GeekBrains

Дипломный проект:
"Разработка интерактивного игрового теста для детей “Квиз Сказок”"

IT-специалисты:
Frontend-программисты
Аракелян В.Р.
Артемьев Н.П.
Денисова А.А.
Рузанов А.А.

Москва
2024

Оглавление

1. Введение

1.1. Постановка проблемы и актуальность темы.

Современное образование стремится к инновационным методам обучения, однако существует необходимость в создании интерактивных инструментов, способствующих развитию у детей умения решать задачи, развивать логическое мышление и обучаться с удовольствием. Существующие формы контроля знаний часто не учитывают индивидуальные особенности каждого ученика, что может привести к недостаточной мотивации и эффективности обучения.

Актуальность данного проекта обусловлена потребностью в создании игрового теста, специально адаптированного для школьников, который не только проверяет их знания, но и стимулирует интерес к обучению. Разработка такого приложения влечет за собой возможность эффективного использования современных технологий в образовательном процессе и поддерживает принципы индивидуализации обучения.

С развитием цифровых технологий и доступностью смартфонов и планшетов, интерактивные обучающие приложения становятся все более популярными среди детей. Однако, несмотря на это, существует недостаток качественных игровых тестов, способных эффективно привлечь и удержать внимание молодого поколения.

1.2. Цели и задачи проекта

Цель проекта состоит в создании инновационного игрового теста, который не только оценивает уровень знаний учеников, но и способствует их активному обучению, развитию критического мышления и умения работать в команде. Проект нацелен на внедрение интерактивных методов обучения в школьную практику, повышение мотивации учащихся и формирование у них интереса к предметам.

Задачи проекта включают в себя:

- Разработка технического задания, определяющего функциональные и нефункциональные требования к игровому тесту;
- Определение целевой аудитории и ее особенностей для эффективного адаптирования контента;
- Проектирование системы, включающее в себя выбор технологического стека, инструментов и сервисов;

- Реализация ключевых функций приложения, обеспечивающих его плавную и удобную работу;
- Анализ достигнутых результатов с целью оценки эффективности проекта и выявления возможных улучшений.

2. Основная часть

2.1. Теория

2.1.1. Разработка технического задания, функциональных и нефункциональных требований

Первоначальный этап разработки включал в себя составление технического задания (ТЗ), в котором определялись ключевые параметры и требования к будущему интерактивному игровому тесту для детей. В рамках ТЗ были обозначены основные требования к приложению, такие как:

- Описание игрового процесса, механика игры, включая логику взаимодействия с игроком и алгоритм обработки ответов. По задумке игровой тест представляет собой одностраничное приложение (SPA), в котором последовательно меняются сцены, в который происходит то или иное взаимодействие с игроком:
  - перед началом квиза игроку предлагается выбрать персонаж, которым он бы хотел играть (представить два варианта на выбор мальчик/девочка, далее изображения персонажей будут появляться в той или иной сцене теста);
  - далее у игрока просят ввести имя/ник. Это имя будет использование в игре в диалогах для обращения к игроку;
  - в рамках игры с игроком общается рассказчик (вариант "бабушка-рассказчица" и "кот ученый") - общение должно быть оформлено в виде диалогового окна, на которой печатается то или иное повествование рассказчика (использовать эффект печатной машинки, после напечатания текста разговора рассказчика должны появляться кнопки-ответы для выбора игроком). Рассказчик вовлекает игрока в игру путем постановки задачи, которую нужно выполнить (собрать все упавшие страницы из сказок, которые перепутались и т.п.), а также мотивируя игрока при ответе на тот или иной вопрос квиза, и для повышения интереса, рассказывая о различных интересных, увлекательных и познавательных фактах касательно темы игры;
  - в промежутках между "общением" с рассказчиком игроку представляется сцена с вопросом теста. В этой сцене, вместо изображения рассказчика и персонажа игры,

должно появиться изображение, связанное с вопросом теста. Отрисовка вопроса теста аналогична отрисовке диалога рассказчика. Позиционирование кнопок-ответов на вопросы теста сделать под вопросом теста;

- выбор ответа игрока должен запоминаться приложением, - индекс кнопки-ответа, которую выбрал для нажатия игрок заносится в массив ответов игрока. Далее массив ответов игрока сравнивается с массивом верных ответов теста, и на финальной странице должно выводиться обращение к игроку и итог его игры (количество правильных ответов из общего числа вопросов);
- кроме отрисовки рассказчика, персонажа, диалогов, кнопок и экшн изображений, в каждой сцене есть фоновое изображение, которое меняется по ходу игры (использовать несколько вариантов);
- в каждой сценке игры должны быть стационарные кнопки для получения справки и принудительного выхода из игры. По нажатии на одну из кнопок, текущая сцена игры должна затемняться и уходить на второй план, на первом плане должно появиться диалоговое окно с соответствующим текстом: окно справки с текстом-пояснением, что требуется сделать игроку, и окно-предупреждение о закрытии игры и потере прогресса, с дополнительными кнопками "подтвердить" и "закрыть/отменить". При этом, клик игрока вне диалогового окна, приводит к его закрытию;
- финальная страница с результатами игры должна иметь кнопки "завершить" для корректного выхода из приложения, и "начать с начала" для возврата к игре тем же персонажем и введенным именем, но с обнуленным прогрессом предыдущих результатов игры.

- Визуальное оформление и интерфейс приложения, с акцентом на детскую аудиторию.
  Для визуального стиля предполагается сгенерировать изображения для фона, экшн изображений, персонажей и рассказчиков с помощью сервиса Midjourney. Все изображения должны быть выдержаны в единой стилистике, акцентированной на тему игрового теста, а также на аудиторию теста (все элементы игрового теста должны быть привлекательными для целевой аудитории, яркими).
  Хороший интерфейс приложения - это не только красивый дизайн, но и удобство использования для пользователя. Вот несколько основных принципов, которые помогают создать эффективный и приятный интерфейс:
  - Простота и ясность:
    Интерфейс должен быть интуитивно понятным для пользователя. Простые, легко

узнаваемые элементы управления, понятные навигационные пути и минималистичный дизайн способствуют легкости взаимодействия.

- Согласованность:

  Все элементы интерфейса должны быть согласованы между собой: шрифты, цветовая гамма, стили кнопок и другие дизайнерские аспекты. Это создает целостное визуальное впечатление и улучшает восприятие приложения.

- Назначенность:

  Каждый элемент интерфейса должен иметь четкое назначение и быть функционально обоснованным. Необходимо избегать избыточности и перегруженности информацией, делая интерфейс более удобным и понятным для пользователя.

- Отзывчивость:

  Отзывчивость интерфейса очень важна для удовлетворения пользовательских потребностей. Быстрая обратная связь на действия пользователя, плавные анимации, отсутствие задержек – все это способствует позитивному восприятию приложения.

- Доступность:

  Интерфейс должен быть доступен для целевой аудитории, в том числе для различных устройств: адаптивный дизайн, удобство использования на разных устройствах и средствах.

  Соблюдение этих принципов не только улучшает пользовательский опыт, но и делает приложение более успешным и конкурентоспособным на рынке. Более подробно данный пункт описан в п. 2.1.4 ниже.

- Тема игрового теста.

  В качестве тематики игры выбрана тема русских народных сказок.

  Русские народные сказки – это настоящее кладезь не только веселых и увлекательных историй, но и важных уроков и мудрости. Вот почему они так привлекательны для детей:

  - Фантазия и Воображение:

    Русские сказки наполнены чудесами, волшебством и фантастическими сюжетами, что развивает воображение ребенка и помогает расширять его кругозор.

  - Уроки Морали и Жизни:

    В сказках часто заложены важные уроки о дружбе, справедливости, доброте, труде и других ценностях. Через приключения героев дети учатся различать хорошее от плохого и принимать правильные решения.

- Интрига и Сюжет: Напряженный сюжет, неожиданные повороты событий и захватывающие приключения делают русские сказки увлекательными и заинтересовывают детей.
- Традиции и Культура: Через русские сказки дети знакомятся с традициями и культурой своей страны, узнают о русских обычаях, народных героях и мудрости предков.
- Образцы Поведения: Персонажи сказок – это образцы поведения, которые помогают детям сформировать свой характер, развивать чувство ответственности и эмпатию.
- Семейное Время: Тема сказок создает уютную атмосферу семейного времяпрепровождения, когда родители и дети могут вместе наслаждаться и обсуждать происходящее в историях. Итак, русские сказки не только увлекательны и интересны, но и играют важную роль в развитии ребенка, помогая формировать его ценности, фантазию, моральные принципы и культурное наследие.

- Необходимые технологии и инструменты для реализации задуманных функций:
  - HTML, CSS, JavaSript - стандартный инструментарий для разработки такого рода интерактивных приложений;
  - Visual Studio Code, а также расширение Liveshare - для совместного написания кода разработчиками;
  - Midjourney, Photoshop - приложения для генерирования и редактирования изображений, используемых в игре;
  - Github - платформа для сохранения всех этапов разработки и финального кода;
  - Google Meet - платформа для совместных звонков и встреч для обсуждения работы;
  - Google Drive - облачное хранилище текстов и наработок по тематике. Более подробно технологический стек/инструменты описаны в п. 2.1.5 и 2.1.6 ниже.

- Требования к скорости работы приложения, его адаптивность, интуитивность.

  Скорость работы: Приложение должно предусматривать предзагрузку всех изображений перед стартом игры.

  Предзагрузка изображений в веб-приложениях или на мобильных устройствах - это процесс загрузки изображений заранее, до того, как они понадобятся для отображения на странице или в приложении. Этот подход имеет несколько преимуществ и обеспечивает

оптимизацию производительности и пользовательского опыта:

Преимущества предзагрузки изображений:

- Улучшение скорости загрузки: Предварительная загрузка изображений позволяет уменьшить время ожидания для пользователя, так как изображения уже находятся в кеше браузера или памяти устройства;
- Улучшение производительности: Предзагруженные изображения не требуют загрузки в момент отображения, что снижает нагрузку на сервер и сеть, улучшая производительность приложения;
- Предотвращение задержек: Предварительная загрузка позволяет избежать миганий или задержек при загрузке изображений во время переходов между разделами приложения;
- Улучшение качества изображений: Предварительное кеширование изображений обеспечивает возможность предварительной обработки и оптимизации изображений для подготовленного использования;
- Лучший пользовательский опыт: Быстрая загрузка изображений благоприятно сказывается на впечатлении пользователя о приложении, делая его более привлекательным и удобным.

Адаптивность приложения – это способность приложения автоматически подстраиваться под различные условия и параметры, такие как разрешение экрана, устройство, ориентация экрана, размер окна браузера и другие характеристики, для обеспечения оптимального пользовательского опыта на любом устройстве или в любых условиях.

Ключевые аспекты адаптивности приложения:

- Отзывчивый дизайн: Адаптивное приложение имеет отзывчивый дизайн, который позволяет контенту гибко изменяться и масштабироваться в зависимости от размера экрана устройства;
- Гибкость интерфейса: Интерфейс приложения должен быть гибким и адаптивным, так чтобы пользователи могли удобно взаимодействовать с приложением независимо от используемого устройства;
- Оптимизация производительности: Адаптивное приложение должно быть оптимизировано для работы на различных устройствах, обеспечивая плавную работу и быструю загрузку контента;
- Управление ресурсами: Приложение должно уметь эффективно использовать ресурсы устройства, адаптируя свою работу в зависимости от характеристик устройства;

- Кросс-платформенность: Адаптивность также означает способность приложения работать на различных платформах и устройствах без потери функциональности и качества пользовательского опыта;
- Тестирование и оптимизация: Важным аспектом адаптивности является тестирование приложения на различных устройствах и разрешениях экрана для обеспечения корректного отображения и поведения приложения.

Зачем нужна адаптивность приложения:

Адаптивность приложения позволяет обеспечить удобство использования и качественный пользовательский опыт на любом устройстве, увеличивая охват аудитории и повышая удовлетворенность пользователей. Такая гибкость приложения помогает привлекать новых пользователей, удерживать существующих и улучшать общую конкурентоспособность приложения на рынке.

Интуитивность приложения отражает насколько легко и понятно пользователю взаимодействовать с ним без необходимости изучения инструкций или подробного обучения. Когда приложение интуитивно понятно, пользователь способен найти нужный функционал, понимать как им пользоваться, и достигать своих целей без лишних усилий или запутывания.

Ключевые аспекты интуитивности приложения:

- Понятная навигация: Хорошее приложение имеет логичную структуру и навигацию, позволяющую пользователям легко ориентироваться и находить нужную информацию или функции;
- Ясные элементы управления: Интуитивное приложение использует понятные и обычно принятые элементы управления (кнопки, иконки, меню и т.д.), что помогает пользователям быстро понять их назначение.
- Естественное поведение: Приложение должно следовать привычным для пользователей сценариям использования, как будто они взаимодействуют с реальными объектами;
- Контекстуальная обратная связь: Приложение должно предоставлять понятную обратную связь на действия пользователя, помогая понять результаты своих действий;
- Минималистичный дизайн: Чистый и минималистичный дизайн способствует облегчению интерфейса и сосредотачивает внимание пользователя на основном функционале;
- Тестирование с пользователем: Проведение тестирования с реальными пользователями помогает выявить слабые места в интуитивности приложения и внести улучшения;

Интуитивность приложения играет важную роль в создании позитивного пользовательского опыта, повышает удовлетворенность пользователя и способствует успешности приложения на рынке.

- Масштабируемость проекта и иные возможности.

  В части дополнительных возможностей можно рассмотреть следующие сценарии:
  - доработка различных опций игры (таймер, подсказки, больше анимации и прочее);
  - использование фреймворков и написание кода игры с их помощью (React, Vue, Angular и т.п.);
  - использование анимаций и звуковых эффектов для улучшения интерактивности и вовлеченности;
  - использование внешнего сервера для сохранения прогресса игры;
  - получение правильных ответов теста с внешнего сервера (для обеспечения безопасности и исключения варианта подсматривания правильных ответов из кода игры).

  Кастельно масштабируемости приложения возможно рассмотреть следующее:
  - создание сервиса для генерирования подобных квестов пользователем;
  - обеспечение интеграции с социальными сетями и иными сторонними ресурсами (быстрый вход в игру, показ результатов и достижений в игре на других ресурсах и прочее).

Таким образом, качественное техническое задание является основой успешной разработки игрового приложения, обеспечивая понимание всех аспектов проекта и помогая команде разработчиков эффективно реализовать поставленные задачи.

2.1.2. Описание целевой аудитории

Определение целевой аудитории для веб-приложения - это важный шаг в процессе разработки, поскольку это позволяет адаптировать приложение под потребности и предпочтения конкретной группы пользователей. Вот несколько ключевых моментов, которые следует учитывать при описании целевой аудитории:

- Демография:
  - Возрастная группа: какие возрастные категории будут использовать приложение;
  - География: из каких географических регионов будут пользователи;
  - Языковые предпочтения: языковые особенности и предпочтения целевой аудитории.
- Технические параметры:

- Устройства: на каких устройствах (компьютер, смартфон, планшет) предполагается использование приложения;
- Операционные системы: на каких платформах (iOS, Android, Windows) будет работать приложение.

- Потребности и проблемы:
  - Проблемы, которые решает ваше приложение: какие проблемы или потребности приложение должно решать для пользователей;
  - Особенности, которые привлекут аудиторию: ключевые особенности и возможности, которые могут привлечь целевую аудиторию.

Анализ и описание целевой аудитории позволит создать более целенаправленное и эффективное веб-приложение, которое будет лучше соответствовать потребностям и ожиданиям пользователей.

Учитывая вышеуказанное, в случае нашего приложения был сделан следующий выбор:

- Демография:

  Целевая аудитория - дети в возрасте 7-12 лет. При анализе были учтены следующие аспекты:
  - уровень понимания и когнитивные способности детей соответствующего возраста (навыки чтения, знания сказок и т.д.);
  - предпочтения детей в области игр и развлечений;
  - особенности восприятия информации и интерфейсного дизайнау детей. География и язык - Россия, русский язык, весь мир для русскоговорящей аудитории.
- Технические параметры:

  Приложение должно успешно работать на всех современных устройствах и операционных системах.
- Потребности и проблемы:

  Решение задачи доступности игрового обучения детей, вовлечение их в процесс обучения, получения новых знаний, развития навыков запоминания и прочих. Особенности приложения, которые могут привлечь целевую аудиторию: интерфейс, анимация, яркие картинки, тематика игры, мотивация на успех.

2.1.3. Разработка архитектуры приложения

Архитектура приложения определяет его структуру и способ организации компонентов для обеспечения эффективной работы и удобства сопровождения. В процессе разработки приложения были использованы следующие принципы:

- Применение концепции единой страницы (SPA) для обеспечения плавного пользовательского опыта и минимизации времени загрузки.
  Single Page Application (SPA) – это тип веб-приложения, которое загружает все необходимые ресурсы (HTML, CSS, JavaScript) один раз при начальной загрузке страницы, а затем динамически обновляет контент без перезагрузки страницы при взаимодействии пользователя с приложением. SPA отличается от традиционных многостраничных приложений тем, что он работает в рамках одного HTML-документа, что создает плавный и мгновенный пользовательский опыт.
  Плюсы Single Page Application:
  - Быстрая навигация: Поскольку все необходимые ресурсы загружаются один раз, пользователи могут переключаться между разделами приложения без задержек и перезагрузок страниц;
  - Отзывчивость и скорость: SPA позволяют создавать более динамичные и интерактивные пользовательские интерфейсы за счет использования AJAX для обновления данных на странице без перезагрузки;
  - Улучшенный пользовательский опыт: SPA обеспечивают гладкую и интуитивную навигацию, что приводит к повышению удовлетворенности пользователей и улучшению общего впечатления от приложения;
  - Уменьшение нагрузки на сервер: SPA позволяют минимизировать количество запросов к серверу за счет загрузки данных частями и их кэширования, что позволяет сократить нагрузку на сервер и сеть;
  - Удобство разработки и обслуживания: Разработка SPA позволяет легко организовать код в виде компонентов, что облегчает разделение ответственности и обслуживание приложения;
  - SEO-оптимизация: Современные SPA-фреймворки (например, React, Angular, Vue) предоставляют инструменты для рендеринга на стороне сервера, что позволяет улучшить индексацию и оптимизацию для поисковых систем;
  - Мобильная совместимость: SPA легко адаптируются для работы на мобильных устройствах, что делает приложение более удобным и доступным для пользователей с различными устройствами;

  Single Page Application – это эффективный и современный подход к разработке веб-приложений, который обеспечивает быстродействие, высокую отзывчивость и улучшенный пользовательский опыт.

- Модульное построение приложения для упрощения разработки и добавления новых функций в будущем.

  Модульное построение приложения — это подход к разработке программного обеспечения, при котором код приложения разделяется на независимые модули или компоненты, каждый из которых выполняет определенную функцию или задачу. Модульность способствует улучшению структуры приложения, упрощает сопровождение, поддержку и масштабируемость проекта.

  Плюсы модульного построения приложения:

  - Чистый и структурированный код: Разделение приложения на модули способствует созданию более понятной и организованной структуры кода;
  - Повторное использование кода: Модули могут быть повторно использованы в различных частях приложения или в других проектах, что сокращает время разработки и уменьшает риск ошибок;
  - Изоляция функционала: Каждый модуль отвечает за определенный функционал, что упрощает обслуживание, тестирование и дебаггинг кода;
  - Параллельная разработка: Разные части приложения могут быть разработаны независимо друг от друга, что позволяет команде работать параллельно над различными модулями;
  - Легкая подмена модулей: При необходимости один модуль может быть легко заменен другим без влияния на остальные части приложения;
  - Улучшенная масштабируемость: Модульное построение упрощает добавление нового функционала или расширение приложения за счет гибкой архитектуры;
  - Удобство поддержки и обновлений: Изменения в одном модуле могут быть внесены без необходимости перекомпиляции всего приложения, что упрощает обновление и поддержку кода;
  - Более понятная команда разработки: Каждый разработчик может работать над своим модулем, что способствует повышению продуктивности и улучшению совместной работы в команде.

    Модульное построение приложения является эффективным подходом к разработке программного обеспечения, способствующим повышению гибкости, удобства обслуживания, отказоустойчивости и общего качества разрабатываемого продукта.

- Составление блок-схемы, отражающей взаимосвязь между компонентами.

Блок-схема приложения - это графическое представление логики работы приложения с использованием стандартных символов и блоков, чтобы показать последовательность

операций, принятие решений, поток данных и другие аспекты функционирования программы. Блок-схема помогает визуализировать структуру приложения, его алгоритмы и процессы.

Назначение блок-схемы приложения:

- Визуализация алгоритмов: Блок-схема позволяет ясно представить алгоритмы и последовательность действий, не углубляясь в технические детали программирования;
- Анализ и оптимизация: С помощью блок-схемы можно быстро выявить слабые места, улучшить процессы и алгоритмы работы приложения;
- Документация процессов: Блок-схема служит отличным инструментом для документирования логики и функционала приложения, что упрощает понимание для разработчиков и специалистов;
- Коммуникация и обучение: Блок-схема помогает облегчить общение в команде разработки, обучение новых сотрудников и понимание работы приложения сторонними лицами;
- Ошибки и дебаггинг: При анализе блок-схемы можно обнаружить ошибки в логике работы приложения, что облегчает дебаггинг и исправление проблем;
- Проектирование системы: Перед началом разработки приложения, блок-схема помогает спланировать структуру, логику и взаимосвязь компонентов системы.

Блок-схема приложения помогает структурировать процессы и алгоритмы работы, что упрощает анализ, оптимизацию и визуализацию логики приложения, делая его функционирование более понятным и эффективным.

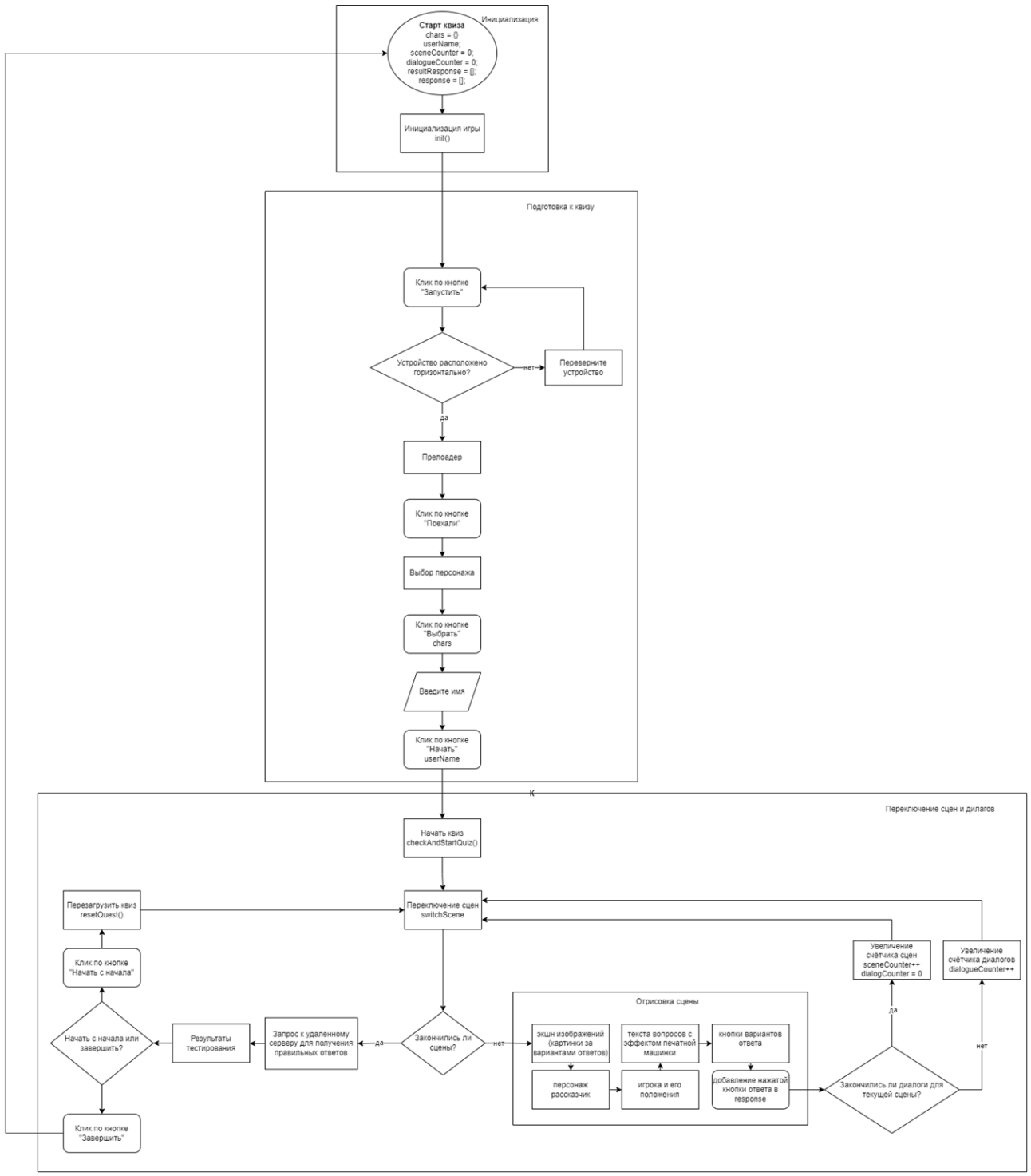

Рисунок 1. Блок-схема приложения.

Элементы блок-схемы носят справочный характер, основной целью является передать логику работы приложения.

Работа приложения состоит из 3 основный этапов:

Этап 1. Инициализация.

На этом этапе происходит создание переменных, а именно:

- chars - содержит выбранный персонаж.
- userName - имя игрока.
- sceneCounter - отвечает за смену сцен по ходу квиза.
- dialogueCounter - отвечает за смену диалогов в текущей сцене.
- resultResponse - массив с правильными ответами.
- response - массив с ответами игрока.

Так же запускается функция инициализации игры - init().

Этап 2. Подготовка к квизу.

- Рендеринг приветственного экрана с кнопкой “Запустить”, после клика на которую происходит проверка на положение устройства. Если устройство расположено не горизонтально, то происходит рендеринг страницы с анимацией предложения “Перевернуть устройство”.
- Если устройство расположено горизонтально, запускается “Прелоадер” для загрузки изображений.
- После окончания загрузки изображения предлагается нажать на кнопку “Поехали” для запуска ряда действий для инициализации игрока, а именно:
  - Выбор персонажа подразумевающий клик по кнопке “Выбрать” с интересующим персонажем и запись выбора в переменную chars.
  - Ввод имени игрока с запись имени в переменную userName.

Этап 3. Переключение сцен и диалогов.

- Запуск функции checkAndStartQuiz() с проверки корректности введенных данных на этапе подготовки к квизу и запуск функций switchScene().
- switchScene() проверяет закончилась ли сцена и если нет, то:
  - Если нет, то:
    - Происходит подэтап отрисовка сцены:
      - Экшн изображения (картинка за вариантами ответов).
      - Персонаж рассказчика (бабушка, кот учёный).
      - Игрок и его положение.
      - Текст вопрос с эффектом печатной машинки.
      - Кнопки вариантов ответов
      - Добавление индекса нажатой кнопки ответа в массив response
    - Проверка закончилась ли диалоги для текущей сцены:
      - Если нет, то происходит увеличение счетчика диалогов dialogueCounter на единицу и вновь возвращается в функцию switchScene().

- Если да, то происходит обнуление счетчика диалогов dialogueCounter, а также увеличения счётчика сцен sceneCounter на единицу и вновь возвращается в функцию switchScene().

○ Если да, то:

■ Происходит запрос к удаленному серверу для получения правильных ответов.

■ Отрисовка экрана с результатами тестирования.

■ Диалоговое окно с возможностью начать с начала или завершить:

- Если произошёл клик по кнопке “Начать с начала”, то через функцию resetQuest() происходит перезагрузка квиза и запускается функция switchScene().
- Если произошёл клик по кнопке “Завершить”, то происходит возвращение в “Старт квиза”.

2.1.4. Проектирование пользовательского интерфейса и взаимодействия с ним

Этот этап включал разработку пользовательского интерфейса (UI) с учетом особенностей целевой аудитории и лучших практик в области дизайна для детей. Основные аспекты проектирования включали в себя:

- Создание ярких и привлекательных элементов интерфейса, привлекающих внимание детей;
- Обеспечение простоты и интуитивной понятности интерфейса для максимального удобства использования;
- Адаптивный дизайн, позволяющий приложению корректно отображаться на различных устройствах и экранах.

Проектирование пользовательского интерфейса (UI) и взаимодействия (UX) для приложения интерактивного игрового теста для детей требует учета ряда особенностей, связанных с возрастными особенностями аудитории и их способностью взаимодействовать с приложением. Вот несколько ключевых шагов, которые могут помочь в этом процессе:

1. Исследование аудитории:
   - Проведение исследований целевой аудитории, в данном случае - детей. Определите возрастные группы и их способности к восприятию информации и взаимодействию с приложением.
2. Определение целей:

- Определение цели приложения. Например, это может быть обучение определенным предметам, развитие навыков или просто развлечение.

3. Создание пользовательских сценариев:
   - Разработка пользовательского сценария, описывающего, как дети будут взаимодействовать с приложением. Учет их потребностей, интересов и уровня взаимодействия.
4. Разработка интерфейса:
   - Создание яркого, привлекательного и интуитивно понятного интерфейса. Использование ярких цветов и простых форм для элементов интерфейса.
   - Обеспечение больших кнопок элементов управления, чтобы дети могли легко нажимать на них.
   - Возможное использование анимаций и звуковых эффектов для улучшения интерактивности и вовлеченности.

5. Удобство использования:

   - Обеспечение простоты и удобства использования приложения.
   - Обеспечение возможности прерывания и возобновления игры в любой момент.

6. Тестирование с целевой аудиторией:

   - Проведение тестирования прототипа приложения с детьми из целевой аудитории. Получение обратной связи для улучшения пользовательского опыта.

7. Обеспечение безопасности и конфиденциальности:

   - Обеспечение защиты данных пользователей, особенно если приложение предполагает сбор какой-либо личной информации о детях.
   - Рассмотрение возможности добавления родительского контроля и ограничений.

8. Обновление и поддержка:

   - После выпуска приложения продолжение сбора обратной связи от пользователей и внесение улучшения в интерфейс, а также взаимодействие на основе их запросов и комментариев.

Проектирование UI/UX для детского приложения требует тщательного подхода и постоянного внимания к потребностям и возможностям вашей целевой аудитории.

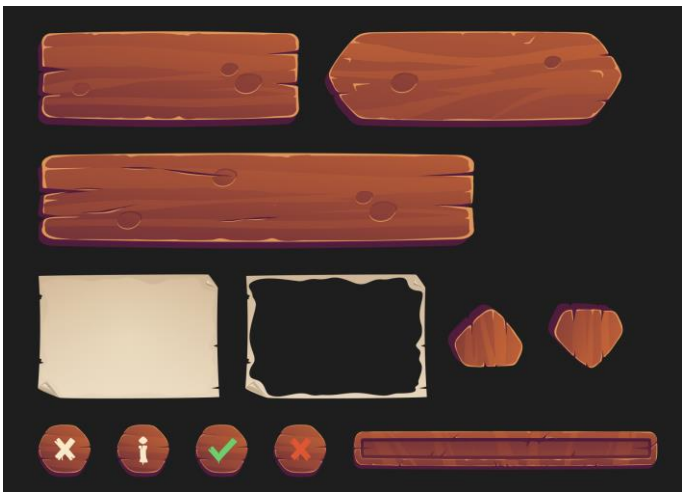

Рисунок 2. UI-kit web-приложения

2.1.5. Описание технологического стека

Технологический стек определяет набор технологий и инструментов, которые будут использоваться при разработке приложения. В данном случае был выбран следующий технологический стек:

- HTML, CSS и JavaScript для разработки клиентской части приложения, так как они являются стандартными технологиями для веб-разработки и хорошо подходят для создания интерактивных интерфейсов.
- HTML (HyperText Markup Language - язык гипертекстовой разметки) используется для создания структуры и содержимого веб-страницы. Он представляет собой набор элементов или тегов, которые определяют тип и содержание элемента на странице. HTML дает возможность определить заголовки, параграфы, списки, изображения, формы и другие компоненты страницы. Вместе с CSS, HTML обеспечивает отображение и структурирование элементов на странице.
- CSS (Cascading Style Sheets - каскадные таблицы стилей) используется для оформления и внешнего вида веб-страницы, создания разнообразных эффектов, адаптивного дизайна и анимации. Он определяет, как HTML-элементы должны быть отображены на странице, задает шрифты, цвета, отступы, размеры, позиции, фоны и многое другое. С помощью CSS можно создавать красивые и удобные пользовательские интерфейсы.
- JavaScript (JS) является языком программирования, который добавляет динамичность и интерактивность на веб-страницы. JS позволяет выполнять различные действия на сайте, например, изменять содержимое страницы без перезагрузки, реагировать на

пользовательские действия (щелчки мыши, нажатия клавиш и т.д.), обрабатывать данные форм, создавать сложные взаимодействия с сервером и другие возможности. Он также может использоваться для создания анимаций и игр.

Комбинирование HTML, CSS и JS позволяет создавать полноценные веб-приложения и сайты. HTML определяет структуру и содержимое, CSS обеспечивает внешний вид и представление элементов, а JS добавляет интерактивность и функциональность. Эти технологии тесно взаимодействуют друг с другом, что позволяет создавать богатый пользовательский опыт и эффективно управлять веб-страницей или приложением.

Вместе эти технологии обеспечивают разработку веб-страниц с привлекательным внешним видом, кросс-браузерной совместимостью и богатыми возможностями интерактивности. Они являются основой фронтенд-разработки и неотъемлемой частью создания веб-приложений и сайтов.

2.1.6. Выбор инструментов и сервисов

Выбор инструментов и сервисов для разработки зависит от многих факторов, включая командную работу, удобство использования, производительность и поддержку. В нашем случае, выбор состава инструментов включает в себя следующие аспекты:

- Visual Studio Code с расширением Live Share:
  - Visual Studio Code (VSC): Это мощный и легкий в использовании редактор кода, разработанный Microsoft. Он предлагает широкий выбор расширений для различных языков программирования и технологий, что делает его универсальным инструментом для разработчиков;
  - Live Share: Это расширение для VSC, позволяющее разработчикам совместно работать над кодом, обмениваться комментариями, отладчиком и терминалом, не покидая редактор. Это особенно полезно для удаленной работы или совместной разработки.
- GitHub:
  - Хостинг Git-репозиториев: GitHub предоставляет удобный способ хранить и управлять версиями кода с помощью Git;
  - Совместная работа: GitHub облегчает совместную работу над проектами, позволяя создавать ветки, делать запросы на объединение изменений и проводить код-ревью;

    - Интеграции: GitHub интегрируется с множеством сервисов и инструментов разработки, что упрощает процесс разработки и развертывания приложений.
- Midjourney:
    - Это исследовательский проект и программа искусственного интеллекта, которая создаёт изображения по текстовому описанию;
    - Чтобы создать изображение, пользователь отправляет команду боту в мессенджере Discord. Вводит сообщение /imagine и после приглашения prompt описывает желаемое изображение;
    - После этого пользователю предлагают выбрать одно из четырёх сгенерированных изображений и получить его в высоком графическом разрешении.
- Photoshop:
    - Многофункциональные инструменты редактирования: Photoshop предоставляет широкий спектр инструментов для создания элементов UI и редактирования изображений, включая ретушь, коррекцию цвета, текст и работу со слоями;
    - Слои и маски для гибкого редактирования: Принцип работы со слоями и масками позволяет пользователям изменять изображения без разрушения оригинальных данных, обеспечивая гибкость и контроль над процессом;
    - Генерация и обработка изображений: Photoshop предоставляет возможности для создания новых изображений и обработки существующих с использованием мощных инструментов и фильтров. Это включает функции, такие как создание эффектов, рисование, растровую и векторную графику, а также работу с 3D-изображениями.
- Google Chrome для тестирования и дебаггинга:
    - Инструменты разработчика: Chrome предоставляет набор инструментов разработчика, таких как инспектор элементов, консоль и отладчик JavaScript, что помогает в тестировании и отладке веб-приложений;
    - Профилирование производительности: С помощью Chrome можно профилировать производительность веб-приложений, идентифицировать узкие места и улучшить их производительность.
- Google Drive:
    - Хранение файлов: Google Drive предоставляет бесплатное пространство для хранения файлов в облаке. Пользователи могут загружать файлы различных форматов, таких как документы, изображения, видео и многое другое;
    - Синхронизация файлов: Google Drive позволяет синхронизировать файлы между различными устройствами, такими как компьютеры, смартфоны и планшеты. Это обеспечивает доступ к вашим файлам из любой точки с доступом в Интернет;

- Общий доступ к файлам: предоставляет возможность обмена файлами и папками с другими пользователями, раздавая им различные уровни доступа, такие как просмотр, комментирование и редактирование.

Эти инструменты и сервисы взаимодействуют друг с другом, обеспечивая удобное и эффективное рабочее окружение для разработки, тестирования и управления проектами.

При выборе инструментов и сервисов для разработки, тестирования, дебаггинга, хранения файлов было принято во внимание несколько ключевых факторов:

- Удобство использования: Все выбранные инструменты предоставляют интуитивно понятный пользовательский интерфейс и широкие возможности для эффективной работы разработчиков;
- Функциональность: Visual Studio Code обеспечивает мощный набор инструментов для написания кода, а Live Share позволяет совместно работать над проектами, что повышает производительность команды. GitHub предоставляет удобное хранилище для кода и средства для совместной работы. Midjourney позволяет генерировать оригинальные изображения, а Photoshop позволяет редактировать эти изображения. Google Chrome предоставляет мощные инструменты для тестирования и дебаггинга веб-приложений;
- Интеграция: Все выбранные инструменты легко интегрируются друг с другом, что обеспечивает плавный и эффективный рабочий процесс;
- Поддержка и сообщество: Visual Studio Code, GitHub и Google Chrome имеют активное сообщество пользователей и обширную документацию, что делает процесс изучения и использования этих инструментов более удобным;
- Безопасность: GitHub обеспечивает безопасное хранение кода и управление доступом к репозиториям, а Google Chrome предоставляет инструменты для обнаружения и устранения уязвимостей в веб-приложениях.

В целом, выбор этих инструментов обеспечивает удобное и эффективное рабочее окружение для разработки, тестирования и управления проектами, что способствует повышению производительности и качества разработки.

2.2. Практика

2.2.1. Структура приложения

На этом этапе была определена структура приложения, включая организацию файлов и папок, а также взаимосвязи между различными компонентами. Это позволяет легко навигировать по проекту и поддерживать его в актуальном состоянии.

Порядок и наименования папок и файлов приложения:

- Папка components:
  - boozy.ttf - файл шрифта;
  - imagePreloader.js - файл формата JavaScript, в котором хранятся объекты содержащие расположение файлов изображений, и методы, которые выполняют функцию предварительной загрузки изображений;
  - sceneConfigurations.js - файл формата JavaScript, который содержит массив объектов сценария и диалогов приложения.
- Папка images:
  - Содержит все файлы изображений квеста в форматах png и svg.
- Корневая папка проекта:
  - answers.json - файл формата json, содержащий массив правильных ответов приложения;
  - index.html - файл формата HTML, содержащий верстку страницы квеста, а так строки подключения файлов форматов js и scc, для компиляции и стилизации страниц приложения;
  - main.js - файл формата JavaScript, содержит все основные методы и функции для работы приложения;
  - style.css - файл формата CSS, содержит все стили и форматы шрифтов приложения.

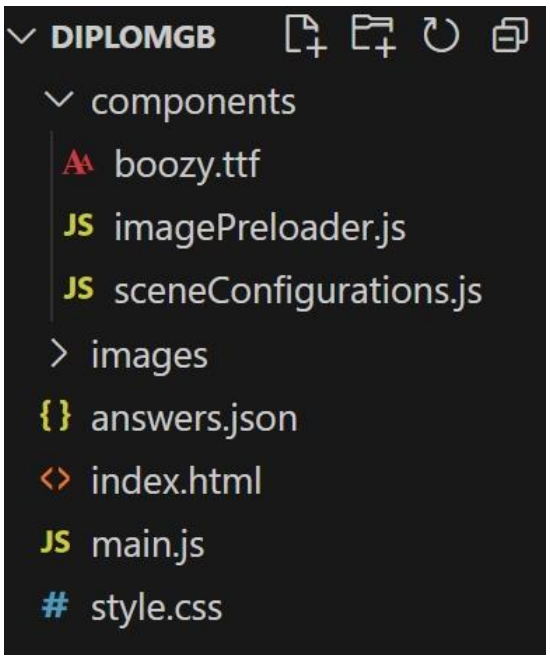

Рисунок 3. Структура проекта

2.2.2. Разработка приложения: Этапы и Описание функций

Разработка приложения была разбита на этапы, каждый из которых включал в себя определенный набор функций и задач. Это включало в себя отображение информации о текущем состоянии игры, обработку действий пользователя, реализацию функционала тестирования и обработку ответов пользователя.

Этапы разработки приложения:

Этап 1. Инициация, первый код:

- проработка первого варианта дорожной карты проекта;
- определение основных требований к конечному продукту (квиз, игровая форма обучения/проверки знаний у детей) и его формата/структуры (сценки с фоновым изображением и блоками типа "вопрос-ответ", "диалог-вопрос-ответ", и пр);
- определение темы квиза с целью поиска и генераций фоновых изображений и иных картинок (знание русских народных сказок);
- определение инструментария и среды для дальнейшей работы (VSC(Liveshare), Github, Midjourney, Google Docs, Google Chrome, Photoshop);
- определение формата приложения - SPA. Это позволит "не разрывать связь" между модулями игры (например, стартовой страницей и непосредственно игрой), чтобы

загружать следующий модуль без перезагрузки всей страницы. Таким образом, смена модулей будет производиться динамически, и игроки будут оставаться "в игре", наблюдая загрузку следующего модуля (а не смотреть на пустое окно загружающегося браузера).

Первые наброски кода для понимания логики:

HTML

Создан макет сцены квиза для понимания его формата.
В Body добавлены несколько разделов (div): раздел для фона сцены, раздел для стационарных кнопок ("справка" и "выйти"), раздел для рубрики квиза (блок с диалогом и вопросом и блоки вариантов ответов).
Присвоение элементам классов осуществлялось по методологии БЭМ;

CSS

Основная стилистика для страницы (ширина и высота страницы и фонового изображения, обнуление стандартных стилей, примерные стили кнопок и разделов, и прочее);

JS

Кроме основного файла для скрипта (main.js) сразу создаем отдельный файл sceneConfigurations.js для хранения массива со сценами квиза.
В файле sceneConfigurations создаем для экспорта константу sceneConfigurations, в которой определяем массив объектов, имеющих следующие свойства: background, buttons и dialogue. Данную константу импортируем в файл main.js, где прописываем основную линию квиза:

1. создаем константы для основных разделов HTML документа (всем div-ам присвоены классы, соответственно, их можно найти с помощью Document метода querySelector()). Поскольку многие div-элементы будут использоваться неоднократно, - упрощаем работу браузера в части необходимости повторного поиска того или иного элемента. Указанное позволит сэкономить ресурсы и, тем самым, улучшить оптимизацию работы приложения;
2. создаем переменную-счетчик sceneCounter для прохода по всем объектам массива sceneConfigurations, и присваиваем ей значение первого элемента массива - "0";
3. создаем функцию switchScene(), которая, используя конструкцию if/else, проверяет прошел ли игрок все сцены квиза и:

   если "нет", то формирует сцену квиза. Фону и вопросу (диалогу) сцены присваиваются значения, получаемые из того объекта массива sceneConfigurations, индекс которого соответствует значению sceneCounter. Варианты ответов (предварительно обнуленные командой .innerHTML = ‘’), чтобы не дублироваться на следующей сцене) создаются путём добавления новых div-элементов - кнопок с вариантами ответа. Принимая во внимание, что кнопок может быть несколько, их создание осуществляется проходом через

массив buttons в соответствующем объекте массива sceneConfigurations => используем функцию map().

Далее добавляем слушатель событий на каждую созданную кнопку с помощью метода addEventListener() для того, чтобы поймать событие клика на кнопку. Слушатель событий оборачиваем в отдельную call-back функцию и присваиваем ей имя, чтобы потом мы могли удалить слушатель и не перегружать браузер. По итогам нажатия на любую из кнопок-ответов в квизе должна смениться сцена, - соответственно, sceneCounter должен увеличиться на единицу (1).

если "да", удаляем слушатель событий и заканчиваем работу функции.

Этап 2. Наполнение и плавность

Наполнение игры (картинки, вопросы и варианты ответов).

Добавлен код: скелет стартовой страницы игры, плавный переход от страницы к странице, добавили анимацию при наведении на кнопки.

HTML

Добавляем разметку раздел (div) для стартовой страницы.

CSS

Центрируем текст стартовой страницы (position: absolute; top: 50%; left: 50%; transform: translate(-50%, -50%); text-align: center);

Чтобы при загрузке приложения единственной видимой частью была стартовая страница, - остальные разделы html не отображаем (display: none).

Добавляем код для обеспечения плавности перехода со страницы на страницу (свойства transition, opacity) - совместно с js кодом, который описан ниже.

Добавляем эффект наведения на кнопки с вариантами ответов (transition transform - scale).

JS

Написан код для обеспечения плавности перехода со страницы на страницу (совместно с CSS): добавляем eventListener на кнопки "начать тест" и "закрыть", по итогам нажатия на которые меняется видимость (opacity) страниц (исчезает текущая, далее делается активной и появляется новая страница (class ‘active’ и opacity; используется функция setTimeout).

К eventListener кнопок с вариантами ответов на вопросы квиза добавляем функцию для подсчёта выбранных ответов. Функция, используя метод push, добавляет индекс выбранной кнопки в массив response. Далее данный массив будем сравнивать с массивом правильных ответов.

Этап 3. Первые усложнения кода

Дорабатываем диалоги и усложняем массив sceneConfigurations ими (конфигурация из двух сцен, в каждой сцене свой фон, рассказчик, диалоги и вопросы квиза). К диалогам добавили свои кнопки-ответы, которые не учитываются как кнопки-ответы на вопросы квиза (разделили функционал).

Решен вопрос с использованием userName (переменная, которая будет получать имя игрока по input) в диалогах => массив sceneConfigurations теперь представляет собой переменную, содержащую в себе анонимную стрелочную функцию с атрибутом userName, которая использует этот атрибут в своем массиве с объектами сцен квиза.

CSS

Доработали основную стилистику квиза:

Создан корневой элемент root и в нем различные переменные для использования по всему коду.

Все размеры переведены из px в rem.

Создана переменная высоты диалога (dialogueHeight), которая представляет собой сложенные высоты всех элементов сцены.

Проработаны экшн изображения, в т.ч. "маски" к картинкам (см. все стили к .quest__action-img:after и .quest__action-img img).

Кнопкам настроен внешний вид - используем свойства border (border-width: 1rem; border-style: solid; border-color: transparent; border-image-source: url(./images/btn.svg); border-image-slice: 20 fill;).

JS

Переработка функции switchScene с учетом усложнения массива sceneConfigurations (теперь свойство dialogue каждого объекта массива само собой представляет массив из различных диалогов и кнопок к ним):

- добавили счетчик для сцены и счетчик для отдельного диалога сцены;
- отдельно отображается сцена, ее фон и прочее, отдельно отображаются диалоги:
- диалог без вопроса квиза не имеет экшн картинку (ActionImage), его кнопки не участвуют в подсчете правильных ответов квиза (сделали конструкцию if/else в части наличия или отсутствия в диалоге кнопок квиза - hastestbuttons)
- диалог с вопросом квиза отображает свою экшн картинку, кнопки вариантов ответа располагаются под вопросом (создается div actionImage, в нем картинка actionImageItem, где

адрес картинки - это путь к массиву sceneConfigurations и далее к свойству actionImage соответствующего его объекта).

В функции switchScene обращение к массиву sceneConfigurations изменено на обращение к sceneConfigurations(userName).

Этап 4. Переходы к началу квиза и выход

Созданы предварительные страницы квеста: экран выбора персонажа, приветствия, и переходы к ним и от них. Теперь стартовая страница при нажатии кнопки "начать" ведет на экран приветствия/выбора персонажа, а далее уже есть кнопка "начать квиз". Также проработан момент, чтобы кнопка "начать квиз" срабатывала только если пользователь введет свое имя в input имени.

Проработан функционал значка "закрыть".

HTML

В разметку раздела quest в начало добавлен раздел quest__character, состоящий из трех quest__character-part: приветствие, выбор персонажа и вводом имени.

Создана разметка предварительных страниц к квизу (страница-приветствие, страница-выбора персонажа, страница ввода имени игрока).

CSS

Основные стили к разделам.

Стилизация в рамках работы значка "закрыть" - при появлении окна-предупреждения о том, что будет сброшен прогресс по игре, фон игры (кроме окна-предупреждения) затемняется и размывается: создается псевдоэлемент .quest.active:before со следующими стилями:

```
{
   content: '';
   z-index: -1;
   background: #00000080;
   transition: 0.3s opacity ease;
   backdrop-filter: blur(5px);
}
```

Opacity по умолчанию - 0, а при нажатии значка "закрыть" элементу .quest.active:before присваивается opacity: 1.

Также всем элементам раздела (.quest.active.confirm:before, .quest__confirm, .quest.active:before, .quest__background, .quest__controls) присваиваются соответствующие z-index, чтобы они появлялись друг над другом в нужном порядке.

JS

Созданы новые константы по добавленной в HTML разметке (startQuiz, questCharacterParts, questFirstPartBtn, chooseCharacterBtns, questCharacterInput).

Функционал значка "закрыть" проработан с помощью функции closeQuest.addEventListener("click"...), которая после клика на значок:

- создает в разметке раздел (div) pop-up окно с предупреждением о сбросе прогресса и кнопками "да" и "нет";
- присваивает класс "confirm" окну игры (questWindow) для целей его стилизации - затенение и размытие;
- создает слушатели событий для кнопок "да" и "нет" соответственно. По нажатию кнопки "да" удаляем окно-предупреждение (удаляем из разметки раздел с классом quest__confirm), возвращаем стили (удаляем класс "confirm" у окна игры, тем самым убираем затенение и размытие), обнуляем счетчики sceneCounter и dialogueCounter, обнуляем массив ответов квиза, запускаем заново функцию switchScene (сбрасываем прогресс игры). По нажатию кнопки "нет" удаляем окно-предупреждение и возвращаем стили (удаляем класс "confirm" у окна игры, тем самым убираем затенение и размытие). То же самое - при нажатии на любое место, кроме окна-предупреждения (код как при нажатии кнопки "нет").

Кнопка startQuiz переводит на страницу выбора персонажа, вызывает функцию handleCharactersParts(), которая, в свою очередь, вызывает функцию handleCharactersParts(), - функции, отвечающие за видимость текущей части разметки и невидимость всех остальных.

Кнопка questFirstPartBtn переводит в следующие разделы предварительных страниц теста, убирая видимость других - в теле слушателя события клика по этой кнопке есть функция activateNextPart().

Кнопка startQuiz теперь переводит в сам квиз, при этом проверяет, ввел ли пользователь свое имя. Если ввел, - то переменной userName присваивается значение, введенное игроком.

Добавлена функция init(), которая отвечает за инициацию квиза и его "переходных" кнопок, включая кнопки "закрыть" - выход из квиза (слушатели событий на кнопки startBtn, questFirstPartBtn, startQuiz и closeQuest - описанные выше).

Этап 5. Рассказчик и персонаж квиза

На кнопки выбора персонажа добавили слушатель событий, который по клику на тот или иной персонаж записывает соответствующий выбор игрока в параметр функции sceneConfigurations.

В соответствующие диалоги квиза (те, в которых нет вопросов квиза) добавлены таблички с именем того или иного рассказчика (Бабушка и Кот) .

HTML

Раздел с классом div class="quest__text-dialogue" обернули в раздел div class="quest__text" для того, чтобы в данном родительском разделе появлялся div - табличка с рассказчиком.

CSS

Удалили возможность игроку выделять тексты в игре свойством user-select: none.

К стилям добавляем вендорные префиксы для работы во всех устаревших браузерах (см. caniuse.com).

JS

В функцию switchScene() добавлен код для появления в сценах рассказчика. Рассказчик появляется в сценах без вопросов теста.

На кнопки выбора персонажа с помощью цикла forEach добавлен слушатель событий для записи выбора персонажа в параметр функции sceneConfigurations(). Количество параметров данной функции теперь два: userName и charArr.

Этап 6. Код для итоговой страницы квиза. Дополнения.

Добавили страницу с результатами, подсчет результатов, кнопки "завершить" и "начать заново" (разметка добавляется с помощью JS).

CSS

Добавили стили финальной страницы квиза.

Общий стиль для всех текстов quest.

JS

Дописали финальный else в функции switchScene(), который работает как только все сцены квиза пройдены (sceneCounter >= sceneConfigurations(userName, charArr).length), в котором:

- осуществляется подсчет количества правильных ответов (сравнение массива полученных от игрока ответов с массивом корректных ответов методом reduce(), при совпадении ответа игрока с правильным ответом увеличивается счетчик count;
- убирается видимость у диалогов квиза с кнопками;
- добавляется фоновое изображение итоговой страницы;
- добавляется разметка с разделом quest__result, состоящим из quest__result-title (подсчитывает процент правильных ответов и, в зависимости от того, больше или меньше он 50%, выводится соответствующих текст - молодец или попробуй еще), quest__result-finals (показывает статистику - количество правильных ответов) и quest__result-btns (кнопки "начать заново" и "завершить");
- на раздел .quest__result-btns прикрепляется слушатель событий, который слушает клики по данному разделу (метод делегирования событий) и, в случае клика на кнопку с id #result-btn__startover, переводит игрока в начало квиза (в первую сценку квиза, после ввода имени и после выбора персонажа); а в случае клика на кнопку с id #result-btn__end, переводит на страницу загрузки квиза (закрывает приложение). Слушатель также удаляет разметку с разделом quest__result (разметку финальной страницы квиза) и возвращает разметку сцен квиза (questTextContainer.style.display = "").

Действия по закрытию квиза перенесли в две функции, т.к. они используются также в функционале значка "закрыть":

- function resetQuest() - обнуление прогресса квиза (sceneCounter, dialogueCounter и массива response) и вызов заново функции switchScene(). Данная функция отдельно нужна для кнопки "начать заново" в финальной странице квиза;
- function completeQuest() - плавно меняет стили появляющихся и исчезающих страниц, а также запускает обнуление прогресса теста (вызывает функцию resetQuest()).

Этап 7. Пишущая машинка

Проработан эффект пишущей машинки для всех текстов диалогов. Также кнопки диалогов появляются после полного появления текста соответствующего диалога.

CSS

Всем текстам, а также кнопкам диалогов, задано свойство по умолчанию opacity: 0.

JS

В функцию switchScene() добавлен код для плавного появления и исчезновения кнопок диалогов (меняем opacity).

Создали отдельную функцию printText(), которая вызывается функцией switchScene(). Функция делит каждый текст на символы, далее с помощью метода map() для каждого символа создает свой div-элемент span и включает его в раздел верстки textContainer. И далее с помощью функции setTimeout каждому span-элементу присваивается свойство opacity: 1.

Функция addBtns() также вынесена за пределы switchScene(). В функцию добавлен функционал появления (opacity: 1). Одним из аргументов функции printText() является callback - функция, которая вызывается как только все span-элементы получат свою opacity: 1 (завершится цикл работы со всеми символами текста диалога). При вызове printText() в качестве callbackпередается функция addBtns(), - таким образом, кнопки появятся только после появления текстом диалога.

Этап 8. Правки кода

Помимо рефакторинга кода также проработана функция fetch() для получения массива правильных ответов с внешнего сервера.

Проработан эффект плавности появления и исчезновения кнопок диалогов.

JS

Правки кода:

- функция switchScene() теперь не вызывается в функции resetQuest() - удалили вызов, т.к. при reset-е она отрабатывала и далее вызывалась повторно в основном коде - тем самым квиз начинался со второй страницы;
- создана переменная printTimeout - ее имя получила функция setTimeout() в функции printText(), чтобы далее обращаться к этой части функции по имени и удалять ее - также при reset игры (в функции resetQuest()) обнуляется значение printTimeout. Иначе тексты диалогов продолжали бы печататься при перезагрузке игры;
- создана функция fetchAnswers(), куда в качестве параметра(аргумента) callback перенесен весь последний else из switchScene(). В Функции fetchAnswers() данный callback запускается только после того, как приложение загрузит массив верных ответов (массив верных ответов теперь хранится вне игры, и мы его получаем из внешнего сервера).

Этап 9. Прокрутка кнопок

Работа над добавлением эффекта прокрутки к кнопкам-ответам на диалоги/вопросы квиза, если они не умещаются на экране.

HTML

В разметку в раздел кнопок добавили дополнительный контейнер (теперь общий раздел - nav-container, далее контейнер для кнопок игры - quest__btns, и контейнер для значков "вверх" и "вниз" - quest__arrows) для позиционирования значков "вверх" и "вниз" прокрутки кнопок диалогов в случае, если они не помещаются на экране пользователя. Теперь к значкам"вверх" и "вниз" добавлено позиционирование относительно этого нового контейнера.

CSS

Добавлен массив значков "вверх" и "вниз" - arrows.

Контейнер (nav) - position: relative. Значки "вверх" и "вниз" - position: absolute, плюс для каждого из них нулевое значение top и bottom, соответственно. У контейнера удален свой вшитый scroll-bar.

Кнопкам диалогов добавлены паддинги, чтобы при их увеличении (scale 1.1) в результате наведения на них (hover), их края не выходили за пределы контейнера и, соответственно. не пропадали.

JS

Дорабатываем функцию handleArrowBnts() - меняем opacity у значком "вверх" и "вниз" в зависимости от того, где в scrollbar "находится" пользователь (конструкция if/else проверяет положение questBtnsContainer.scrollTop). Данная функция вызывается слушателем событий "scroll" на questBtnsContainer-е.

На каждый значок "вверх" и "вниз" циклом forEach добавили eventListener ‘click’, который по клику прокручивает до следующей кнопки (вверх и вниз, соответственно).

Этап 10. Рефакторинг. Добавление кнопки "справка"

Проработали функционал значка (i) (кнопка "справка") - практически скопирован код значка "закрыть", но со своей кнопкой и со своим текстом.

Добавили код работы с изображениями рассказчика и персонажа игры. Отредактировали повторяющийся код в отдельные функции, основываясь на принципе DRY (Don’t repeat yourself).

HTML

В разметке переструктурирован раздел для кнопок "справка" и "закрыть": им добавлен общий раздел (div class="quest__controls"), в котором есть отдельный раздел для кнопки "справка" (quest__controls-info) и кнопки "закрыть" (quest__controls-close).

CSS

Стили для значка "справка" практически аналогичны стилям значка "закрыть".

JS

В функцию init() добавлен слушатель событий на кнопку (i) - код практически аналогичен коду кнопки "закрыть".

По некоторым повторениям кода проведен рефакторинг: появились новые функции: handlePopUp(), handleConfirmButtons(), checkAndStartQuiz().

Добавлен функционал использования изображений рассказчика и персонажа:

Свойство storyteller в sceneConfigurations теперь представляет собой объект, содержащий свойства name и image.

Также и char свойство, отвечающее за персонаж, будет объектом, содержащим рисунок.

У обоих storyteller char и также появятся свойство position, отвечающее за расположение рисунков в сцене игры.

Функция switchScene() дополнена дополнением рисунка рассказчика (часть функции с блоком if/else (currentDialogue.storyteller) и рисунка персонажа (блок if/else (currentDialogue.char).

В функцию resetQuest() добавлен код с удалением этих рисунков в случае обнуления квиза.

Этап 11. Позиционирование рассказчика и персонажа.

CSS

Добавлено позиционирование рассказчика и персонажа. которое будет работать по умолчанию. Высота рассказчика и персонажа рассчитывается по формуле относительно 100dvw, таким образом, чтобы при уменьшении страницы просмотра, обе картинки в сцене меняли размеры соответственно. Аналогичен и расчет их позиционирования относительно нижней части экрана (также по формуле). Z-index: -1, для того, чтобы картинки не загораживали диалоги сцен.

JS

В некоторых sceneConfigurations.dialogues добавлено свойство position персонажу и/или рассказчику. Альтернативное позиционирование предполагается по оси X, соответственно,

свойства position определяют, с левой или с правой стороны экрана будет персонаж/рассказчик (left: true/false), и далее на какой % от левой/правой границы экрана делать отступ (свойство offset).

Если в sceneConfigurations.dialogues не будет свойства position, то рисунки будут расставлены по умолчанию согласно CSS.

В main добавлена функция handlePersonPosition(). Данная функция вызывается в функции switchScene() в случаях, если в текущем диалоге игры есть свойство рассказчика (currentDialogue.storyteller) или персонажа (currentDialogue.char). В параметрах данной функции вышеуказанное свойство, а также класс его рисунка (".quest__storyteller-image" или ".quest__char-image"). Функция проверяет, есть ли в свойствах рассказчика/персонажа свойство position, если есть - заменяет стили CSS свойствами, указанными в position, а если нет - обнуляет эти свойства так, чтобы работал по умолчанию CSS.

Этап 12. Предзагрузка изображений.

JS

Все изображения, используемые в игре, включены в объект images (экспортируемый в файл main.js). Для удобства, изображения разложены по отдельным объектам, в зависимости от того, к какому предмету они относятся (персонаж, фон, рассказчик и т.п.). В итоге имеем пары ключ-значение (значением выступают пути к файлу), разложенные по своим тематикам.

Выясняем общее количество изображений для предзагрузки (формируем из images массив imagesAsArray и получаем его длину: totalLength = imagesAsArray.length).

Далее проходим циклом по всем объектам images и если у вложенного объекта есть пара ключ-значение, - создаем new Image() и загружаем его методом onload().

Дополнительно следим за прогрессом загрузки (количество загруженных изображений относительно totalLength) - функция updateProgress(), которая получает значение прогресса загрузки и вставляет его в свойство ширины раздела разметки, отвечающего за показ прогресса (".quest__preloader-progress").

Для разметки создана функция preloader(), которая сначала формирует разметку для показа прогресса загрузки игры (div class="quest__preloader" и его дочерние элементы), вызывает функцию preloadImages() и далее, как только изображения будут загружены, плавно уменьшает видимость раздела "quest__preloader". Аргументом preloader() является callback функция, которая срабатывает в самом конце для целей переключения на следующую страницу игры.

Этап 13. Работа над эффектами

JS

Добавлены функции fadeIn() и fadeout(), которые обеспечивают плавность перехода со страницы на страницу квиза, плавность появления/исчезновения изображений рассказчика и персонажа, экшн изображений и пр.

Создана функция createOverlay(), которая для переходов на страницы с меняющимся фоном создает затемнение (div class="quest__overlay") и тем самым, обеспечивает плавную смену изображений.

Этап 14. Адаптивная версия игры

HTML

В разметку добавлен раздел quest__rotate, который по умолчанию невидим (display: none) и становится видимым только если пользователь заходит в игру на телефоне и держит его вертикально (orientation: portrait). В рамках данного раздела также создан раздел quest__rotate-text с текстом "поверни устройство", и раздел quest__rotate-img, который с помощью CSS отражается как макет телефонного устройства.

CSS

Доработан адаптив для @media screen and (max-width: 960px).

Создана функция rotate, которая показывает как нужно перевернуть устройство в случае использования телефона вертикально (quest__rotate-img поворачивает горизонтально).

Описание функций

Функция switchScene()

Задача функции: ключевая функция приложения. Отвечает за переход по сценам/диалогам игры, формирование сцен/диалогов (включая отрисовку фона и изображений игры), а также формирует финальную страницу с результатами игры.

Параметры функции: функция без параметров.

Какими функциями вызывается: в функции switchScene() вызывает саму себя, addBtns(), checkAndStartQuiz(), fetchAnswers().

Какие функции вызывает: switchScene(), handleQuizPart(), printText(), handlePersonPosition(), fadeIn(), fadeOut(), resetQuest(), completeQuest().

Описание функции: Функция "проходит" по всем сценам и диалогам непосредственно квиза, а именно: массива sceneConfigurations (каждый проход счетчик dialogueCounter увеличивается

на 1, и далее также sceneCounter, изначально равные 0, с увеличением счетчика sceneCounter обнуляется счетчик диалогов), формируя константы, отвечающие за текущую сцену/диалог (currentScene, currentDialogue) и кнопки (questBtns), и:

- вызывает функцию handleQuizPart() для формирования экшн изображения либо для его исчезновения, в зависимости от того, является ли текущая сцена/диалог страницей с вопросом квиза;
- если текущая сцена предполагает наличия фонового изображения (в массиве sceneConfigurations для этого диалога предусмотрено свойство background), - присваивает разметке ".quest__background-image" путь согласно значению свойства background в массиве sceneConfigurations;
- формирует раздел с вопросом теста или текстом рассказчика в диалоге: изначально удаляет предыдущий текст из данного раздела (на случай, если он каким-либо образом уже был - обнуляется значение questDialogue.textContent), а далее запускается функция printText() и addBtns() с параметрами соответствующей сцены/диалога квеста;
- если текущая сцена/диалог предполагают вывод имени и/или изображения рассказчика (textStorytellerElem, storytellerImageElem), то есть в массиве sceneConfigurations для данного диалога предусмотрены свойства storyteller/storyteller.image, - создает соответствующую разметку и вставляет туда данные согласно имеющимся значениям в свойствах storyteller/storyteller.image массива sceneConfigurations, вызовом функции fadeIn() обеспечивает плавность появления изображения рассказчика на странице, вызовом функции handlePersonPosition() обеспечивает его позиционирование на странице. Если в текущем диалоге изображения и/или имени рассказчика не предусмотрено, - данные элементы удаляются из разметки (включая их плавное затемнение, осуществляемое с помощью функции fadeOut());
- аналогично осуществлено появление/исчезновение персонажа на той или иной странице игры в случае если для текущего диалога в массиве sceneConfigurations предусмотрено свойство char;

Как только функция отработала весь массив sceneConfigurations, осуществляются следующие действия:

- видимость (opacity) персонажа и рассказчика (в случае если они сформированы в последней сцене) снижается до 0 (данные изображения исчезают);
- вызывается функция fetchAnswers() для получения массива правильных вариантов ответов квиза из внешнего сервера, и далее:

- производится подсчет количества правильных ответов игрока (путем сравнения массива его ответов (массива response) с массивом, возвращенным функцией fetchAnswers(), - resultResponse;
- удаляется контент последней страницы квиза (обнуление значения questBtnsContainer.innerHTML, отмена показа раздела разметки questTextContainer;
- осуществляется замена фона и формирование разметки страницы результатов квиза (div class="quest__result"), куда также вставлены данные о результатах квиза. В данной разметке предусмотрены кнопки "Начать сначала" и "Завершить", на которых предусмотрены слушатели события "клик". По клику на кнопку "Начать сначала" вызываются функции resetQuest() для удаления результатов пройденного квиза и запускается снова функция switchScene(). По клику на кнопку "Завершить" вызывается функция completeQuest(). Оба слушателя подразумевают, что по клику на любую из данных кнопок, будет удалена разметка, отвечающая за страницу результатов квиза.

Функция printText()

Задача функции: отображение текста вопросов и диалогов на экране с эффектом "печатной машинки".

Параметры функции: text - принимает текущую сцену диалога из массива sceneConfigurations.

textContainer - HTML-контейнер, куда будет добавляться текст.

callback - аргумент для вызова функции addBtns()**.**

Какими функциями вызывается: функцией switchScene().

Какие функции вызывает: addBtns().

Описание функции: используя метод split() с параметрами "" преобразует строку text в массив, где каждый символ текста отдельно является его элементом. С помощью метода map перебираем массив text, который принимает callback функцию с аргументами char и index. С помощью итерации каждого элемента массива(char) создается переменная span, которая с помощью метода createElement() создает элемент для разметки страницы HTML. Далее элементу span задается свойство textContent со значением char. После чего с помощью метода append в контейнер textContainer добавляется значение элемента span. В созданную ранее глобально переменную printTimeout присваивается значение, содержащее функцию setTimeout, которая через каждый промежуток времени, заданный в значении index * 20, присваивает стилю opacity значение "1" соответствующему элементу span. Данное свойство изначально было задано с параметром "0", изменение данного свойства в сочетании с

временным промежутком, заданным в setTimeout(), отображает буквы на экране в порядке очереди, тем самым позволяя добиться эффекта "печатной машинки". Далее идёт проверка на длину массива букв text.length - 1, которая сравнивается со значением index, и при их равенстве, вызывает callback, в который передана функция addBtns(), что позволяет отобразить кнопки ответов или диалогов сразу после отображения последнего элемента(буквы) массива text.

Функция addBtns()

Задача функции: формирует кнопки-ответы игрока для выбора ответа на вопрос квиза, для перехода на следующий диалог, а также для получения выбранных игроком ответов на вопросы квиза.

Параметры функции: currentDialogue - текущий диалог сцены в массиве sceneConfigurations.

Какими функциями вызывается: в функции switchScene() функцией printText(). Является callback функции printText() и срабатывает после полной отработки данного метода.

Какие функции вызывает: switchScene(), createOverlay(), fadeOut(), handleArrowBtns().

Описание функции: используя метод map(), функция проходит по всем элементам btns конкретного диалога в массиве sceneConfigurations, для каждого элемента создает в разметке свой элемент (div), присваивает ему класс (quest__btns-item) и контент (соответствует значениям свойства btns). На каждую созданную таким образом кнопку вешается слушатель события "клик", по которому вызывается функция switchScene() для переключения на следующий диалог/сцену игры (вызов функции switchScene() обернут в конструкцию if/else для случая смены сцены игры, где меняется фон, и необходимо его сменить плавно с помощью функции createOverlay()), а также, если текущая кнопка является кнопкой-ответом на вопрос квиза, - индекс кнопки методом push() вносится в массив ответов игрока (response).

Кнопка вставляется (метод append()) в соответствующий раздел разметки (questBtnsContainer) и с небольшой задержкой (10 миллисекунд с помощью setTimeout()) кнопке присваивается "opacity: 1" для воспроизведения плавной анимации появления (по умолчанию в CSS кнопкам присвоен "opacity: 0").

Функция также вызывает handleArrowBtns(), созданную в целях показа стрелок в случае недостаточного для кнопок места на странице.

Функция handleQuizPart()

Задача функции: отображение экшн изображения для страниц с вопросами квеста.

Параметры функции: currentDialogue - текущий диалог массива sceneConfigurations.

Какими функциями вызывается: switchScene().

Какие функции вызывает: fadeIn(), fadeOut().

Описание функции: используя конструкцию if/else текущий диалог проверяется на факт, относится ли диалог к вопросу квиза (есть ли в текущем диалоге sceneConfigurations свойство isTestBtns, которому присвоено значение true). И, если да, то:

- окну квиза присваивается атрибут istest, равный true (в css наличие данного атрибута предполагает иную стилистику и расположение элементов (кнопок-ответов на вопрос квеста) на странице;
- в разметку добавляется раздел для экшн изображения (quest__action-img) и функцией fadeIn()вызывается его плавное появление.

В случае else (проверка не выявила, что данная страница относится к странице вопроса квиза), - атрибуту istest окна квиза возвращается значение false, раздел quest__action-img удаляется из разметки и функцией fadeOut() обеспечивается плавное исчезновение экшн изображения.

Функции handleCharacterPage() и handleCharactersParts()

Задача функций: отображение предварительных к квизу страниц (приветствие, выбор персонажа, указание имени игрока).

Параметры функций: параметром функции handleCharacterPage() является булевое значение (принимает true или false). У функции handleCharactersParts() параметры отсутствуют.

Какими функциями вызываются: switchScene().

Какие функции вызывают: checkAndStartQuiz(), init().

Описание функций: обеспечивают корректное появление раздела верстки ".quest__character" как в случае загрузки игры, так и в случае ее приостановки/перезагрузки: присваивает класс "active" данному разделу верстки, если в параметр функции принят аргумент true, и удаляет данный класс в случае аргумента false. Если данный раздел верстки отображен после перезагрузки игры, то handleCharacterPage() удалит видимость у каких-либо кнопок и диалогов игры. Аналогично, если к моменту загрузки игры в памяти браузера остался выбор игроком персонажа игры, handleCharactersParts() обнуляет данный выбор (удаляет класс active у всех разделов, связанных с персонажем игры).

Функция activateNextPart()

Задача функции: обеспечение перехода к началу квеста.

Параметры функции: функция не имеет параметров.

Какими функциями вызывается: init(), слушателем событий на кнопках ".quest__character-begin" и на кнопках выбора персонажа ".quest__character-item" (при клике на любую из них).

Какие функции вызывает: -.

Описание функции: в случае если у текущей страницы игры есть класс "active", - функция его удалит и присвоит этот класс следующей странице (класс "active" в css дает видимость элементу).

Функция completeQuest()

Задача функции: обеспечивает окончание квеста по нажатию соответствующих кнопок игроком ("завершить" - при окончании квеста и/или "подтвердить завершение" - при принудительном выходе из квеста до его окончания).

Параметры функции: функция не имеет параметров.

Какими функциями вызывается: switchScene(), init() .

Какие функции вызывает: resetQuest().

Описание функции: убирает текущему окну игры видимость (opacity = 0), используя паузу с помощью функции setTimeout() удаляет текущему окну класс "active". далее вызывает функцию resetQuest() для продолжения закрытия игры.

Функция resetQuest()

Задача функции: обеспечивает полное и корректное завершение игры.

Параметры функции: функция не имеет параметров.

Какими функциями вызывается: switchScene(), resetQuest().

Какие функции вызывает: -.

Описание функции: обнуляет все счетчики (sceneCounter, dialogueCounter, массив response), удаляет все имеющиеся на текущей странице элементы, если таковые есть (рассказчик, персонаж), останавливает выполнение функции printText(), т.к. она может продолжать "печатать" диалоги квиза.

Функция fetchAnswers(callback)

Задача функции: получение JSON файла, который содержит массив правильных ответов на тест.

Параметры функции: callback.

Какими функциями вызывается: switchScene().

Какие функции вызывает: вызывают фунцию, которая передана в callback().

Описание функции: при вызове асинхронной функции fetchAnswers её параметром является функция callback.На первом этапе происходит проверка по методам try/catch.В блоке try задается переменная answer, который метод fetch с оператором await, аргументом данного

метода является строка с именем файла.В результате чего, в переменную answer, мы получаем promise. Далее идёт проверка if, на получение promis от сервера, в случае если отрицательного результата, объявляем ошибку с текстом "Не удалось получить ответы".Затем в объявленную ранее переменную resultResponse, значением которой является пустой массив.В функции fetchAnswers(), мы переопределяем значение переменной resultResponse, на значение answer.json() с оператором await.После чего идет вызов функции переданной в callback().В блоке catch происходит обработка ошибок, и отображение их в console.

Функция handleArrowBtns()

Задача функции: необходима для показа кнопок квеста в случае недостаточности им места на странице.

Параметры функции: функция не имеет параметров.

Какими функциями вызывается: addBtns(), init() слушателями событий скролла, изменения размера окна браузера.

Какие функции вызывает: -.

Описание функции: в случае если высота контейнера кнопок в браузере игрока будет меньше высоты контейнера кнопок игры, сверху и/или снизу контейнера кнопок появляются соответствующие стрелочки "вверх"/"вниз" для скролла по контейнеру. Функция создает видимость для данных стрелочек в различных сценариях действий игрока (если есть кнопки внизу и/или наверху, т.е. есть метро для скролла).

Функция handlePopUp()

Задача функции: обеспечивает удаление разделов верстки ".quest__confirm" и ".quest__info" - окон, всплывающим при нажатии игроком кнопок "закрыть" и "справка" соответственно.

Параметры функции: container (раздел верстки, который нужно удалить), button (кнопка, на которую нажал игрок для появления popup-окна), className (класс, который нужно удалить у окна игры), target (таргет, на клик в который среагирует слушатель события "клик").

Какими функциями вызывается: init() слушателями событий на кнопках "справка" и "подтвердить" у кнопки "закрыть".

Какие функции вызывает: -.

Описание функции: в случае если игрок нажал кнопку "справка" и/или "закрыть" на странице игры всплывают соответствующие окна. Функция handlePopUp() удалит эти окна из верстки, если игрок нажмет в любое место страниц, кроме самого popup-окна и/или еще раз кнопок "справка"/"закрыть" (функция проверяет корректность нажатия именно в пустое место окна браузера), а также удалит класс, активирующий работу кнопок "справка" и "подтвердить" (в

CSS наличие такого класса обеспечивало затемнение видимости основного окна игры при появлении popup окон, и прочие эффекты).

Функция handleConfirmButtons()

Задача функции: присвоение класса confirm кнопкам подтверждения пользователя.

Параметры функции: container, className.

Какими функциями вызывается: init().

Какие функции вызывает: -.

Описание функции: функция начинает свою работу с проверки на наличие в questWindow переданного аргументом класса. В случае, если такой класс есть, происходит создание переменной confirmContainer, которой присваивается значение из questWindow, заданное при помощи метода querySelector с переданным ему в качестве аргумента наименования контейнера(container). Далее в теле if происходит повторная проверка на наличие confirmContainer: в случае, если условие выполняется, происходит удаление из questWindow элемента с заданным className. Затем происходит удаление контейнера confirmContainer.

Функция checkAndStartQuiz()

Задача функции: запрос от пользователя введения ника и начало квеста.

Параметры функции: функция без параметров.

Какими функциями вызывается: -.

Какие функции вызывает: createOverlay(), fadeOut(), handleCharacterPage(), switchScene().

Описание функции: функция начинает свою работу с проверки заполнения строки ника пользователя. В случае, если данное условие не выполняется, questCharacterInput присваивается класс error. На экране происходит подсветка красным цветом и квест не начинается. В случае, если пользователь ввел ник, происходит запуск функции createOverlay(). Далее, для плавности перехода к первой сцене квеста, с помощью функции setTimeout() происходит вызов функции fadeOut(), которая принимает в качестве аргумента элемент с классом quest__overlay. Затем происходит проверка на элемент с классом error и, в случае его наличия, этот класс удаляется у данного элемента. Происходит вызов функции handleCharacterPage с аргументом false. Переменной userName присваивается значение ника, введенное пользователем, и происходит вызов switchScene(). Происходит начало первой сцены квеста.

Функция handlePersonPosition()

Задача функции: отображение персонажей квеста в различных позициях.

Параметры функции: person, personClass.

Какими функциями вызывается: switchScene().

Какие функции вызывает: -

Описание функции: при вызове функции создается переменная personImg, которой присваивается значение элемента с классом personClass. Далее происходит проверка на наличие свойства position у переменной person, в случае, если условие выполняется, - переменной personImg присваиваются свойства left и right значением "". Затем происходит проверка на значение person.position.left === true, в случае выполнения условия свойству left присваивается значение person.position.offset + "%", в ином случае, свойству right присваивается значение person.position.offset + "%". В случае невыполнения условия наличия person.position, свойствам left и right присваиваются значения "".

Функция init()

Задача функции: функция-инициация игры. В ней собраны все основные слушатели событий.

Параметры функции: функция без параметров.

Какими функциями вызывается: вызов функции осуществляется автоматически программой (при открытии игры).

Какие функции вызывает: fadeIn(), completeQuest(), handleCharachterPage(), activateNextPart(), checkAndStartQuiz(), handleConfirmButtons(), handlePopUp(), handleArrowBtns().

Описание функции: Функция прикрепляет следующие слушатели событий click/enter/scroll/resize:

1. На кнопку "Запустить" (".start-quest__btn"). По клику на эту кнопку формируется следующее окно игры (ему присваивается класс "active" для работы CSS по отражению окна на странице, а также, с помощью функции fadeIn() обеспечивается плавное появление окна), в том числе, создается фон (присваивается атрибут src с адресом фото фона). Далее, вызовом функции handleCharachterPage() на странице формируется раздел для выбора персонажа игры и отстраиваются разделы с вариантами персонажей (разделам ".quest__character-part" присваивается класс "active").
2. На кнопку "Поехали" (".quest__character-begin"), по клику на которую вызывается функция activateNextPart().
3. На обе кнопки выбора персонажа (".quest__character-item"). По клику на любую из них также вызывается функция activateNextPart(), а также в переменную, отвечающую за путь к изображению персонажа (char) присваивается соответствующий путь в зависимости от выбранного игроком персонажа.

4. На кнопку "Начать" (".quest__character-start"), по клику на которую вызывается функция checkAndStartQuiz().

5. На поле ввода имени игрока (".quest__character-input") при нажатии клавиши "enter" вызывается функция checkAndStartQuiz().

6. На кнопку "закрыть" (".quest__controls-close") совершаются следующие действия:

- после проверки на отсутствие popup-окна-подтверждения, такое окно (с классом "quest__confirm") создается в разметке, в том числе со своими кнопками-значками "подтвердить" и "закрыть";
- после проверки на отсутствие класса "confirm" у окна страницы, такой класс ей присваивается (наличие такого класса отображает визуальный эффект затемнения основного окна со всеми его элементами при появлении popup);
- на созданные в popup-окне кнопки-значки "подтвердить" и "отменить" прикрепляются слушатели события "клик", т.о. при клике производится вызов функций handleConfirmButtons(), а в случае "подтвердить", также еще и completeQuest();
- слушатель события на любое место, помимо popup-окна, который по клику вызовет функцию handlePopUp(), - отработает закрытие popup-окна и возврата к игре.

7. На кнопку-значок "справка" (".quest__controls-info") аналогичный п.6 выше слушатель, который также обеспечивает появление всплывающего окна справки, затемнение основной страницы игры, и по кнопке-значку "закрыть справку", а также клику в любую точку, кропе popup-окна, закрывает popup-окно и возвращает к игре.

8. На кнопки-стрелочки ("arrows"), появляющиеся при помощи функции handleArrowBtns(), при клике на которые обеспечивается прокрутка кнопок-ответов на вопросы квиза вверх/вниз соответственно.

9. На контейнер с кнопками-ответами на вопросы квеста (".quest__btns") прикреплен слушатель скролла, который вызывает функцию handleArrowBtns().

10. На все окно браузера в случае изменения его размера (resize) также слушатель среагирует вызовом функции handleArrowBtns().

Функция createOverlay()

Задача функции: затемнить страницу игры для ее плавного исчезновения.

Параметры функции: функция без параметров.

Какими функциями вызывается: checkAndStartQuiz(), addBtns().

Какие функции вызывает: fadeIn().

Описание функции: в разметку окна игры, у которого нет класса ".quest__overlay", с помощью функции fadeIn(), добавляется раздел (div) с классом ".quest__overlay".

Функция preloader()

Задача функции: показ окна предзагрузки игры.

Параметры функции: параметром функции выступает callback, как совокупность действий, осуществляемых после полной отработки preloader() (описание данных действий дано выше в описании функции init()).

Какими функциями вызывается: init().

Какие функции вызывает: fadeOut(), preloadImages().

Описание функции: создает страницу показа предзагрузки - включает в разметку раздел с классом "quest__preloader", далее запускает функцию preloadImages() и, по окончании ее работы, с помощью функции fadeOut() плавно удаляет созданную разметку.

Функция preloadImages()

Задача функции: предзагрузка всех изображений игры.

Параметры функции: параметрами функции являются:

- images - все изображения. используемые в игре;
- callback, - совокупность действий, осуществляемых после полной отработки preloadImages() (описание данных действий дано выше в описании функции preloader()).

Какими функциями вызывается: preloader().

Какие функции вызывает: fadeOut(), updateProgress().

Описание функции: создает две переменные-счетчики, отвечающие за количество загруженных изображений (loadedImages) и общее количество изображений в константе images (totalImages). Циклами for-in проходит по всем вложенным объектам images с ключами, имеющими значения - пути к изображениям (src). При нахождении каждого такого объекта увеличивает счетчик totalImages на 1, далее создает объект new Image(), присваивает ему путь(src) и методом onload запускает его загрузку. По окончании загрузки данного new Image(), увеличивает счетчик loadedImages и запускает функцию updateProgress() с аргументами - двумя счетчиками. В случае когда количество загруженных изображений будет равно общему количеству изображений для загрузки игры (loadedImages === totalImages), функция вызывает callback.

Функция updateProgress()

Задача функции: показ прогресса предзагрузки игры.

Параметры функции: счетчики loadedImages и loadedImages, созданные функцией preloadImages().

Какими функциями вызывается: preloadImages().

Какие функции вызывает: -.

Описание функции: находит раздел разметки с классом ".quest__preloader-progress" и меняет ему ширину в зависимости от соотношения количества загруженных изображений к общему количеству изображений для загрузки.

2.2.3. Тестирование и отладка

В процессе разработки проводилось тестирование и отладка кода приложения для обеспечения его качества и работоспособности. В процессе тестирования проверялась корректность работы всех функций приложения, а также его совместимость с различными устройствами и браузерами. Ошибки и неполадки были выявлены и устранены, чтобы обеспечить плавный и безошибочный пользовательский опыт.

3. Заключительная часть

В ходе работы были решены следующие задачи:

- Разработка технического задания: Были определены функциональные и нефункциональные требования к игровому тесту, учитывая особенности образовательных целей проекта и потребности целевой аудитории.
- Определение целевой аудитории и ее особенностей: Проведен анализ особенностей целевой аудитории, что позволило эффективно адаптировать контент приложения под нужды пользователей различных возрастных групп.
- Проектирование системы: Были выбраны технологические решения и инструменты, определен архитектурный дизайн и пользовательский интерфейс, обеспечивающие удобство использования и высокую функциональность приложения.
- Реализация ключевых функций приложения: Осуществлена разработка и внедрение основных функциональностей приложения, включая генерацию тестов, обработку ответов пользователей и отображение результатов тестирования.
- Анализ достигнутых результатов: Проведен анализ эффективности приложения на основе обратной связи от пользователей и оценки его влияния на образовательный процесс детей. Полученные результаты позволили выявить сильные и слабые стороны проекта и предложить пути их улучшения.

Решение этих задач позволило успешно реализовать проект и достичь поставленных целей, обеспечив эффективное использование приложения в практике обучения детей на тему сказок.

Было разработано и реализовано игровое приложение "Квиз Сказок", предназначенное для тестирования детей на знание сказочных сюжетов и персонажей. В связи с вызванными современными требованиями к образованию существует необходимость в создании инновационных методов обучения, способных стимулировать интерес у детей к обучению и развивать их критическое мышление.

Целью данного проекта было разработать инновационное приложение, которое помогло бы не только оценивать знания учеников, но и активно вовлекало бы их в обучающий процесс. В результате успешной реализации проекта были достигнуты следующие цели:

- Создание инновационного образовательного приложения: Приложение "Квиз Сказок" представляет собой инновационный подход к обучению детей, сочетая в себе элементы игры и образования для достижения максимальной эффективности обучения.
- Повышение мотивации и интереса учащихся к обучению: Приложение стимулирует учащихся учиться через интерактивные задания и игровые механики, что способствует более эффективному усвоению материала и повышению обучаемости.
- Внедрение интерактивных методов обучения в школьную практику: Проект открывает новые перспективы в образовательном процессе, демонстрируя преимущества использования современных технологий для достижения образовательных целей.

Анализ результатов использования приложения показал его эффективность в обучении детей на тему сказок. Он не только помогает детям углубить свои знания в этой области, но и развивает их умение анализировать и делать выводы. Дальнейшее совершенствование приложения и его адаптация к различным возрастным группам представляются перспективными направлениями для будущих исследований и разработок в области образования детей.

Работа над данным проектом позволила приобрести ценный опыт в области веб-разработки, создания и проектирования пользовательских интерфейсов и анализа потребительских потребностей. Результаты данной работы могут быть использованы как основа для улучшения качества образования с помощью геймификации образовательного процесса.

Подытоживая вышеизложенное, можно утверждать, что данная дипломная работа является важным шагом в развитии сферы обучающих приложений.

Приложение

Веб-приложение:

https://aravag.github.io/diplomGB/

Рисунок 4. Стартовый экран

Рисунок 5. Экран загрузки

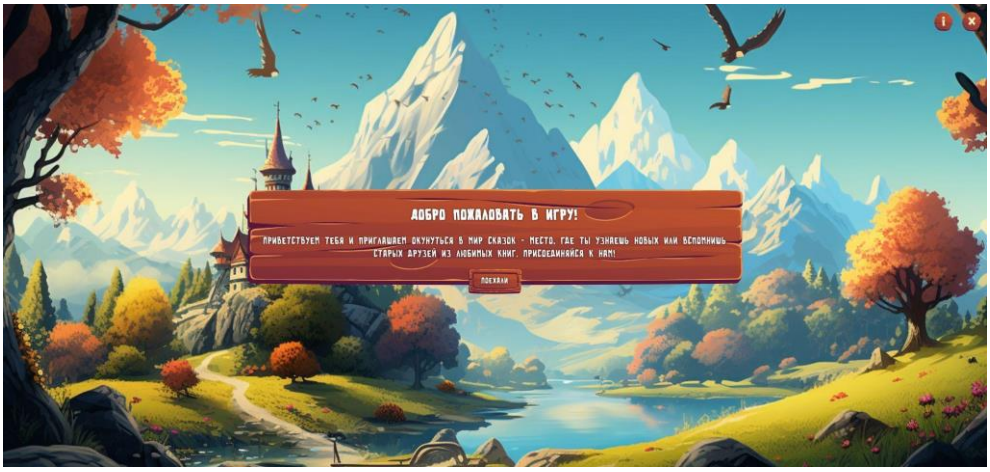

Рисунок 6. Приветственный экран

Рисунок 7. Экран выбора персонажа

Рисунок 8. Экран ввода имени пользователя

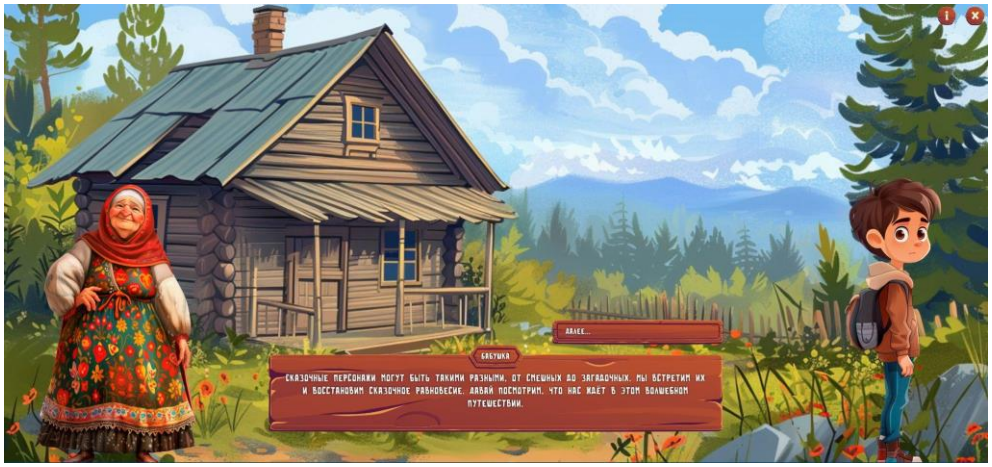

Рисунок 9. Экран сцены

Рисунок 10. Финальный экран с результатами

Рисунок 11. Экран вопроса квиза

Рисунок 12. Экран принудительного закрытия квиза

Рисунок 14. Экран поворота устройства