импровизированное оружие: любой предмет, который можно держать одной или двумя руками. Рассказать подробнее об атаке импровизированным оружием? |
| B5.1-a | $yes | *Да* | В качестве импровизированного оружия может выступить, например, разбитая бутылка, ножка от стола или даже мертвый гоблин. Если предмет похож на настоящее оружие, Мастер может разрешить использовать Бонус мастерства. Если нет, оружие причиняет урон 1d4. У импровизированного метательного оружия нормальная дистанция 20 фт, максимальная – 60 фт. |
| \* {($tell|$what) \* оружи\* \* $out-of-rules} \* | *Расскажи об атаке импровизированным оружием.*  *Какое оружие называется импровизированным?* |
| B5.1-b | $no | *Нет* | Хорошо. Чем еще я могу помочь? |
| B6.1 | \* {($tell|$what|$when) \* \*провоцир\* \* $attack \*} \* | *Что такое провоцированная атака?*  *Когда я провоцирую атаку?* | Если одно существо покидает досягаемость другого существа, враждебного ему, оно провоцирует одну рукопашную атаку, которая совершается реакцией. |
| B6.2 | \* {($how (не|избежать)) \* \*провоцир\* \* $attack \*} \* | *Что нужно сделать, чтобы не спровоцировать атаку?*  *Как избежать провоцирования атаки?* | Чтобы избежать провоцированной атаки, нужно совершить действие Отход. |
| B6.2-a | \* {($tell|$what|$how) \* (это|этом)} \* | *Расскажи об этом подробнее*  *Что это?*  *Как это сделать?* | Переход в состояние C4.1 |
| B7.1 | \* {$possibility \* оружи\* \* (одновременно|сразу|(за ход)|(за один ход))} \* | *Можно ли сражаться двумя оружиями?*  *Могу ли я сражаться двумя оружиями одновременно?*  *Могу я использовать два оружия за один ход?*  *Я могу атаковать двумя оружиями сразу?* | Ты можешь сражаться двумя оружиями, если у каждого из них есть свойство «легкое». Если у одного из оружий есть также свойство «метательное», ты можешь совершить им дальнобойную атаку. Модификатор характеристики к урону от атаки вторым оружием не прибавляется! |
| B8.1 | \* {$options \* $attack} \*  \* {$how \* $possibility \* $attack} \* | *Какие есть виды атаки?*  *Как можно атаковать врага?* | Ты можешь совершить по врагу дальнобойную или рукопашную атаку, с использованием оружия или безоружную. Есть и особые виды атаки: Захват и Толкание. Захват делает врага схваченным, Толкание сбивает его с ног. Также атаку можно спровоцировать, в этом случае она совершается реакцией. Рассказать подробнее об одном из видов атаки? |
| B8.1-a | \* {$options \* (еще|ещё|другой)} \* | *Есть ли какие-то другие варианты?*  *Есть ли какие-то другие виды?* | переход в состояние B8.6 |
| B8.2 | \* {($what|$how|$possibility) \* $capture} \* | *Что такое захват?*  *Как совершить захват?*  *Можно ли захватить врага?*  *Можно ли поймать врага?*  *Можно ли обездвижить врага?* | Захват – один из видов атаки. Используя как минимум одну свободную руку, ты пытаешься схватить цель, совершая проверку Силы (Атлетика), противопоставленную проверке Силы (Атлетика) или Ловкости (Акробатика) цели. При успехе цель становится захваченной. |
| B8.3 | \* {($what-to-do|$undo) \* $capture} \*  \* {(\*свободиться|вырваться) \* $capture} \* | *Как высвободиться из захвата? Что делать, если меня схватили?*  *Что нужно сделать, чтобы освободиться из захвата?* | Для того, чтобы высвободиться, в свой ход соверши проверку Силы (Атлетика) или Ловкости (Акробатика), противопоставленную проверке Силы (Атлетика) противника. |
| B8.4 | \* {$possibility \* ($go|$move) \* $capture} \* | *Можно ли нести схваченное существо?*  *Можно ли перемещать схваченное существо?*  *Можно ли тащить схваченное существо?*  *Можно ли перемещаться со схваченным существом?*  *Можно ли ходить вместе со схваченным существом?* | Да. Для этого придется потратить в два раза больше скорости (только если существо не меньше тебя как минимум на два размера). |
| B8.5 | \* {($what|$how|$possibility) \* $knock-down} \* | *Можно ли сбить врага с ног?*  *Как сбить с ног?*  *Как сбить с ног повалить на землю?*  *Что такое Толкание?*  *Как совершить Толкание?* | Толкание – особый вид атаки, с помощью которого ты можешь сбить врага с ног либо оттолкнуть от себя на 5 футов. Для этого нужно совершить проверку Силы (Атлетика) против проверки Силы (Атлетика) или Ловкости (Акробатика) цели. |
| B8.6 | \* {($tell|$possibility) \* ($attack|$action) \* $out-of-rules} \* | *Можно ли придумать свой вид атаки?*  *Расскажи об импровизированном действии.*  *Можно ли совершить действие, которого нет в правилах?* | Если ты хочешь совершить действие, не упомянутое в правилах, Мастер сообщит, возможно ли это действие, и какие проверки нужно совершить для определения успешности. |
| B9.1 | \* {($what|(как понять)) \* $attack} \* | *Какое действие считается атакой?*  *Как понять, что я совершаю атаку?* | Атакой считается любое действие, для которого нужно совершить бросок атаки. |
| C1.1 | \* {($tell|$how|$what) \* засад\*} \* | *Расскажи подробнее о засаде*  *Как устроить засаду?*  *Что такое засада?* | Для того, чтобы устроить Засаду, нужно совершить проверку Ловкости (Скрытность) против пассивной Мудрости противников. Спрятаться в бою можно только в том случае, если противник тебя не видит и не слышит, или видит недостаточно хорошо. |
| C1.2 | \* {$why \* (засад\*|(невидим\*|(не видно)|(не (~видеть|~увидеть))))} \* | *Зачем устраивать засаду?*  *Какие преимущества я получу, если устрою засаду?*  *Если я невидим, какие у меня преимущества?* | Переход в состояние J3.1 |
| C1.3 | \* {$attack \* (засад\*|(невидим\*|(не видно)|(не- (~видеть|~увидеть))))} \* | *Если враг в засаде, его можно атаковать?*  *Если враг невидим, как его атаковать?*  *Если враг спрятался, можно ли его атаковать?* | Броски атаки по невидимому существу совершаются с помехой. |
| C2.1 | \* {предмет\* \* (не (~тратить|~использовать)) \* $action} \* | *Какие предметы я могу использовать, не тратя действие?*  *Какие предметы я могу использовать, не совершая действие?*  *Какие предметы я могу использовать, не используя действие?* | Ты можешь совершить любое простое действие, для которого Мастер не потребует потратить действие. Например, вынуть меч из ножен, взять предмет со стола, потушить пламя, сказать что-то товарищу. |
| C2.2 | \* {предмет\* \* $must \* $action} \* | *Для каких предметов мне нужно потратить действие?*  *Я должен потратить действие, чтобы использовать предмет?* | Если ты уже использовал один предмет и хочешь взаимодействовать со вторым, нужно потратить на это действие. Некоторые магические и другие особые предметы всегда используются действием, что указывается в их описании. В любом случае, последнее слово остается за Мастером. |
| C2.3 | \* {$possibility \* \* предмет\* \* ($more-than-one|$how-much)} \* | *Я могу взаимодействовать с двумя предметами за ход?*  *Сколько предметов я могу использовать за ход?* | Если ты уже использовал один предмет и хочешь взаимодействовать со вторым, нужно потратить на это действие.  Некоторые предметы требуют того, чтобы на них было потрачено действие. |
| C3.1 | \* {(кто|$me) \* $possibility \* $spell} \* | *Могу ли я наложить заклинание?*  *Кто может накладывать заклинания?*  *Кто может кастовать?*  *Может ли мой персонаж наложить заклинание?* | Если твой класс предполагает владение магией, ты можешь в свой ход наложить заклинание. Учти, что некоторые заклинания для наложения требуют концентрации в течение минуты или дольше. Вся необходимая информация содержится в описании заклинания. |
| C3.2 | \* {$how \* $spell} \* | *Как наложить заклинание?*  *Как скастовать заклинание?* | Информация о том, как наложить заклинание, содержится в его описании. Если при наложении требуется совершить бросок атаки, его формула: d20 + бонус мастерства + твой модификатор базовой характеристики для заклинаний |
| C3.3 | \* {$how-much \* $spell \* $action} \* | *Сколько заклинаний можно наложить за ход?*  *Сколько раз в ход я могу кастовать?* | Одно, если ты не обладаешь умением, позволяющим совершать больше действий. Некоторые заклинания накладываются бонусным действием, это указано в их описании. |
| C3.4 | \* {($what|$tell|$what-to-do) \* концентраци\*} \* | *Что такое концентрация?*  *Что будет, если концентрация прервется?*  *Что если я потерял концентрацию?* | Некоторые заклинания требуют поддержания концентрации. Если ты теряешь концентрацию, действие заклинания оканчивается. |
| C3.5 | \* {$how \* (~поддерживать|~поддержать) \* концентраци\*} \*  \* {($when|что) \* (прерва\*|прерыва\*) \* концентраци\*} \* | *Как поддерживать концентрацию?*  *Что может прервать концентрацию?*  *Когда концентрация прерывается?* | Концентрацию может прервать:   * Накладывание другого заклинания, требующего концентрацию * Получение урона (при провале сбасброска Телосложения со сложностью равной 10 или половине урона) * Недееспособность или смерть |
| C3.6 | \* {$spell \* ($attack|$success)} \* | *В каком случае при наложении заклинания нужно совершать бросок атаки?*  *Когда я кастую, нужно кидать на попадание?* | Когда ты накладываешь заклинание, бросок атаки нужно совершать, только если это указано в его описании. В противном случае считается, что заклинание срабатывает автоматически. |
| C4.1 | \* {($what|$why|$tell) \* отход\*} \* | *Что дает действие «отход»?*  *Что такое отход?*  *Расскажи об отходе* | Если ты совершаешь действие Отход, то до конца текущего хода твое перемещение не провоцирует атаки. |
| C5.1 | \* {($what|$why|$tell) \* подготов\*} \* | *Что значит «подготовка»?*  *Расскажи о действии подготовка.* | «Подготовка» позволяет совершить действие не в свой ход, а после того, как произойдет определенное событие. |
| C5.2 | \* {(какое|какие|что) \* $possibility \* подготов\*} \* | *Какие действия можно подготовить?*  *Что можно подготовить?* | Подготовить можно любое действие, в том числе перемещение, атаку и заклинание. |
| C5.3 | \* {($how|$possibility) \* подготов\* \* $action} \* | *Как подготовить действие?*  *Что сделать, чтобы подготовить действие?*  *Можно ли подготовить действие?* | Для того, чтобы подготовить действие, нужно определить, что твой персонаж будет делать и при каком условии. Например: «Если культист наступит на люк, я дёрну рычаг и открою его». |
| C5.4 | \* {($possibility|$how) \* подготов\* \* $spell} \* | *Можно ли подготовить заклинание?*  *Можно ли осуществить подготовку заклинания?* | Да, заклинание можно подготовить, но у него должно быть время накладывания «1 действие», и для его удерживания требуется концентрация. |
| C6.1 | \* {($what|$why|$tell) \* поиск\*} \*  \* {$possibility \* (~искать|~найти|~находить|поиск\*)} \* | *Расскажи о действии «поиск»*  *Зачем нужен «поиск»?* | Действие «Поиск» совершается для того, чтобы найти какой-либо предмет. |
| C6.1-a | \* {$must \* характеристик\*} \* | *Какую характеристику нужно проверить для этого?*  *Нужно ли для этого проверять характеристику?* | Для действия «Поиск» Мастер может призвать к проверке Мудрости (Внимательность) или Интеллекта (Анализ). |
| C7.1 | \* {($what|$why|$tell) \* помощ\*} \*  \* {$possibility \* (~помогать|~помочь|помощ\*)} \* | *Расскажи о действии «помощь»*  *Зачем действие помощь?* | Если ты совершишь действие «Помощь», существо, которому ты помог, до начала твоего следующего хода совершит бросок атаки или проверку характеристики с преимуществом. |
| C7.1-a | \* {$must \* характеристик\*} \* | *Какую характеристику нужно проверить для этого?*  *Нужно ли для этого проверять характеристику?* | Нет, для действия «Помощь» совершать проверку характеристики не нужно. |
| C8.1 | \* {($what|$why|$tell) \* ~рывок} \* | *Что такое рывок?*  *Зачем нужен рывок?* | Действие «Рывок» удваивает твою скорость. |
| C8.2 | \* {possibility \* ~рывок \* $obstacle} \* | *Я могу совершить рывок в труднопроходимой местности?*  *Возможен ли рывок в труднопроходимой местности?* | Да, различные эффекты, влияющие на твою скорость, складываются. |
| C8.3 | \* {$possibility \* ($go|скорост\*) \* (\*быстр\*|увелич\*)} \*  \* {($possibility|$what\_to\_do|$how) \* (ускориться|ускорени\*)} \* | *Можно ли увеличить свою скорость?*  *Можно ли передвигаться быстрее?*  *Что сделать, чтобы ускориться?*  *Как ускориться?* | Ты можешь совершить Рывок.  Переход в состояние C8.1 |
| C9.1 | \* {($what|$why|$tell) \* уклонени\*} \* | *Что такое уклонение?*  *Что дает уклонение?* | Действие «Уклонение» дает помеху всем броскам атаки по тебе и преимущество твоим спасброскам Ловкости, если ты видишь атакующего. Эти эффекты действуют до начала твоего следующего хода. |
| C9.1-а | \* {$must \* характеристик\*} \* | *Какую характеристику нужно проверить для этого?*  *Нужно ли для этого проверять характеристику?* | Нет, для действия «Уклонение» совершать проверку характеристики не нужно. |
| C9.2 | \* {$possibility \* (уклонени\*|уклониться|(уйти (от|из-под) атаки)) \* [$attack]} \* | *Могу ли я уклониться от удара?*  *Можно ли уйти из-под атаки?* | Ты можешь уклониться.  Переход в состояние C9.1 |
| D1.1 | \* {$when \* бонус\* \* $action} \* | *Когда можно совершить бонусное действие?*  *Бонусное действие совершается до или после основного?* | Бонусное действие совершается после основного. За ход нельзя совершить больше одного бонусного действия. |
| D1.2 | \* {($how-much|(какое|какие|каким\*)) \* бонус\* \* $action} \* | *Сколько у меня есть бонусных действий?*  *Какое бонусное действие я могу совершить?*  *Как узнать, какое бонусное действие я могу совершить?* | Возможность совершать бонусные действия тебе дают расовые, классовые или иные способности персонажа, которые прописаны в твоем Листе. Ты можешь обладать несколькими бонусными действиями, но за один ход можно совершить только одно. |
| D2.1 | \* {($what|$why|$tell) \* реакци\*} \* | *Что такое «реакция»?*  *Расскажи о действии реакция.* | Реакция — это мгновенный ответ на срабатывание некоего условия. Самая распространенная реакция – провоцированная атака. |
| D2.2 | \* {($how-much|$more-than-one) \* $possibility \* (реакци\*|реагир\*)} \* | *Сколько реакций можно совершить за ход?*  *Сколько раз можно среагировать до начала следующего хода?*  *Можно ли совершить две реакции сразу?* | После совершения Реакции нельзя совершить вторую до начала твоего следующего хода. |
| E1.1 | \* {$learn \* (скорост\*|$distance)} \*  \* {$learn \* $possibility \* $go} \* | *Как узнать свою скорость?*  *Где написана моя скорость?*  *Где посмотреть мою скорость?*  *Как понять, на сколько я могу переместиться?* | Твоя скорость указана на твоем Листе персонажа. Стандартная скорость Гуманоида – 30 фт. |
| E2.1 | \* {$possibility \* $go \* ($action|$attack|((по частям)|частями))} \* | *Я могу прервать перемещение действием?*  *Могу ли я переместиться, атаковать и затем переместиться еще раз?*  *Можно ли тратить перемещение по частям?* | Ты можешь прерывать перемещение. Например, если у тебя скорость 30 футов, ты можешь переместиться на 10 футов, совершить действие, а затем переместиться ещё на 20 футов. |
| E3.1 | \* {($options|$how) \* $go} \* | *Какие бывают виды перемещения?*  *Как можно перемещаться?*  *Какие есть способы передвижения?* | В бою обычно передвигаются пешком. Ты можешь совершить действие Рывок, чтобы удвоить свою скорость. В труднопроходимой местности героям приходится карабкаться, протискиваться через кусты или прокладывать дорогу в глубоком снегу. Это влияет на скорость перемещения. |
| E3.1-a | \* {$how \* [скорост\*]} \* | *Как?*  *Как это влияет на скорость перемещения?* | Переход в E4.2 |
| E3.2 | \* {(сочетать|совмещать) \* $options \* $go} \*  \* {($on-foot|$go) \* ($fly|$swim|$crawl)} \*  \* {$swim \* ($fly|$go|$on-foot|$crawl)} \*  \* {$fly \* ($go|$swim|$on-foot|$crawl)} \*  \* {$crawl \* ($fly|$swim|$go|$on-foot)} \* | *Я могу сочетать разные виды перемещения?*  *Я могу часть пути пройти пешком, а часть пролететь?*  *Я хочу часть пути проплыть, а часть пройти по земле.* | Если у тебя есть несколько скоростей, ты можешь переключаться между ними во время перемещения. При каждом переключении вычитай уже пройдённое расстояние из новой скорости. Разница покажет, на сколько ещё ты можешь переместиться. |
| E4.1 | \* {($what|$learn|$tell) \* $obstacle} \* | *Что такое «труднопроходимая местность»?*  *Как понять, что местность труднопроходимая?*  *Какая местность считается труднопроходимой?* | Любые объекты, которые затрудняют перемещение — например, щебень, кусты, крутые лестницы — создают труднопроходимую местность. Пространство другого существа, как враждебного, так и нет, тоже считается труднопроходимой местностью. |
| E4.2 | \* {$obstacle \* скорост\*} \*  \* {$difference \* $obstacle \* обычн\*} \* | *Как труднопроходимая местность влияет на скорость?*  *В труднопроходимой местности скорость меньше?*  *Как перемещение по труднопроходимой местности отличается от перемещения по обычной?* | Каждый фут перемещения по труднопроходимой местности стоит 1 дополнительный фут. |
| E5.1 | \* {$what-to-do \* $knocked-down} \* | *Что делать, если меня сбили с ног?*  *Что делать, если я упал ничком?* | На вставание требуется потратить половину скорости. В лежачем положении можно ползти; на каждый фут будет потрачено два фута скорости. |
| E5.2 | \* {$how \* (подняться|встать)} \* | *Что нужно сделать, чтобы подняться?*  *Как встать на ноги?* | На вставание требуется потратить половину скорости. |
| E5.3 | \* {($possibility|скорост\*) \* $crawl} \* | *С какой скоростью можно ползти?*  *Можно ли ползти?*  *Можно ли перемещаться ползком?* | Каждый фут перемещения ползком стоит 1 дополнительный фут. |
| E6.1 | \* {$possibility \* $space \* [$control|$position] \* (~другой|~существо|~чужой)} \* | *Я могу пройти через клетку другого существа?*  *Я могу пройти через пространство другого существа?* | Ты можешь проходить сквозь пространство невраждебных существ. Сквозь пространство враждебного существа можно пройти, только если его размер как минимум на две категории больше или меньше твоего. |
| E6.2 | \* {$possibility \* $space \* [$control|$position] \* $friend} \* | *Я могу пройти через клетку дружественного существа?*  *Я могу пройти через пространство невраждебного существа?*  *Можно ли пройти через клетку друга?*  *Я могу пройти через пространство дружественного сокомандника?*  *Я могу пройти через клетку другого игрока?* | Да, но на это придется потратить в два раза больше скорости. Также ты не можешь добровольно закончить перемещение в пространстве другого существа. |
| E6.3 | \* {$possibility \* $space \* [$control|$position] \* $foe} \* | *Я могу пройти через клетку врага?*  *Я могу пройти через пространство монстра?*  *Я могу пройти через клетку противника?*  *Я могу пройти через пространство враждебного существа?*  *Я могу пройти через клетку недружественного существа?* | Сквозь пространство враждебного существа можно пройти, только если его размер как минимум на две категории больше или меньше твоего. Если ты покидаешь досягаемость враждебного существа, ты  провоцируешь атаку. Добровольно закончить перемещение в этой области ты не можешь. |
| E6.4 | \* {$space \* $control \* (~каждый|~существо|$friend|$foe)} \* | *Какое пространство контролирует каждое существо?*  *Сколько футов контролирует каждое существо?* | Каждое существо контролирует пространство в 5 футов. |
| F1.1 | \* {($what|$how|$possibility) \* $hide} \* | *Что такое укрытие?*  *Как я могу укрыться от атак противника?*  *Можно ли использовать различные объекты для защиты?*  *Можно ли спрятаться за деревом?* | В бою можно использовать различные объекты, дающие полное или частичное укрытие. За счет укрытия ты получаешь различные преимущества. Существует несколько степеней укрытия. |
| F1.1-a | какие \*  \* $tell \* | *Какие?*  *Расскажи подробнее* | Переход в состояние F3.1 |
| F2.1 | \* {(какие|что|(за чем)) \* $possibility \* hide} \* | *Какие объекты можно использовать для укрытия?*  *Что можно использовать для укрытия?*  *За чем можно спрятаться?* | Для укрытия можно использовать стены, деревья, другие препятствия и даже других существ. Разные объекты дают разные степени укрытия. |
| F3.1 | \* {$why \* $hide} \*  \* {($tell|$what|какие) \* степен\* \* $hide} \* | *Какие преимущества дает мне укрытие?*  *Какие степени укрытия существуют?* | Есть несколько степеней укрытия: наполовину, на три четверти и полностью. О какой ты хочешь узнать? |
| F3.1-a | \* {($tell|$what|$why) \* $hide \* наполовину} \* | *Если я укрыт наполовину, что мне это дает?*  *Какие преимущества укрытия наполовину?* | Цель с укрытием на половину получает бонус +2 к КД и спасброскам Ловкости. Такое укрытие дает низкая стена, большая мебель, ствол дерева, существо. |
| \* (наполовину|перв\*) \* | *Наполовину*  *о первой* |
| F3.1-b | \* {($tell|$what|$why) \* $hide \* (на три четверти)} \* | *Если я укрыт на три четверти, что мне это дает?*  *Какие преимущества укрытия на три четверти?* | Цель с укрытием на три четверти получает бонус +5 к КД и спасброскам Ловкости. Такое укрытие дает опускная решётка, бойница в стене или широкий ствол дерева. |
| \* ((на три четверти)|втор\*) \* | *На три четверти*  *о второй* |
| F3.1-c | \* {($tell|$what|$why) \* $hide \* полностью} \* | *Если я укрыт полностью, что мне это дает?*  *Каковы преимущества полного укрытия?* | На цель с полным укрытием нельзя непосредственно нацеливать атаки и заклинания, хотя некоторые заклинания могут захватить цель областью воздействия. Цель обладает полным укрытием, если полностью скрыта за препятствием. |
| \* (полностью|трет\*) \* | *Полностью*  *о третьей*  *о последней* |
| F3.1-d | \* (всё|все|всех|всём|кажд\*) \* | *Обо всех*  *О каждом* | Цель с укрытием на половину получает бонус +2 к КД и спасброскам Ловкости. Такое укрытие дает низкая стена, большая мебель, ствол дерева, существо.  Цель с укрытием на три четверти получает бонус +5 к КД и спасброскам Ловкости. Такое укрытие дает опускная решётка, бойница в стене или широкий ствол дерева.  На цель с полным укрытием нельзя непосредственно нацеливать атаки и заклинания, хотя некоторые заклинания могут захватить цель областью воздействия. |
| G1.1 | \* {($tell|какие|какой|каким) \* $options \* $damage} \* | *Какие есть виды урона?*  *Расскажи о разновидностях урона?* | Разные атаки, заклинания и прочие эффекты причиняют разные виды урона. Среди них: дробящий, звук, излучение, кислота, колющий, некротическая энергия, огонь, психическая энергия, рубящий, силовое поле, холод, электричество, яд. |
| G1.2 | \* {$learn \* $options \* $damage} \* | *Как определить вид урона?*  *Как понять, какой вид урона причиняет атака?* | У каждого оружия и заклинания указано, какой вид урона оно причиняет. |
| G1.3 | \* {$why \* $options \* $damage} \* | *На что влияет вид урона?*  *Для чего нужно знать вид урона?* | Вид урона важен при определении уязвимости, сопротивления или иммунитета к урону. |
| G2.1 | \* {($tell|$what) \* уязвим\*} \* | *Что такое уязвимость к виду урона?*  *Расскажи об уязвимости к виду урона* | Если у существа есть уязвимость к определенному виду урона, урон этого вида для него удваивается. |
| G2.2 | \* {$learn \* $me \* уязвим\*} \*  \* {($how|$when|откуда) \* уязвим\*} \* | *Как понять, что у меня есть уязвимость?*  *Откуда появляется уязвимость?* | Уязвимость к виду атаки тебе дают твои классовые, расовые или иные особенности, прописанные в Листе персонажа. |
| G2.3 | \* {$learn \* $foe \* уязвим\*} \* | *Как понять, что у врага есть уязвимость?*  *Как узнать об уязвимостях врага?*  *Можно ли узнать об уязвимостях врага?* | Ты можешь совершить проверку Мудрости, чтобы твой персонаж догадался или вспомнил об особенностях врага. Сложность проверки подскажет Мастер. |
| G2.4 | \* {(избавиться\*|избавить\*) \* уязвим\*} \* | *Как избавиться от уязвимости?*  *Я могу избавиться от уязвимости?* | От уязвимости нельзя избавиться, это черта твоего персонажа. |
| G3.1 | \* {($tell|$what|$how) \* сопротивл\*} \* | *Что такое сопротивление к виду урона?*  *Расскажи о сопротивлении к виду урона* | Если у существа есть сопротивление к определенному виду урона, урон от такой атаки уменьшается вдвое. |
| G3.2 | \* {$learn \* $me \* сопротивл\*} \*  \* {($how|$when|откуда) \* сопротивл\*} \* | *Как понять, что у меня есть сопротивление?*  *Откуда появляется сопротивление?* | Сопротивление к виду атаки тебе дают твои классовые, расовые или иные особенности, прописанные в Листе персонажа. |
| G3.3 | \* {$learn \* $foe \* сопротивл\*} \* | *Как понять, что у врага есть сопротивление?*  *Как узнать о сопротивлении врага к урону?*  *Можно ли узнать о сопротивлении врага?* | Ты можешь совершить проверку Мудрости, чтобы твой персонаж догадался или вспомнил об особенностях врага. Сложность проверки подскажет Мастер. |
| G4.1 | \* {($tell|$what|$how) \* иммунитет\*} \* | *Что такое иммунитет к виду урона?*  *Расскажи об иммунитете к виду урона* | Если у существа есть иммунитет к определенному виду урона, оно не получает урона от такой атаки. |
| G4.2 | \* {$learn \* $me \* иммунитет\*} \*  \* {($how|$when|откуда) \* иммунитет\*} \* | *Как понять, что у меня есть иммунитет?*  *Откуда появляется иммунитет?* | Иммунитет к виду атаки тебе дают твои классовые, расовые или иные особенности, прописанные в Листе персонажа. |
| G4.3 | \* {$learn \* $foe \* иммунитет\*} \* | *Как понять, что у врага есть иммунитет?*  *Как узнать об иммунитете врага?*  *Можно ли узнать об иммунитете врага к виду урона?* | Ты можешь совершить проверку Мудрости, чтобы твой персонаж догадался или вспомнил об особенностях врага. Сложность проверки подскажет Мастер. |
| H1.1 | \* {($how|$possibility) \* $heal} \*  \* {$what-to-do \* $damage} \* | *Как восстановить хиты?*  *Как восстановить здоровье?*  *Как быть, если я потерял хиты? Как лечиться?*  *Можно ли лечиться?* | Во время боя хиты можно восстанавливать с помощью лечащих заклинаний и зелий лечения. |
| H1.2 | \* {($what|$how) \* зелье \* $heal} \* | *Что такое зелье лечения?*  *Как работает зелье лечения?* | Зелье лечения – магическая красная жидкость, которое восстанавливает 2к4 + 2 хитов. Зелье выпивается или заливается в рот другому существу действием. |
| H1.3 | \* {$where \* зелье \* $heal} \* | *Где взять зелье лечения?*  *Где можно купить зелье лечения?* | Зелье лечения можно приобрести в лавках магических снадобий, которые есть в каждом крупном городе (но лучше уточни у Мастера). Один флакон зелья обойдется примерно в 50 золотых. |
| H1.4 | \* {($tell|какое|какие) \* заклинани\* \* $heal} \* | *Какие заклинания восстанавливают хиты?*  *Какие заклинания лечат?*  *Какие заклинания являются лечебными?* | К лечащим заклинаниям относятся Лечение ран, Лечащее слово, Молебен лечения, Аура живучести, Полное исцеление и другие. |
| H1.4-a | \* {$tell \* [подробнее|больше]} \*  \* {($how|$tell)} \* $heal-magic} \* | *Расскажи подробнее о Лечении ран*  *Как наложить Ауру живучести?* | Прости, приятель, я демо-версия и не знаю всего. Загляни в Книгу Игрока! |
| H1.5 | \* {($how|$when|$possibility) \* ($heal|$undo) \* $state} \*  \* {$what-to-do \* $spell} \* | *Как снять эффект от заклинания?*  *Как окончить состояние?*  *Как вылечить болезнь?*  *Как вернуть зрение?*  *Когда заканчивается состояние?*  *Что делать если меня заколдовали?* | Состояние длится до тех пор, пока не будет прервано, или в течение времени, которое указано в эффекте, вызвавшем это состояние. Некоторые состояния можно снять с помощью заклинаний. Например, Малое восстановление может вылечить глухоту, отравление, паралич или слепоту. |
| H2.1 | \* {$what-to-do \* $health \* (0|нол\*|нул\*)} \* | *Что, если хиты опустились до 0 или ниже?*  *Что происходит, когда хиты опускаются до нуля?* | Если твои хиты опустились до нуля, ты теряешь сознание. Если хиты опустились ниже нуля на значение, равное максимуму твоих хитов, наступает мгновенная смерть. |
| H2.2 | \* {$how \* $save} \* | *Как я могу вернуть существо в сознание?*  *Как я могу спасти от смерти?*  *Как я могу привести в чувство?* | Чтобы стабилизировать существо, необходимо совершить проверку Мудрости (Медицина) со Сл 10, или наложить соответствующее заклинание (например, Уход за умирающим). |
| H2.3 | \* {$how \* $save-yourself} \*  \* {($tell|$what|$how) \* (спасброс\* от смерти)} \* | *Расскажи о спасбросках от смерти*  *Как прийти в сознание?*  *Как работают спасброски от смерти?* | Если ты начинаешь ход с 0 хитов, ты должен совершить особый спасбросок, называемый спасброском от смерти. Каждый ход бросай d20; выпало 10 или больше – «успех», меньше – «провал». Чтобы прийти в сознание, нужно совершить три успешных броска. После трех провалов наступает смерть. |
| H2.4 | \* {$what-to-do \* $unconscious \* ($damage|$attack)} \* | *Что будет, если существу с 0 хитами нанести урон?*  *Что случится, если существу без сознания нанести урон?* | Существо автоматически получает два «провала» спасброска от смерти. Если урон равен максимальному количеству хитов существа, оно умирает. |
| H3.1 | \* {($how|$possibility) \* воскрес\*} \* | *Можно ли воскресить существо?*  *Как воскресить существо?* | Существо можно воскресить с помощью заклинаний Оживление, Воскрешение, Истинное воскрешение или Исполнение желаний. Если никто в отряде не владеет одним из них, вы можете отыскать волшебника в крупном городе – спросите об этом Мастера. Учтите, что такая магия обойдется вам очень недешево. |
| I1.1 | \* {($what|$learn) \* $initiative} \* | *Как определить, кто за кем ходит?*  *Что такое инициатива?*  *Что такое инициатива в бою?*  *Как определить порядок ходов?* | Инициатива определяет порядок совершения ходов во время сражения. Чтобы вычислить значение твоей инициативы, брось d20 и прибавь свой модификатор Ловкости. |
| I1.2 | \* {$formula \* $initiative} \* | *Как посчитать инициативу?*  *Как вычислить инициативу?*  *Скажи формулу, по которой вычисляется инициатива* | d20 + модификатор Ловкости |
| I1.3 | \* {$equality \* $initiative} \* | *Что делать, если у двух человек одинаковое значение инициативы?*  *Если совпадают инициативы, что делать?* | Если возникает ничья, порядок определяет Мастер. В качестве альтернативы можно бросить d20. |
| I2.1 | \* {($tell|$what) \* $suddenness} \* | *Что значит «захватить врасплох»?*  *Расскажи об элементе неожиданности в сражении* | Если ни одна из сторон не старалась вести себя тихо, группы автоматически замечают друг друга. В противном случае Мастер сравнивает проверки Ловкости (Скрытность) тех, кто прячется, с пассивным значением Мудрости (Внимательность) противоположной стороны. Все персонажи и чудовища, не заметившие угрозы, в начале сцены захвачены врасплох. Они пропускают свой первый ход. |
| I2.2 | \* {($how|$learn) \* $suddenness} \* | *Как захватить (врага) врасплох?*  *Как понять, смогли ли герои захватить врагов врасплох?* | Чтобы захватить противника врасплох, нужно совершить проверку Ловкости (Скрытность). Мастер сравнит выпавшие значения с пассивным значением Мудрости (Внимательность) врага; если первое больше второго, вы преуспели. |
| I2.3 | \* {($what-to-do|$why) \* $suddenness} \* | *Что, если меня захватили врасплох?*  *Что дает захват врасплох?*  *Какие преимущества получают герои, заставшие противников врасплох?* | Все персонажи, захваченные врасплох, в свой первый ход не могут перемещаться или совершать действия. |
| J1.1 | \* {($what|$tell|происходит\*) \* $unconscious} \* | Расскажи о состоянии бессознательный.  Что значит «бессознательный»?  Что происходит с существом без сознания? | Находящееся без сознания существо автоматически проваливает спасброски Силы и Ловкости; броски атаки по существу совершаются с преимуществом. Любая атака, попавшая по такому существу, считается критическим попаданием, если нападающий находится в пределах 5 футов от него. |
| J2.1 | \* {($what|$tell|происходит\*) \* (испуг\*|напуг\*)} \* | Расскажи о состоянии испуганный.  Что значит «напуганный»?  Что происходит с испуганным существом? | Испуганное существо совершает с помехой проверки характеристик и броски атаки, пока источник его страха находится в пределах его линии обзора. Существо не способно добровольно приблизиться к источнику своего страха. |
| J2.2 | \* {($what-to-do|$undo) \* (испуг\*|напуг\*)} \* | Как окончить состояние испуганный?  Что делать, если меня напугали? | Переход в состояние H1.5 |
| J3.1 | \* {($what|$tell|происходит\*) \* (невидим\*|(не видно)|(не (~видеть|~увидеть)))} \* | Расскажи о состоянии невидимый.  Что значит «невидимый»? | Невидимое существо невозможно увидеть без помощи магии или особого чувства. С точки зрения скрытности существо считается сильно заслонённым. Местонахождение существа можно определить по шуму, который оно издаёт, или по оставленным им следам. Броски атаки по невидимому существу совершаются с помехой, а его броски атаки — с преимуществом. |
| J4.1 | \* {($what|$tell|происходит\*) \* недееспособн\*} \* | Расскажи о состоянии недееспособный.  Что значит «недееспособный»?  Что происходит с недееспособным существом? | Недееспособное существо не может совершать действия и реакции. |
| J4.2 | \* {($what-to-do|$undo) \* недееспособн\*} \* | Как окончить состояние недееспособный?  Что делать, если я стал недееспособным? | Переход в состояние H1.5 |
| J5.1 | \* {($what|$tell|происходит\*) \* $deaf} \* | Расскажи о состоянии оглохший.  Что значит «оглохший»?  Что происходит с оглохшим существом? | Оглохшее существо ничего не слышит и автоматически проваливает все проверки характеристик, связанные со слухом. |
| J5.2 | \* {($what-to-do|$undo) \* $deaf} \* | Как окончить состояние оглохший?  Что делать, если меня оглушили? | Переход в состояние H1.5 |
| J6.1 | \* {($what|$tell|происходит\*) \* $petrified} \* | Расскажи о состоянии окаменевший.  Что значит «окаменевший»?  Что происходит с окаменевшим существом? | • Окаменевшее существо, а также все немагические предметы, которые оно несёт или носит, трансформируется в камень. Его вес увеличивается в 10 раз, и оно перестаёт стареть.  • Оно «недееспособно», не способно двигаться и говорить, а также не осознаёт своё окружение.  • Броски атаки по существу совершаются с преимуществом.  • Оно автоматически проваливает спасброски Силы и Ловкости.  • Оно получает сопротивление ко всем видам урона.  • Существо получает иммунитет к ядам и болезням. Если яд или болезнь уже действовали на существо, их действие приостанавливается, но не исчезает. |
| J6.2 | \* {($what-to-do|$undo) \* $petrified} \* | Как окончить состояние окаменевший?  Что делать, если на меня наложили заклинание окаменения? | Переход в состояние H1.5 |
| J7.1 | \* {($what|$tell|происходит\*) \* опута\*} \* | Расскажи о состоянии опутанный.  Что значит «опутанный»?  Что происходит с опутанным существом? | Скорость опутанного существа равна 0, и оно не получает выгоды ни от каких бонусов к скорости. Броски атаки по такому существу совершаются с преимуществом, а его броски атаки — с помехой. Существо также совершает с помехой спасброски Ловкости. |
| J7.2 | \* {($what-to-do|$undo) \* опута\*} \* | Как окончить состояние опутанный?  Что делать, если меня опутали? | Переход в состояние H1.5 |
| J8.1 | \* {($what|$tell|происходит\*) \* $blind} \* | Расскажи о состоянии ослепленный.  Расскажи о состоянии ослеплённый  Что значит «ослепленный»?  Что происходит с ослеплённым существом? | Ослеплённое существо ничего не видит и автоматически проваливает все проверки характеристик, связанные со зрением.  Броски атаки по такому существу совершаются с преимуществом, а его броски атаки совершаются с помехой. |
| J8.2 | \* {($what-to-do|$undo) \* $blind} \* | Как окончить состояние ослепленный?  Что делать, если меня ослепили? | Переход в состояние H1.5 |
| J9.1 | \* {($what|$tell|происходит\*) \* (отравлен\*|отравил\*)} \* | Расскажи о состоянии отравленный.  Что значит «отравленный»?  Что происходит с отравленным существом? | Отравленное существо совершает с помехой броски атаки и проверки характеристик. |
| J9.2 | \* {($what-to-do|$undo) \* (отравлен\*|отравил\*) } \* | Как окончить состояние отравленный?  Что делать, если меня отравили? | Переход в состояние H1.5 |
| J10.1 | \* {($what|$tell|происходит\*) \* очарова\*} \* | Расскажи о состоянии очарованный.  Что значит «очарованный»?  Что происходит с очарованным существом? | Очарованное существо не может атаковать того, кто его очаровал, а также делать его целью умения или магического эффекта, причиняющего вред. Искуситель совершает с преимуществом все проверки характеристик при социальном взаимодействии с очарованным существом. |
| J10.2 | \* {($what-to-do|$undo) \* очарова\*} \* | Как окончить состояние очарованный?  Что делать, если меня очаровали? | Переход в состояние H1.5 |
| J11.1 | \* {($what|$tell|происходит\*) \* $stunned} \* | Расскажи о состоянии ошеломленный.  Что значит «ошеломлённый»?  Что происходит с ошеломленным существом? | Ошеломлённое существо «недееспособно», не способно перемещаться и говорит, запинаясь. Оно автоматически проваливает  спасброски Силы и Ловкости, а броски атаки по существу совершаются с преимуществом. |
| J11.2 | \* {($what-to-do|$undo) \* $stunned} \* | Как окончить состояние ошеломленный?  Что делать, если меня ошеломили? | Переход в состояние H1.5 |
| J12.1 | \* {($what|$tell|происходит\*) \* парализова\*} \* | Расскажи о состоянии парализованный.  Что значит «парализованный»?  Что происходит с парализованным существом? | • Парализованное существо «недееспособно» и не способно перемещаться и говорить.  • Оно автоматически проваливает спасброски Силы и Ловкости.  • Броски атаки по парализованному существу совершаются с преимуществом.  • Любая атака, попавшая по нему, считается критическим попаданием, если нападающий находится от него в пределах 5 футов. |
| J12.2 | \* {($what-to-do|$undo) \* парализова\*} \* | Как окончить состояние парализованный?  Что делать, если меня парализовали? | Переход в состояние H1.5 |
| J13.1 | \* {($what|$tell|происходит\*) \* $knocked-down} \* | Что значит «лежащий ничком»?  Что происходит с существом, которого сбили с ног?  Что происходит с существом, которое лежит ничком? | Сбитое с ног существо способно перемещаться только ползком, пока не встанет. Оно совершает с помехой броски атаки, а по нему броски атаки совершаются с преимуществом, если нападающий находится в пределах 5 футов от него. В противном случае броски атаки совершаются с помехой. |
| J14.1 | \* {($what|$tell|происходит\*) \* $capture} \* | Расскажи о состоянии схваченный.  Что значит «схваченный»?  Что происходит со схваченным существом? | Скорость схваченного существа равна 0, и оно не получает выгоды ни от каких бонусов к скорости. Состояние оканчивается, если схвативший становится недееспособен или если какой-либо эффект выводит схваченное существо из зоны досягаемости того, кто его удерживает, или из зоны удерживающего эффекта. |

***Приложение 4***

$action = (~делать|~сделать|действ\*|~ход)

$attack = (атак\*|удар\*|~поразить)

$blind = (зрени\*|слепот\*|ослепш\*|ослеплен\*|ослеплён\*|ослепил\*)

$bye = (пока|прощай\*|пока-пока|(до свидания)|(добро\* (дня|утра|вечера|ночи))|\*-бай|споки\*)

$capture = (~схватить|~захватить|захват\*|\*схваченн\*|обездвиж\*|~поймать)

$control = (~занимать|~занять|~контролировать)

$crawl = (~ползти|~проползти|~ползать|ползком|(на коленях))

$crit\_failure = (1|критпровал\*|(критическ\* $failure))

$crit\_success = (20|критуспех\*|(критическ\* $success))

$damage = (~урон|отня\*|сня\*|~рана|ранени\*|потеря\*)

$deaf = (оглохш\*|оглушен\*|оглушить|оглушил\*|слух|слышать)

$difference = (разниц\*|различ\*|отлич\*)

$distance = (дистанци\*|расстояни\*)

$dnd = (dnd|днд|(dungeons (and|&) dragons)|dungeons&dragons|(подземель\* и дракон\*)|~игра)

$equality = (одинаков\*|совпаден\*|~совпадать|~совпасть|ничь\*|равн\*)

$failure = (промах\*|неудачн\*|неуспешн\*|провал\*)

$fight = (сраж\*|сраз\*|бит\*|~бой|драк\*|~драться|~подраться)

$fly = (~лететь|~пролететь|~летать|(по воздуху))

$foe = (враг\*|противник\*|монстр\*|недруг\*|враждебн\*|недружествен\*|чудовищ\*|(не $friend))

$formula = (~считать|~посчитать|вычисл\*|формул\*|кинуть|бросить)

$friend = (~друг|товарищ\*|дружествен\*|невраждебн\*|коллег\*|сокомандник\*|однопарти\*|игрок\*|(не $foe))

$go = (~идти|~ходить|~пройти|~проходить|~перемещаться|~передвигаться|~двигаться|~пересечь|~пересекать|\*движени\*|перемещени\*)

$got\_it = (понятно|понял\*|ясно|ясненько)

$hand\_to\_hand = (ближн\*|\*рукопашн\*|вплотную|(лицом к лицу)|(в пределах досягаемости)|(в пространстве противника))

$heal = (~лечить|~лечиться|вылечить|лечебн\*|лечен\*|восстанавл\*|восстанов\*)

$heal\_magic = ((~лечение ран)|(~лечащее ~слово)|(~молебен лечения)|(~аура живучести)|(~полное ~исцеление))

$health = (хит\*|здоровь\*)

$hi = (привет\*|здравствуй\*|здоров\*|даров\*|(добр\* (день|вечер|утро))|(доброго времени суток))

$hide = (укрыт\*|укрыв\*|прятаться|спрятаться|спрятал\*|прячет\*|защит\*)

$how = (как|(каким образом)|принцип\*|механик\*|(что \* чтобы))

$how\_much = (сколько|количеств\*|(как много))

$initiative = (инициатив\*|поряд\*|очередност\*|очерёдност\*|(кто за кем)|(кто когда))

$knock\_down = ((сбить \* с ног)|свалить|повалить|опрокинуть|толкнуть|столкнуть|толкать|толкани\*)

$knocked\_down = (ничком|~упасть|((~сбитый|сбит|сбил\*) с ног)|опрокинул\*|повалил\*)

$learn = (узнать|прочитать|посмотреть|понять|определить|написан\*|указан\*|информаци\*|сведени\*)

$long\_range = (дальн\*|дальнобойн\*|издалека|лук\*|арбалет\*|(с расстояния)|\*стрел\*|~метать|~метнуть|брос\*|кину\*|швырну\*|швыря\*)

$me = (мой|моего|моему|моим|моем|моём|мои|моих|моим|моими|меня|мне|мной|свой|своего|своему|своим|своем|своём)

$more\_than\_one = (два|две|двух|двум|двумя|несколько|нескольких|нескольким|несколькими)

$move = (~перемещать|~двигать|~передвигать|~тащить|~таскать|~перетаскивать|~нести)

$must = (должен|должна|необходимо|нужно|обязательно|обязан\*)

$no = (нет|(не (надо|хочу|стоит|нужно)))

$no\_questions = ({(не|нет|\*кончились) \* вопрос\*}|ничего|никаких)

$obstacle = (труднопроходим\*|~щебень|~песок|куст\*|подлес\*|лес\*|снег\*|болот\*|{(лестниц\*|ступен\*) крут\*})

$on\_foot = (пешком|(по земле)|шагом)

$options = (вид\*|вариант\*|разновидност\*|опци\*|способ\*)

$out\_of\_rules = (импровизирова\*|((нет|не) \* в правилах)|~другой|альтернативн\*)

$petrified = (~камень|окамене\*)

$position = (~стоять|~находиться|позици\*|положени\*)

$possibility = (~мочь|могу|\*можно|возможн\*|возможен|разрешено|позволено|~разрешать|~позволять)

$save = (спасти|((вернуть|привести) \* в (сознание|чувство)))

$save\_yourself = (спастись|((вернуться|прийти|придти) в (сознание|чувство)))

$space = (пространств\*|клетк\*|территори\*|фут\*)

$spell = (заклина\*|спел\*|маги\*|\*колдов\*|кастовать|касту\*)

$state = (состояни\*|болезн\*|эффект\*)

$steps = (~шаг|пошагово|~этап|поэтапно|~состоять|~порядок)

$stunned = (ошеломлен\*|ошеломлён\*|ошеломить|ошеломил\*)

$success = (успех\*|успешн\*|попа\*|удачн\*)

$suddenness = (((захват\*|застать|застал\*|заставши\*) \* врасплох)|неожиданн\*|внезапн\*)

$swim = (~плыть|~проплыть|~плавать|((по|в) воде))

$tell = ((\*скажи\*|поведай\*) \* (о|об))

$thanks = (спасиб\*|благодар\*|спс)

$unconscious = (((0|нол\*|нул\*) хит\*)|(без сознания)|бессознательн\*)

$undo = (\*кончи\*|\*канчи\*|вернуть|снять|избавиться)

$what = ((что (такое|значит|это))|(считается|называется|называют|является))

$what\_to\_do = ((что \* если)|(что, если)|(как быть\*))

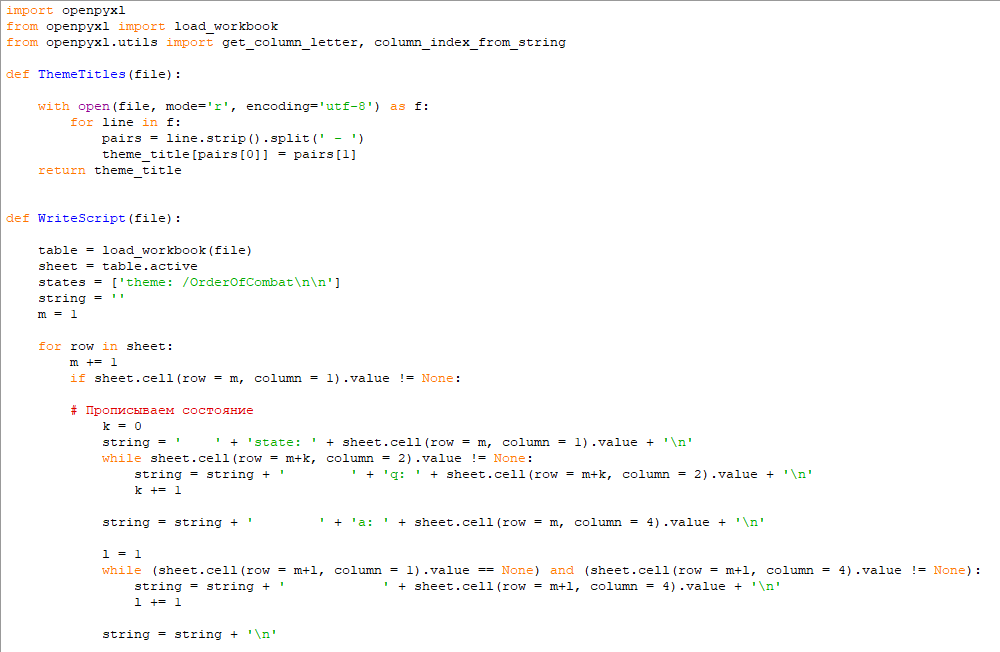
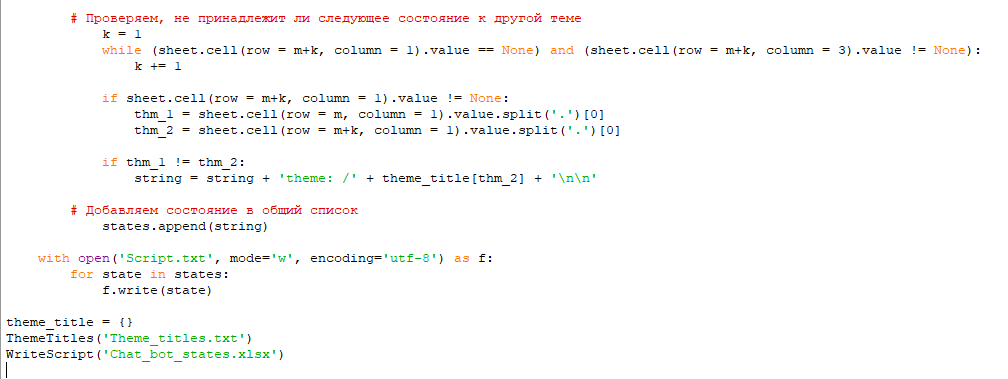
$when = (почему|когда|(в каком случае)|(до или после))

$where = (где|откуда|(у кого))

$why = (зачем|(что \* (дает|даёт))|(для чего)|преимуществ\*|бонус\*|смысл\*|~влияние|~влиять|~повлиять)

$yes = (да|давай\*|конечно|валяй|хорошо|ладно|ок|окей)

***Приложение 5***



***Приложение 6***

