

**Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования
«Санкт-Петербургский политехнический университет Петра Великого»
(ФГАОУ ВО СПбПУ)
Институт компьютерных наук и кибербезопасности
Высшая школа программной инженерии**

КУРСОВАЯ РАБОТА

по дисциплине «(ЗФО)Верстка и прототипирование сайтов»

Тема: Разработка игрового приложения «Определи вес

Выполнил
студент гр. 35130903/30301

В.А. Смолькин

Руководитель
старший преподаватель

Е.В. Комарова

«14» января 2026 г.

Санкт-Петербург-2026

Содержание

Введение.....	3
Обзор предметной области.....	4
Разработка структуры игры, архитектура приложения, реализация	7
Тестирование	14
Функциональное тестирование игрового процесса:.....	14
Тестирование ввода и валидации:	15
Автоматизированное нефункциональное тестирование (производительность, качество и безопасность).....	16
Заключение	18
Список использованных источников	20
Приложения	21
Приложение А. Исходный код игрового приложения	21
Приложение Б. UML-диаграмма приложения	47
Приложение В. Блок схема игрового процесса	48

Введение

Данная курсовая работа посвящена разработке интерактивного веб-приложения – игры «Определи вес». Цель проекта заключается в создании браузерной игры, в которой пользователь должен угадывать вес различных животных по изображению. В результате выполнения работы должно получиться полнофункциональное приложение с игровым процессом, разделённым на уровни сложности, системой подсказок и подсчётом очков, а также с таблицей рекордов лучших результатов.

При разработке приложения использовались стандартные веб-технологии фронтенда: язык разметки HTML для структуры страниц [1], язык таблиц стилей CSS для оформления внешнего вида [2] и язык программирования JavaScript для реализации интерактивной логики [3]. Архитектура игры спроектирована таким образом, чтобы отделить логику от представления: данные игры (списки животных, уровни сложности и результаты игроков) хранятся и обрабатываются через программные структуры, а интерфейс взаимодействует с пользователем с помощью динамически обновляемого HTML-DOM. Для хранения состояния игры и результатов в процессе работы используется механизм Web Storage API – локальное хранилище браузера localStorage [4].

Ключевыми моментами при проектировании были простота и интуитивность интерфейса, а также обучающий характер игры. В приложении предусмотрены подсказки «больше/меньше», ограниченное число попыток и опциональный таймер, что делает геймплей увлекательным и динамичным. Целевая аудитория игры – широкая: дети школьного возраста, интересующиеся животными, а также взрослые пользователи, желающие проверить свои знания или интуицию относительно веса различных животных. Проектируемый интерфейс и игровой процесс ориентированы на то, чтобы быть понятными и удобными для пользователей любого возраста. В дальнейшем планируется адаптировать дизайн под мобильные устройства, чтобы целевая аудитория могла играть на смартфонах и планшетах.

Обзор предметной области

Игровое приложение «Определи вес» относится к жанру образовательных викторин, в частности к играм на угадывание числовых значений. По своей сути механика игры близка к классической игре «угадай число», где необходимо отгадать задуманное число, получая подсказки «больше» или «меньше». В данном случае загадано не произвольное число, а фактический вес реального животного, изображение которого показывается игроку. Таким образом, игра сочетает элементы обучения (знакомство с весовыми характеристиками различных животных) и развлечения (процесс угадывания с ограничениями по попыткам и времени).

Цель игры: угадать вес животного, изображённого на экране, с допустимой погрешностью. Игрок вводит предполагаемый вес в килограммах, и приложение проверяет ответ. Если погрешность не превышает установленный порог, ответ считается правильным. В разработанной игре в качестве порога выбрано $\pm 10\%$ от реального веса животного – это означает, что угадывание считается успешным, если введённое значение находится в диапазоне $\pm 10\%$ от истинной массы животного. Например, если вес животного 100 кг, то ответы в диапазоне 90–110 кг будут засчитываться как верные. Такой допуск сделан для облегчения игры, учитывая, что точное значение веса обычно сложно угадать.

Логика и правила: после старта игры игрок проходит последовательно несколько уровней. В начале каждого уровня игроку даётся **5 разных животных**, для каждого из которых нужно определить вес. Уровни различаются по сложности:

- **Лёгкий уровень** – начальный уровень сложности. Здесь игроку предоставляется больше всего попыток и наиболее щадящие условия: **5 попыток** на каждое животное, **лимит времени 60 секунд** (если включён режим на время) и множитель сложности $1\times$ для расчёта итогового счёта. Подсказки после неверных ответов помогут скорректировать следующий ввод.
- **Средний уровень** – промежуточная сложность. Количество попыток сокращается до **4**, время на ответ – **45 секунд** (в режиме на время), а множитель итогового счёта равен $2\times$. Это стимулирует более точные и быстрые ответы.
- **Сложный уровень** – максимальная сложность. Игрок имеет лишь **3 попытки** на каждое животное, таймер (при включенном режиме) ограничен **30 секундами**, а к финальному счёту применяется множитель $3\times$. На этом уровне требуется наилучшая интуиция или знания, так как возможности для проб и ошибок минимальны.

Каждый уровень состоит из набора из **5 животных**, случайно выбранных из общей базы без повторений. Таким образом, за прохождение всех трёх уровней пользователь последовательно встретит **15 различных животных** (игра содержит

базу из 20 видов животных, включая как относительно лёгких, так и очень тяжёлых представителей фауны). На **рисунке 1** приведена UML-диаграмма классов, отражающая основные сущности и структуру данных игры (уровни сложности, животные, состояние игры и т.д.).

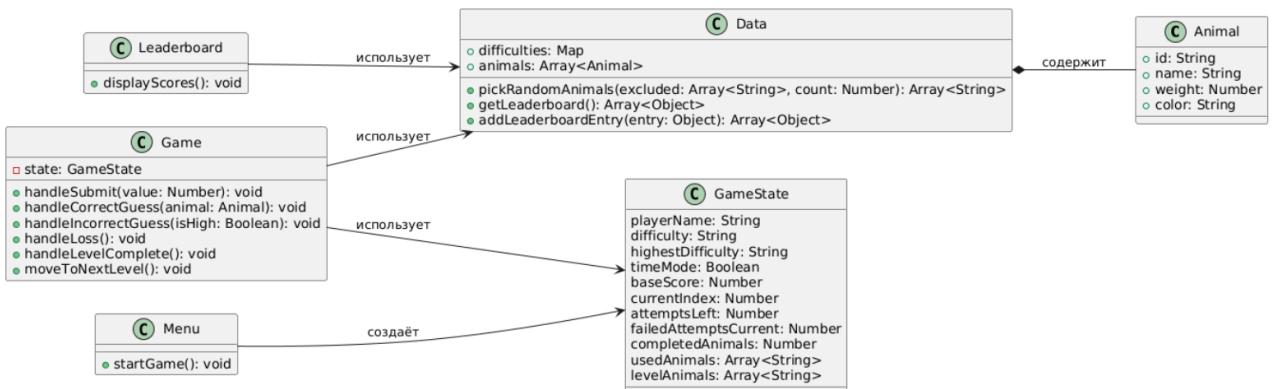


Рисунок 1. UML-диаграмма классов архитектуры приложения.

Игровой процесс внутри уровня построен так: на экране отображается название животного и его изображение (или заглушка, если изображение недоступно). Пользователь вводит свой вариант веса. Если ответ верен (с учётом допуска $\pm 10\%$), игра выводит сообщение о правильном ответе, отображая приблизительный реальный вес животного, и начисляет очки. Затем отображается следующее животное. Если же ответ **не** угадан, игра сообщает, что введённое значение слишком велико или слишком мало, и уменьшает счётчик оставшихся попыток. Игрок может пробовать снова, пока не исчерпан лимит попыток. В режиме с таймером отсчитывается оставшееся время на попытку: если время вышло, эта неудача засчитывается как потеряянная попытка. Когда вес угадан правильно либо закончились попытки, текущий раунд (животное) завершается. Процесс повторяется для всех 5 животных уровня.

Структура уровней: по завершении всех 5 раундов уровня оценивается результат. Если игрок успешно угадал вес каждого животного (неважно, с какой попытки, главное – не выйти за лимит попыток), уровень считается пройденным. Игроку показывается диалог с сообщением об успешном прохождении и с информацией о набранных за уровень очках. При этом вычисляется **итоговый счёт** за уровень с учётом множителя сложности и режима времени. Базовые очки суммируются из всех угаданных весов (за каждое животное можно получить максимум 10 баллов, если угадывание было с первой попытки; за каждую потраченную дополнительную попытку базовые очки уменьшаются). Затем базовая сумма умножается на коэффициент сложности уровня (1, 2 или 3 в зависимости от уровня) и дополнительно удваивается, если был активирован режим с таймером. Такая система стимулирует проходить игру на более сложных режимах для получения более высокого результата.

Если уровень не пройден (т.е. хотя бы для одного животного не удалось угадать вес за отведённые попытки), игра завершается с сообщением об

окончании и предложением начать заново. При успехе же игроку предлагается либо перейти к следующему уровню сложности (если еще остались непройденные уровни), либо сохранить свой результат в таблице рекордов. Третий уровень (сложный) является финальным – после его успешного завершения игра также заканчивается, и полученные очки могут быть сохранены.

Каждый новый уровень начинается с обнулением попыток и времени и с новым набором животных (не повторяющихся с предыдущих уровней текущей игры). Логика изменения состояния игры между уровнями и переходов реализована в коде приложения и будет рассмотрена в следующей главе. В таблице лидеров сохраняются только *лучшие 20* результатов. Пользователь, завершив игру, может ввести своё имя (делается в начале игры) и сохранить итоговый счёт; если он достаточно высок, он появится в рейтинге. В списке рекордов указывается имя игрока, количество очков, достигнутый уровень сложности и отметка о режиме с таймером. Данные хранятся локально в браузере, поэтому при повторном открытии приложения рекорды остаются доступными.

Разработка структуры игры, архитектура приложения, реализация

Общая структура приложения: игра реализована как мультистраничное одностраничное приложение. Это означает, что для различных этапов используются отдельные HTML-страницы, взаимодействующие друг с другом через хранение состояния в localStorage. Основные страницы приложения: меню (главный экран игры), игровой экран и экран таблицы рекордов. Каждая страница имеет свой сценарий (скрипт) на JavaScript, ответственный за ее функциональность. Логика и данные, используемые всеми частями, вынесены в отдельный модуль данных. Ниже перечислены основные компоненты и модули приложения:

- **Модуль данных игры** – содержит константы и структуры, описывающие *предметную область*: перечень животных с указанием их веса, описание уровней сложности (название уровня, количество попыток, время и множитель), а также функции для работы с локальным хранилищем (инициализация и обновление таблицы рекордов) и вспомогательные алгоритмы (например, случайный выбор новых животных). Этот модуль реализован в файле data.js и подключается как ES-модуль на всех страницах.
- **Страница меню** – стартовый экран (menu.html), где пользователь вводит своё имя, выбирает уровень сложности для старта, а также режим игры (с таймером или без). Здесь же отображается описание уровней (карточки с попытками, временем и множителем для каждого уровня) и кнопки начала игры и перехода к рейтингу. Логика этой страницы в скрипте menu.js формирует начальное состояние игры на основе выбора пользователя и сохраняет его в localStorage перед переходом на экран игры.
- **Страница игры** – основной экран (game.html), на котором происходит процесс угадывания. Здесь отображаются текущий игрок и уровень, счёт и прогресс, визуальное представление оставшихся попыток (в виде точек или индикаторов), таймер (если включён), карточка с изображением животного и поле ввода для ответа. Скрипт game.js обрабатывает ввод пользователя: проверяет корректность числа, сравнивает с весом животного и обновляет состояние (количество попыток, очки, вывод подсказки). Также он отвечает за смену карточек животных и анимацию переходов, обработку окончания уровня или игры.
- **Страница таблицы лидеров** – экран leaderboard.html с отображением сохранённых результатов. Здесь в табличной форме перечислены топ-20 результатов: место, имя, очки, уровень, режим таймера. Данный экран только выводит информацию, не предусматривая ввода от пользователя (кроме переходов по меню). Его функциональность реализована скриптом leaderboard.js, который считывает сохранённый массив результатов из localStorage и динамически заполняет таблицу.

Архитектура приложения можно описать с помощью нескольких основных сущностей (см. диаграмму 1 выше). Модуль **Data** (данные) содержит статические данные (список объектов класса **Animal** с их параметрами, структуры уровней сложности) и методы работы с ними. Класс **Menu** осуществляет инициализацию новой игры – собирает ввод пользователя и формирует объект **GameState** (состояние игры). Объект GameState включает все переменные, необходимые для протекания игры: имя игрока, выбранный уровень, максимальный достигнутый уровень, флаг режима времени, счетчики очков, списки использованных и текущих животных и т.д. (поля GameState показаны на диаграмме). Этот объект сохраняется между страницами через localStorage. Класс **Game** реализует сам процесс игры: он загружает сохранённое состояние, на его основе управляет ходом раундов, обработкой попыток и подсказками. Логика функций handleSubmit, handleCorrectGuess, handleIncorrectGuess и др. относится к этому классу. Класс **Leaderboard** отвечает за отображение и хранение итоговых результатов (по сути взаимодействует с Data для получения/сохранения списка рекордов и выводит их).

Отдельно следует отметить применение **Web API** для работы с браузерным хранилищем и DOM. Для записи состояния игры и новых результатов используется localStorage [4]. Например, при старте игры код сохраняет состояние:

```
function createInitialState() {
  const difficultyKey = difficultyOrder[selectedIndex];
  const levelAnimals = pickRandomAnimals([], 5);
  return {
    playerName: nameInput.value.trim(),
    difficulty: difficultyKey,
    timeMode: timeToggle.checked,
    highestDifficulty: difficultyKey,
    baseScore: 0,
    usedAnimals: [...levelAnimals],
    levelAnimals,
    currentIndex: 0,
    attemptsLeft: difficulties[difficultyKey].attempts,
    failedAttemptsCurrent: 0,
    completedAnimals: 0,
  };
}
```

Листинг 1 – Создание начального состояния игры при начале нового сеанса (формирование объекта GameState на основе выбора пользователя).

В листинге 1 показана функция createInitialState() из модуля меню. Она собирает исходные данные: выбранный уровень сложности (difficultyKey), генерирует 5 случайных животных для уровня (функция pickRandomAnimals возвращает массив идентификаторов животных), и формирует объект состояния с начальными значениями. Объект включает имя игрока, флаг режима с таймером, текущий уровень и максимальный достигнутый уровень (изначально

совпадает со стартовым), базовый счёт (0), массив использованных животных (чтобы не повторять их на следующих уровнях), массив текущих животных уровня, индексы и счётчики для управления ходом игры. После создания объекта он сериализуется и сохраняется через localStorage с помощью localStorage.setItem(). Это происходит при нажатии кнопки *Начать игру*, обработчиком которой является функция startGame() в menu.js. Она также проверяет, что имя не пустое (выводит ошибку, если поле имени не заполнено) и переключает страницу на игровую (переход на game.html).

На игровом экране скрипт game.js загружает сохранённое состояние из localStorage и начинает цикл игры. Первым делом отображается первая карточка животного, обновляются индикаторы попыток и таймер. Основной интерактив в игре – обработчик проверки введённого веса. Пользователь вводит число и нажимает кнопку «Проверить вес», что вызывает функцию проверки. Ниже приведён фрагмент кода, реализующий эту логику:

```
function handleSubmit() {
    const value = Number(weightInput.value);
    if (!value || value <= 0) {
        hintText.textContent = "Введите корректное число.";
        return;
    }
    const animal = currentAnimal();
    if (!animal) {
        return;
    }
    clearTimer();
    weightInput.value = "";
    const tolerance = animal.weight * 0.1;
    const isCorrect = Math.abs(animal.weight - value) <= tolerance;
    if (isCorrect) {
        handleCorrectGuess(animal);
    } else {
        handleIncorrectGuess(value > animal.weight);
    }
}
```

Листинг 2 – Проверка введённого пользователем веса (функция обработки ответа в game.js).

В листинге 2 функция handleSubmit() получает числовое значение из поля ввода. Если значение некорректно (пустое, 0 или отрицательное), выводится сообщение об ошибке ввода и функция завершает работу. Затем извлекается текущий объект животного (currentAnimal() возвращает животное, которое нужно угадать на данном шаге). Остановка таймера clearTimer() необходима, чтобы время не продолжало отсчитываться во время обработки. Очищается поле ввода (чтобы пользователь мог вводить следующий ответ с пустого поля). Затем рассчитывается допуск tolerance как 10% от реального веса. Путём сравнения Math.abs(animal.weight - value) <= tolerance определяется, является ли ответ

правильным (переменная `isCorrect`). Если да – вызывается функция `handleCorrectGuess(animal)`, которая осуществляет переход к следующему заданию (или уровню) и начисление очков. Если ответ неверный – вызывается `handleIncorrectGuess(value > animal.weight)`. В эту функцию передаётся булево значение, указывающее, была ли попытка *слишком большой* (перевес) или *слишком маленькой*. Внутри `handleIncorrectGuess` уменьшается счётчик оставшихся попыток и выводится соответствующая подсказка: «Слишком много! Попробуйте меньше.» либо «Слишком мало! Попробуйте больше.». После каждой попытки (и корректной, и некорректной) обновляется отображение интерфейса (очки, прогресс, попытки) и перезапускается таймер для следующей попытки или задания. Когда попытки заканчиваются, вызывается `handleLoss()` – обработчик проигрыша, завершающий игру. Когда все животные уровня угаданы, функция `handleLevelComplete()` подводит итоги уровня (см. сообщение «Уровень пройден!») и предлагает продолжить или сохранить результат.

Архитектура приложения построена таким образом, что **все изменения состояния хранятся в объекте GameState**, который обновляется по ходу игры и сохраняется в `localStorage` после значимых событий (например, после каждой попытки – для сохранения прогресса на случай перезагрузки страницы, и особенно при переходах между уровнями). При завершении игры (выигрыш или проигрыш) временное состояние очищается из `localStorage`. Если игрок выбирает сохранить результат, то с помощью функции `addLeaderboardEntry()` из модуля Data итоговый результат добавляется в таблицу лидеров и также сохраняется. Массив лидеров (`leaderboard`) хранится в `localStorage` постоянно. При первом запуске игры (в скрипте меню) вызывается `initLeaderboard()`, который заносит туда начальный список (демонстрационные данные) при отсутствии существующих результатов.

Стоит отметить некоторые аспекты реализации:

- Для создания интерактивного интерфейса используются функции работы с DOM API [1]. Например, скрипт генерирует элементы списка попыток (точки) через `document.createElement()` и изменяет классы CSS для индикации оставшихся попыток. Также динамически создаются карточки животных (заголовок с названием, контейнер для изображения и т.д.) при смене заданий.
- Приложение использует **CSS Grid** и **Flexbox** для адаптивного расположения элементов интерфейса. В частности, блок с карточками выбора сложности на странице меню реализован как CSS Grid (класс `.menu-grid`), автоматически располагающий карточки уровней; игровая панель статистики использует CSS Grid для выравнивания элементов статистики по сетке; кнопки и формы используют Flexbox (`display: flex`) для красивого расположения и центрирования. Благодаря этим технологиям верстки интерфейс остается удобным на разных размерах экрана [2].

- Добавлены CSS-анимации для улучшения восприятия: при появлении новой карточки животного на игровом экране используется плавная анимация сдвига (класс .enter и .exit, см. стили .animal-card.enter и .animal-card.exit в CSS), создающая эффект перелистывания. Также при наведении на элементы меню или фокусе применяется эффект подсветки и небольшого подъёма (CSS-псевдокласс :hover и свойство transform). Это делает интерфейс более отзывчивым и современным.
- В коде особое внимание уделено обработке **событий с клавиатуры** для повышения удобства: в меню можно навигировать стрелками между уровнями и опциями, нажатием Enter переключать параметры или запускать игру. В игровом экране нажатие цифровой клавиши автоматически фокусирует поле ввода веса, а нажатие Enter дублирует действие кнопки «Проверить вес». Эти возможности улучшения UX реализованы через обработку событий keydown на уровне окна (Window).
- **Блок-схема алгоритма** проверки и переходов в игре представлена на рисунке 2. Она наглядно демонстрирует циклический процесс угадывания веса для каждого животного и ветвление логики при правильных и неправильных ответах.

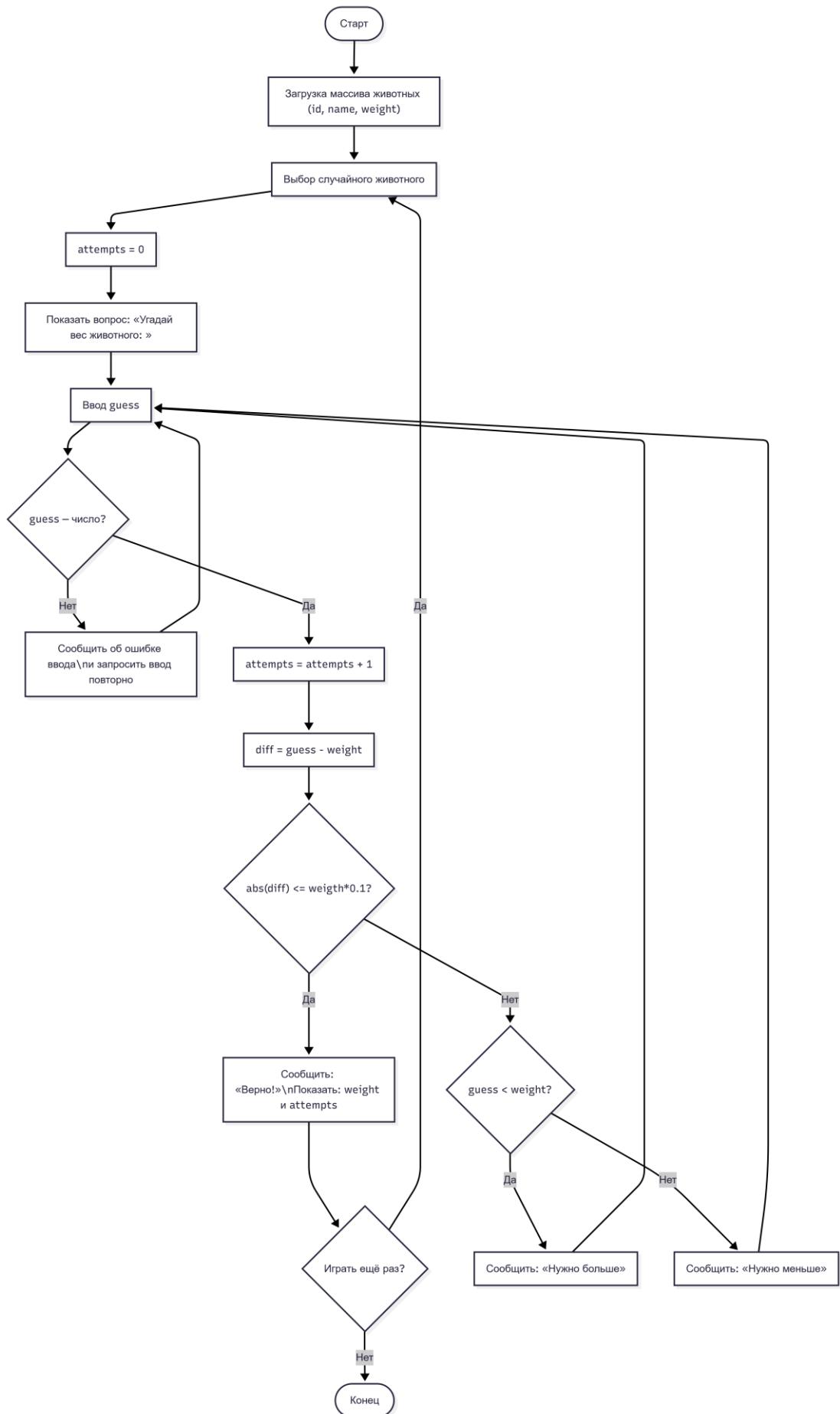


Рисунок 2. Алгоритм процесса угадывания веса животного.

В целом, разработанная архитектура разделяет приложение на логические части, что облегчает поддержку и развитие. Например, чтобы добавить новых животных или изменить параметры уровней, достаточно отредактировать данные в модуле `data.js`. Логика подсчёта очков и перехода между уровнями инкапсулирована в функциях `handleLevelComplete()` и `moveToNextLevel()`, что позволяет при необходимости модифицировать правила, не затрагивая остальные части кода. Благодаря использованию стандартных API браузера [1][3][4] и простых HTML/CSS-технологий [2], приложение не зависит от сторонних библиотек и работает быстро даже на устройствах со скромной производительностью.

Тестирование

Разработанное приложение прошло тщательное тестирование, включающее проверку функциональности всех игровых элементов, граничных случаев ввода и корректности работы на разных уровнях сложности. Тестирование проводилось методом черного ящика – через непосредственное взаимодействие с интерфейсом – и отладочным способом – с анализом состояния через консоль разработчика.

Функциональное тестирование игрового процесса:

- Проверена логика угадывания на всех уровнях. На лёгком уровне целенаправленно вводились как правильные ответы, так и ошибочные. Убедились, что при верном ответе игра выдаёт сообщение с фактическим весом («Верно! Вес ... $\approx N$ кг.»), увеличивает счёт и переходит к следующему животному; при неверном – выводится соответствующая подсказка («Слишком много/мало») и уменьшается индикатор попыток. На средней и сложной сложности повторили тест аналогично, проследив уменьшение количества доступных попыток и времени. Во всех случаях логика сработала ожидаемо: счётчики попыток и таймер обнуляются при переходе к новому заданию, подсказки отображаются корректно, новые животные не повторяются в рамках одной игры.
- Тестирование ситуации проигрыша: последовательно вводились заведомо неверные ответы, превышающие лимит попыток. Приложение правильно зафиксировало окончание игры, отобразило сообщение «Игра окончена. Вы не угадали вес животного.» и предложило начать заново. При нажатии кнопки «Начать заново» состояние сбрасывается и игрок перенаправляется в меню – это протестировано и работает без сбоев.
- Тестирование окончания уровня: вводились правильные ответы для всех 5 животных уровня. После пятого успешного угадывания появилось окно с заголовком «Уровень пройден!» и сообщением о базовых и итоговых очках. Проверено, что очки вычислены верно (например, на лёгком уровне без таймера итог равен базовым, на среднем – базовые удваиваются, на сложном – утраиваются, а с таймером – дополнительно удваиваются). Опция «Продолжить уровень» появляется только после 1-го и 2-го уровней, и при нажатии на неё игра корректно переходит на следующий уровень сложности. После третьего уровня опция продолжения отсутствует, и диалог предлагает сохранить результат.
- Тестирование таблицы рекордов: после завершения игры (как с победой, так и с поражением) нажималась кнопка «Сохранить результат». Приложение переключалось на экран рейтинга, где новый результат появлялся в списке. Проверено добавление разных результатов: если добавить результат с большим счётом, он отображается выше в таблице (таблица сортируется по убыванию очков). Также добавлено более 20 результатов для проверки усечения списка – хранятся только топ-20,

старые результаты выходят за пределы списка, что соответствует требованиям.

Тестирование ввода и валидации:

- Поле ввода имени игрока: проверено, что без ввода имени игра не начинается – появляется сообщение «*Введите имя игрока.*» под полем и фокус возвращается на него. Только после заполнения имени кнопка «Начать игру» успешно запускает процесс. Обработка пробельных символов: введение имени, состоящего из одних пробелов, не принимается (скрипт использует trim()), поэтому пустая строка после обрезки пробелов считается невалидной).
- Поле ввода веса: проверено поведение при нечисловых символах. Так как поле имеет тип number, браузер сам ограничивает ввод нецифровых символов. Пустой ввод или 0 отображают подсказку «*Введите корректное число.*» при попытке проверки – это сработало правильно. Отрицательные значения также не принимаются (при вводе «-5» поле считает это невалидным, а логика handleSubmit дополнительно отфильтровывает value ≤ 0). При больших числах (например, очевидно превышающих вес животного в десятки раз) игра просто обрабатывает их как неправильный ответ, давая подсказку «*Слишком много!*».
- Проверен режим с таймером: при активации переключателя «*Таймер*» перед стартом были проведены игры, специально затягивая ответ дольше лимита. Таймер отображается (формат mm:ss) и достигает 00:00, после чего автоматически срабатывает как неправильная попытка: выводится сообщение «*Время вышло. Попытка потрачена.*», попытки уменьшаются на единицу. Интерфейс при этом обновляется к следующей попытке, и таймер вновь начинает отсчет. Этот цикл протестирован на всех уровнях. Также проверено, что при паузах между вводом (например, ничего не вводить до истечения времени) всё равно срабатывает механизм потери попытки. Таким образом, таймер выполняет свою функцию, делая игру динамичной. Время отсчитывается отдельно на каждую попытку и сбрасывается при переходе к новому животному – эти моменты тоже проверены.
- Кросс-браузерное тестирование: приложение вручную запущено в последних версиях браузеров Google Chrome, Mozilla Firefox и Microsoft Edge на настольном компьютере. Во всех тестируемых браузерах функциональность работала одинаково, верстка отображалась корректно. Также было проведено пробное открытие на смартфоне (Chrome, Android) – адаптивный дизайн позволил интерфейсу перестроиться для узкого экрана (панель заголовка складывается, элементы масштабируются). Основной игровой процесс на мобильном устройстве также функционирует, хотя ввод чисел требует вызова экранной клавиатуры. В дальнейшем планируется дополнительная оптимизация под мобильные

устройства, но в целом даже в текущем виде приложение работоспособно на различных платформах.

Автоматизированное нефункциональное тестирование (производительность, качество и безопасность)

Помимо функциональных проверок, в рамках тестирования целесообразно выполнять автоматизированную оценку нефункциональных характеристик веб-приложения: производительности, соблюдения практик качества и базовых требований безопасности. Для этого используется инструмент **Google Lighthouse** — открытый набор автоматизированных аудитов, предназначенный для повышения качества веб-страниц и веб-приложений.

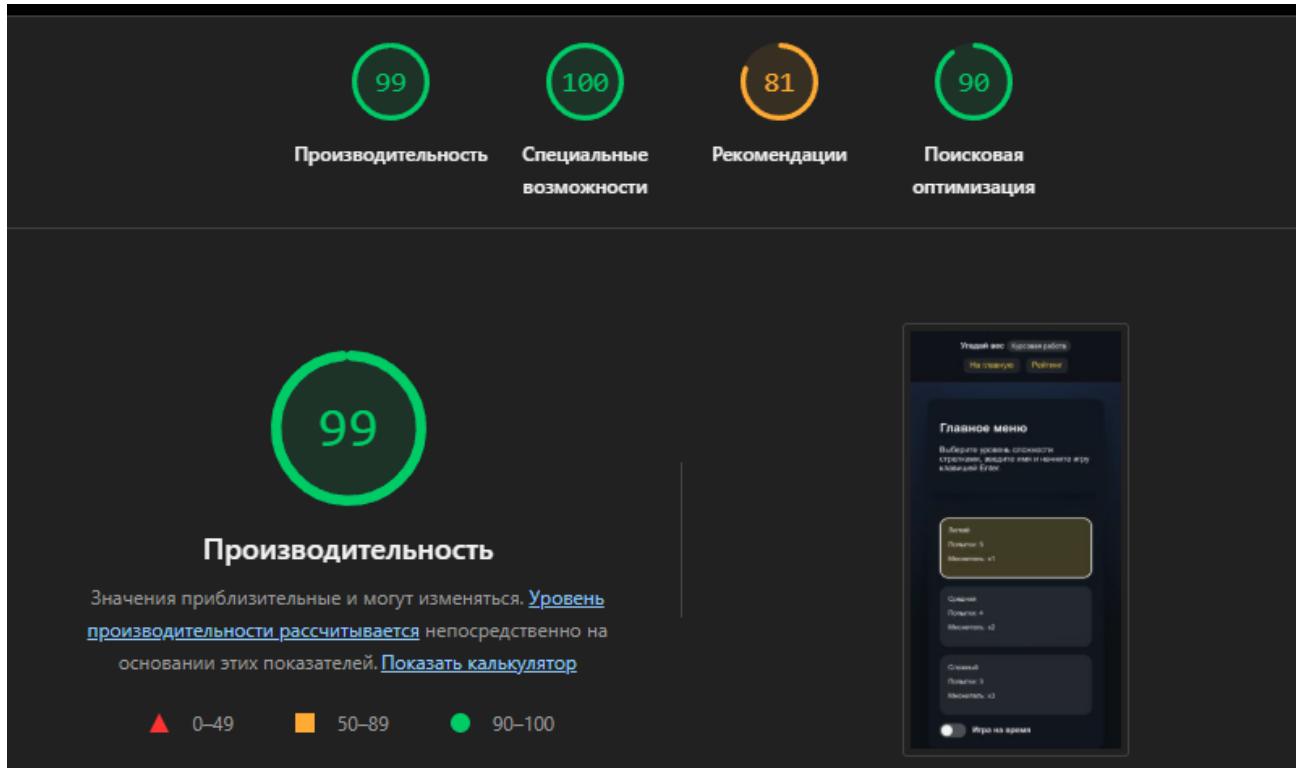


Рисунок 3. Результаты тестирования в Lighthouse

1) Аудит качества и производительности (Lighthouse). Lighthouse выполняет серию проверок и формирует отчет по нескольким направлениям (в зависимости от конфигурации): **Performance**, **Accessibility**, **Best Practices**, **SEO**. Отчет содержит агрегированные оценки, список найденных проблем, а также рекомендаций по их устранению. В части производительности Lighthouse оценивает поведение приложения в условиях, приближенных к реальным пользовательским сценариям, и опирается на показатели загрузки и отзывчивости интерфейса. В качестве ключевых ориентиров применяются метрики семейства Web Vitals; например, **Largest Contentful Paint (LCP)** характеризует скорость отображения основного контента страницы и используется как один из важных индикаторов восприятия быстродействия пользователем. 2) Предотвращение регрессий (Lighthouse CI). Чтобы сделать нефункциональные проверки воспроизводимыми и отслеживать

деградации между версиями, аудит Lighthouse рекомендуется запускать в автоматическом режиме на каждый существенный релиз/коммит в контуре CI/CD. Для этого применяется **Lighthouse CI**, позволяющий автоматизировать прогоны Lighthouse, сравнивать результаты между сборками и блокировать изменения при ухудшении показателей относительно заданных порогов.

На практике устанавливаются *пороговые значения* (quality gates), например:

- минимально допустимая оценка **Performance** и/или верхние границы по ключевым метрикам;
- отсутствие критических нарушений в **Best Practices**;
- отсутствие регрессий относительно базовой (эталонной) сборки.

3) Базовые проверки безопасности на уровне HTTP-заголовков. Отдельным классом нефункциональных проверок являются настройки безопасности доставки контента, которые непосредственно влияют на устойчивость приложения к типовым веб-атакам и изоляцию контекстов. В рамках тестирования рекомендуется валидировать наличие и корректность ключевых защитных заголовков, в частности:

- **X-Frame-Options** — используется для ограничения встраивания страницы во фреймы и снижения риска атак класса clickjacking.

Cross-Origin-Opener-Policy (COOP) — задает политику изоляции контекста окна/вкладки и контролирует взаимодействие с окнами из других origin.

Cross-Origin-Embedder-Policy (COEP) — определяет правила загрузки кросс-доменных ресурсов и применяется совместно с COOP для усиления изоляции.

Проверка данных заголовков может выполняться автоматически (например, отдельным тестом, который делает HTTP-запрос к целевым страницам и сравнивает фактические значения заголовков с ожидаемой политикой окружения).

Результаты тестирования подтверждают, что все заявленные требования к функциональности выполнены. Игра корректно обрабатывает как типичные сценарии (успешное прохождение, частично неверные ответы), так и исключительные ситуации (пустой ввод, выход времени, отсутствие имени и пр.), не допуская сбоев или некорректного состояния. Система начисления очков и сохранения рекордов функционирует по описанным правилам. Данные сохраняются между сессиями — после закрытия и повторного открытия страницы таблица лидеров остается доступной, благодаря сохранению в localStorage. Таким образом, можно заключить, что приложение достигло необходимого уровня надежности и готово для использования конечными пользователями.

Заключение

В ходе выполнения курсовой работы была разработана и протестирована интерактивная веб-игра «Определи вес». Приложение позволяет пользователю угадывать вес различных животных, постепенно повышая уровень сложности. В процессе работы были достигнуты все поставленные цели: создан удобный интерфейс, реализована игровая логика с ограниченными попытками и подсказками, предусмотрен режим с таймером для усложнения игры, а также реализована система подсчёта очков и таблица рекордов.

Разработка данного проекта позволила закрепить на практике навыки верстки и прототипирования веб-интерфейсов, полученные в рамках курса. Были использованы современные возможности HTML5, CSS3 (включая гибкую верстку на Grid/Flexbox и анимации) и возможности JavaScript ES6+ (модули, работа с DOM, Web Storage API). Проект продемонстрировал значимость разделения кода на модули: модульная структура облегчила тестирование отдельных частей и потенциально позволит в будущем расширять функциональность (например, добавлять новые уровни сложности или другие категории объектов для угадывания).

Основные выводы по результатам разработки:

- При создании веб-приложений игрового типа важно обеспечить плавный пользовательский опыт – в данной работе это достигнуто за счёт своевременных визуальных подсказок, анимаций и обработки вводов (как кликов, так и нажатий клавиш), что делает взаимодействие интуитивным.
- Хранение состояния на стороне клиента (в браузере) оказалось удобным решением для односторонней игры: localStorage позволяет передавать данные между этапами без задействования сервера, что существенно упростило архитектуру. Однако следует помнить о специфике Web Storage – например, данные хранятся локально и не предназначены для защиты от мошенничества, что приемлемо для некоммерческой игры, но требует иных подходов в серьёзных приложениях.
- Тестирование выявило, что даже небольшое приложение имеет множество сценариев использования, которые необходимо учесть. Применение методического пошагового тестирования (в том числе экстремальные случаи ввода) позволило сделать игру более устойчивой и удобной (например, добавлена обработка Enter на экранах, улучшена читаемость сообщений об ошибках ввода).
- В результате проделанной работы было получено работающее приложение, которое может быть размещено на веб-странице и доступно для широкой аудитории. Код игры обладает достаточной гибкостью для дальнейших улучшений. Например, возможно расширение базы животных, локализация игры на другие языки, добавление звукового

сопровождения при правильных/неправильных ответах, что повысит привлекательность продукта.

Подводя итог, разработка игрового приложения «Определи вес» продемонстрировала успешное применение теоретических знаний по верстке и прототипированию сайтов на практике. Итоговый продукт соответствует заданным требованиям и может использоваться как демонстрация навыков создания интерактивных веб-страниц, а также как основа для дальнейших творческих экспериментов в области web-разработки.

Список использованных источников

1. **MDN Web Docs. HTML: HyperText Markup Language** – документация по языку HTML (элементы, структура документа) [Электронный ресурс]. URL: <https://developer.mozilla.org/ru/docs/Web/HTML> (дата обращения: 13.01.2026).
2. **MDN Web Docs. CSS: Каскадные таблицы стилей** – документация по CSS, описание свойств, селекторов и современных возможностей верстки [Электронный ресурс]. URL: <https://developer.mozilla.org/ru/docs/Web/CSS> (дата обращения: 13.01.2026).
3. **MDN Web Docs. JavaScript** – справочная информация по языку JavaScript, описание синтаксиса и встроенных объектов [Электронный ресурс]. URL: <https://developer.mozilla.org/ru/docs/Web/JavaScript> (дата обращения: 13.01.2026).
4. **MDN Web Docs. Window.localStorage (Web Storage API)** – описание Web Storage API и свойства localStorage для хранения данных в браузере [Электронный ресурс]. URL: <https://developer.mozilla.org/ru/docs/Web/API/Window/localStorage> (дата обращения: 13.01.2026).
5. **HTML Living Standard** – актуальный стандарт HTML (WHATWG Living Standard) [Электронный ресурс]. URL: <https://html.spec.whatwg.org/> (дата обращения: 13.01.2026).

Приложения

Приложение А. Исходный код игрового приложения

Листинг A.1 – Файл data.js (данные игры и вспомогательные функции).

```
export const STORAGE_KEYS = {
  state: "gw_state",
  leaderboard: "gw_leaderboard",
};

export const difficulties = {
  easy: {
    id: "easy",
    label: "Легкий",
    attempts: 5,
    multiplier: 1,
    timeLimit: 60,
  },
  medium: {
    id: "medium",
    label: "Средний",
    attempts: 4,
    multiplier: 2,
    timeLimit: 45,
  },
  hard: {
    id: "hard",
    label: "Сложный",
    attempts: 3,
    multiplier: 3,
    timeLimit: 30,
  },
};

export const difficultyOrder = ["easy", "medium", "hard"];

export const animals = [
  { id: "lynx", name: "Рысь", weight: 27, color: "#f08a5d" },
  { id: "wolf", name: "Волк", weight: 45, color: "#b83b5e" },
  { id: "fox", name: "Лиса", weight: 6, color: "#6a2c70" },
  { id: "moose", name: "Лось", weight: 408, color: "#355c7d" },
  { id: "bear", name: "Бурый медведь", weight: 185, color: "#f67280" },
  { id: "boar", name: "Кабан", weight: 79, color: "#c06c84" },
  { id: "eagle", name: "Орлан", weight: 4.9, color: "#355c7d" },
  { id: "owl", name: "Сова", weight: 2.7, color: "#99b898" },
  { id: "seal", name: "Тюлень", weight: 82, color: "#2a363b" },
  { id: "dolphin", name: "Дельфин", weight: 175, color: "#00a8cc" },
  { id: "camel", name: "Верблюд", weight: 475, color: "#f8b400" },
  { id: "elephant", name: "Слон", weight: 5300, color: "#6c5b7b" },
  { id: "giraffe", name: "Жираф", weight: 1010, color: "#c06c84" },
  { id: "kangaroo", name: "Кенгуру", weight: 51, color: "#f67280" },
];
```

```

    { id: "panda", name: "Панда", weight: 100, color: "#355c7d" },
    { id: "tiger", name: "Тигр", weight: 195, color: "#ff847c" },
    { id: "horse", name: "Лошадь", weight: 520, color: "#2a9d8f" },
    { id: "rhino", name: "Носорог", weight: 1900, color: "#e76f51" },
    { id: "hippo", name: "Бегемот", weight: 2113, color: "#264653" },
    { id: "zebra", name: "Зебра", weight: 242, color: "#118ab2" },
];
}

const initialLeaderboard = [
    { name: "Алиса", score: 68, difficulty: "Сложный", timed: true },
    { name: "Марко", score: 64, difficulty: "Сложный", timed: true },
    { name: "Соня", score: 61, difficulty: "Сложный", timed: false },
    { name: "Кирилл", score: 59, difficulty: "Сложный", timed: true },
    { name: "Никита", score: 56, difficulty: "Сложный", timed: false },
    { name: "Полина", score: 54, difficulty: "Сложный", timed: true },
    { name: "Артем", score: 52, difficulty: "Сложный", timed: false },
    { name: "Яна", score: 50, difficulty: "Сложный", timed: true },
    { name: "Денис", score: 47, difficulty: "Средний", timed: true },
    { name: "Ева", score: 46, difficulty: "Средний", timed: true },
    { name: "Олег", score: 45, difficulty: "Средний", timed: false },
    { name: "Роман", score: 43, difficulty: "Средний", timed: true },
    { name: "Ирина", score: 42, difficulty: "Средний", timed: false },
    { name: "Даша", score: 41, difficulty: "Средний", timed: true },
    { name: "Сергей", score: 39, difficulty: "Средний", timed: false },
    { name: "Вика", score: 36, difficulty: "Легкий", timed: true },
    { name: "Глеб", score: 35, difficulty: "Легкий", timed: false },
    { name: "Лиза", score: 33, difficulty: "Легкий", timed: true },
    { name: "Максим", score: 32, difficulty: "Легкий", timed: false },
    { name: "Илья", score: 30, difficulty: "Легкий", timed: false },
];
}

export function initLeaderboard() {
    if (!localStorage.getItem(STORAGE_KEYS.leaderboard)) {
        localStorage.setItem(
            STORAGE_KEYS.leaderboard,
            JSON.stringify(initialLeaderboard)
        );
    }
}

export function getLeaderboard() {
    initLeaderboard();
    try {
        return JSON.parse(localStorage.getItem(STORAGE_KEYS.leaderboard)) ||
    [];
    } catch (error) {
        return [];
    }
}

export function saveLeaderboard(entries) {

```

```

        localStorage.setItem(STORAGE_KEYS.leaderboard,
JSON.stringify(entries));
}

export function addLeaderboardEntry(entry) {
const entries = getLeaderboard();
entries.push(entry);
entries.sort((a, b) => b.score - a.score);
const trimmed = entries.slice(0, 20);
saveLeaderboard(trimmed);
return trimmed;
}

export function pickRandomAnimals(excludedIds, count) {
const available = animals.filter(
  (animal) => !excludedIds.includes(animal.id)
);
const shuffled = [...available];
for (let index = shuffled.length - 1; index > 0; index -= 1) {
  const swapIndex = Math.floor(Math.random() * (index + 1));
  [shuffled[index], shuffled[swapIndex]] = [
    shuffled[swapIndex],
    shuffled[index],
  ];
}
return shuffled.slice(0, count).map((animal) => animal.id);
}

```

Листинг A.2 – Файл menu.html (страница главного меню игры).

```

<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Угадай вес – Главное меню</title>
    <link rel="stylesheet" href="styles.css" />
  </head>
  <body>
    <header>
      <div>
        <strong>Угадай вес</strong>
        <div class="badge">Курсовая работа</div>
      </div>
      <nav>
        <a href="../index.html">На главную</a>
        <a href="leaderboard.html">Рейтинг</a>
      </nav>
    </header>
    <main>
      <section class="panel">

```

```

<h1>Главное меню</h1>
<p>
    Выберите уровень сложности стрелками, введите имя и начните игру
    клавишей Enter.
</p>
</section>

<section class="panel">
    <div class="menu-grid" id="difficulty-options"></div>
    <div class="toggle-row">
        <label class="switch" for="timed-toggle">
            <input type="checkbox" id="timed-toggle" />
            <span class="switch-track"></span>
            <span class="switch-label">Игра на время</span>
        </label>
    </div>
</section>

<section class="panel">
    <label for="player-name">Имя игрока</label>
    <input
        type="text"
        id="player-name"
        placeholder="Например, Игрок1"
        autocomplete="off"
    />
    <p class="error" id="name-error"></p>
    <div class="menu-actions">
        <button class="primary" id="start-game">Начать игру</button>
        <a      class="secondary"     id="leaderboard-link"
           href="leaderboard.html"
           >Таблица лидеров</a>
    </div>
</section>
</main>
<script type="module" src="menu.js"></script>
</body>
</html>

```

Листинг A.3 – Файл menu.js (логика главного меню, выбор настроек и запуск игры).

```

import {
    STORAGE_KEYS,
    difficulties,
    difficultyOrder,
    initLeaderboard,
    pickRandomAnimals,
} from "./data.js";

```

```

const difficultyContainer = document.querySelector("#difficulty-options");
const nameInput = document.querySelector("#player-name");
const nameError = document.querySelector("#name-error");
const timeToggle = document.querySelector("#timed-toggle");
const startButton = document.querySelector("#start-game");
const leaderboardLink = document.querySelector("#leaderboard-link");

initLeaderboard();

let selectedIndex = 0;
let menuFocusIndex = 0;
const difficultyButtons = [];

const renderDifficultySettings = () => {
    difficultyContainer.innerHTML = "";
    difficultyButtons.length = 0;

    difficultyOrder.forEach((key, index) => {
        const settings = difficulties[key];
        const card = document.createElement("button");
        card.type = "button";
        card.className = "difficulty-card";
        card.dataset.difficulty = key;
        card.innerHTML = `
            <div class="difficulty-title">${settings.label}</div>
            <p>Попытки: ${settings.attempts}</p>`;

        if (timeToggle.checked) {
            card.innerHTML += `
                <p>Таймер: ${settings.timeLimit} сек.</p>
                <p>Множитель: x${settings.multiplier * 2}</p>
            `;
        } else {
            card.innerHTML += `
                <p>Множитель: x${settings.multiplier}</p>
            `;
        }

        card.addEventListener("click", () => {
            selectDifficulty(index);
            setMenuFocus(0);
        });
        card.addEventListener("focus", () => setMenuFocus(0));
        difficultyButtons.push(card);
        difficultyContainer.appendChild(card);
    });
    selectDifficulty(selectedIndex);
    if (menuFocusIndex === 0) {
        focusCurrentMenuItem();
    }
}

```

```

};

renderDifficultySettings();
timeToggle.addEventListener("change", renderDifficultySettings);

function selectDifficulty(index) {
  selectedIndex = index;
  difficultyButtons.forEach((button, idx) => {
    button.classList.toggle("is-selected", idx === selectedIndex);
  });
}

function createInitialState() {
  const difficultyKey = difficultyOrder[selectedIndex];
  const levelAnimals = pickRandomAnimals([], 5);
  return {
    playerName: nameInput.value.trim(),
    difficulty: difficultyKey,
    timeMode: timeToggle.checked,
    highestDifficulty: difficultyKey,
    baseScore: 0,
    usedAnimals: [...levelAnimals],
    levelAnimals,
    currentIndex: 0,
    attemptsLeft: difficulties[difficultyKey].attempts,
    failedAttemptsCurrent: 0,
    completedAnimals: 0,
  };
}

function startGame() {
  const trimmedName = nameInput.value.trim();
  if (!trimmedName) {
    nameError.textContent = "Введите имя игрока.";
    nameInput.focus();
    return;
  }
  nameError.textContent = "";
  const state = createInitialState();
  localStorage.setItem(STORAGE_KEYS.state, JSON.stringify(state));
  window.location.href = "game.html";
}

startButton.addEventListener("click", startGame);

function setMenuFocus(index) {
  menuFocusIndex = Math.max(0, Math.min(index, 4));
}

function focusCurrentMenuItem() {
  if (menuFocusIndex === 0) {

```

```

        difficultyButtons[selectedIndex]?.focus();
        return;
    }
    if (menuFocusIndex === 1) {
        timeToggle.focus();
        return;
    }
    if (menuFocusIndex === 2) {
        nameInput.focus();
        return;
    }
    if (menuFocusIndex === 3) {
        startButton.focus();
        return;
    }
    leaderboardLink?.focus();
}

function moveMenuFocus(delta) {
    setMenuFocus(menuFocusIndex + delta);
    focusCurrentMenuItem();
}

nameInput.addEventListener("focus", () => setMenuFocus(2));
timeToggle.addEventListener("focus", () => setMenuFocus(1));
startButton.addEventListener("focus", () => setMenuFocus(3));
leaderboardLink?.addEventListener("focus", () => setMenuFocus(4));

window.addEventListener("keydown", (event) => {
    if (event.key === "ArrowDown") {
        event.preventDefault();
        moveMenuFocus(1);
        return;
    }
    if (event.key === "ArrowUp") {
        event.preventDefault();
        moveMenuFocus(-1);
        return;
    }
    if (event.key === "ArrowRight" && menuFocusIndex === 0) {
        event.preventDefault();
        selectDifficulty((selectedIndex + 1) % difficultyButtons.length);
        focusCurrentMenuItem();
        return;
    }
    if (event.key === "ArrowLeft" && menuFocusIndex === 0) {
        event.preventDefault();
        selectDifficulty(
            (selectedIndex - 1 + difficultyButtons.length) %
difficultyButtons.length
        );
    }
})

```

```

        focusCurrentMenuItem();
        return;
    }
    if (event.key === "Enter") {
        event.preventDefault();
        if (menuFocusIndex === 0) {
            selectDifficulty((selectedIndex + 1) % difficultyButtons.length);
            focusCurrentMenuItem();
            return;
        }
        if (menuFocusIndex === 1) {
            timeToggle.checked = !timeToggle.checked;
            timeToggle.dispatchEvent(new Event("change"));
            focusCurrentMenuItem();
            return;
        }
        if (menuFocusIndex === 2) {
            setMenuFocus(3);
            focusCurrentMenuItem();
            return;
        }
        if (menuFocusIndex === 3) {
            startGame();
            return;
        }
        leaderboardLink?.click();
    }
});

focusCurrentMenuItem();

```

Листинг А.4 – Файл game.html (игровой экран, интерфейс процесса угадывания).

```

<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>Угадай вес – Игра</title>
        <link rel="stylesheet" href="styles.css" />
    </head>
    <body>
        <header>
            <div>
                <strong>Угадай вес</strong>
                <div class="badge" id="level-badge">Уровень</div>
            </div>
            <nav>
                <a href="menu.html">Главное меню</a>
                <a href="leaderboard.html">Рейтинг</a>
            </nav>
        </header>

```

```

        </nav>
    </header>
    <main>
        <section class="panel game-header">
            <div class="stat">
                <strong>Игрок:</strong> <span id="player-name"></span>
            </div>
            <div class="stat">
                <strong>Очки:</strong> <span id="current-score">0</span>
            </div>
            <div class="stat">
                <strong>Прогресс:</strong> <span id="progress"></span>
            </div>
            <div class="stat">
                <strong>Попытки:</strong>
                <div class="attempts" id="attempts-list"></div>
            </div>
            <div class="stat" id="timer-block">
                <strong>Таймер:</strong> <span id="timer-value">--:--</span>
            </div>
        </section>

        <section class="panel">
            <div class="card-stage" id="card-stage"></div>
            <p class="hint" id="hint-text"></p>
        </section>

        <section class="panel">
            <label for="weight-input">Введите вес животного (кг)</label>
            <input type="number" id="weight-input" min="0" />
            <div class="menu-actions game-actions">
                <button class="primary" id="submit-weight">Проверить вес</button>
                <button class="secondary" id="give-up">Выйти в меню</button>
            </div>
        </section>
    </main>

    <div class="overlay hidden" id="overlay">
        <div class="modal" role="dialog" aria-modal="true">
            <h2 id="modal-title"></h2>
            <p id="modal-message"></p>
            <div class="modal-actions" id="modal-actions"></div>
        </div>
    </div>

    <script type="module" src="game.js"></script>
</body>
</html>

```

Листинг A.5 – Файл game.js (основная логика игры: обработка попыток, подсказки, переход уровней).

```

import {
  STORAGE_KEYS,
  difficulties,
  difficultyOrder,
  animals,
  addLeaderboardEntry,
  pickRandomAnimals,
} from "./data.js";

const playerNameEl = document.querySelector("#player-name");
const scoreEl = document.querySelector("#current-score");
const progressEl = document.querySelector("#progress");
const attemptsList = document.querySelector("#attempts-list");
const timerBlock = document.querySelector("#timer-block");
const timerValue = document.querySelector("#timer-value");
const cardStage = document.querySelector("#card-stage");
const hintText = document.querySelector("#hint-text");
const weightInput = document.querySelector("#weight-input");
const submitButton = document.querySelector("#submit-weight");
const giveUpButton = document.querySelector("#give-up");
const overlay = document.querySelector("#overlay");
const modalTitle = document.querySelector("#modal-title");
const modalMessage = document.querySelector("#modal-message");
const modalActions = document.querySelector("#modal-actions");
const levelBadge = document.querySelector("#level-badge");

const animalsById = new Map(animals.map((animal) => [animal.id, animal]));

let state = loadState();
let timerId = null;
let timeRemaining = null;

if (!state) {
  window.location.href = "menu.html";
}

playerNameEl.textContent = state.playerName;

function loadState() {
  try {
    const raw = localStorage.getItem(STORAGE_KEYS.state);
    return raw ? JSON.parse(raw) : null;
  } catch (error) {
    return null;
  }
}

function saveState() {
  localStorage.setItem(STORAGE_KEYS.state, JSON.stringify(state));
}

```

```

function getDifficultySettings() {
    return difficulties[state.difficulty];
}

function formatTime(seconds) {
    const mins = String(Math.floor(seconds / 60)).padStart(2, "0");
    const secs = String(seconds % 60).padStart(2, "0");
    return `${mins}:${secs}`;
}

function updateHeader() {
    scoreEl.textContent = state.baseScore;
    progressEl.textContent = `${state.currentIndex + 1} / ${state.levelAnimals.length}`;
    levelBadge.textContent = `Уровень: ${getDifficultySettings().label}`;
    renderAttempts();
    timerBlock.style.display = state.timeMode ? "block" : "none";
}

function renderAttempts() {
    attemptsList.innerHTML = "";
    const total = getDifficultySettings().attempts;
    for (let i = 0; i < total; i += 1) {
        const dot = document.createElement("span");
        dot.className = "attempt-dot";
        if (i < state.attemptsLeft) {
            dot.classList.add("is-remaining");
        }
        attemptsList.appendChild(dot);
    }
}

function adjustStageHeight(card) {
    if (!card) {
        return;
    }
    const height = card.offsetHeight;
    if (height) {
        cardStage.style.height = `${height}px`;
    }
}

function createAnimalCard(animal) {
    const card = document.createElement("div");
    card.className = "animal-card";

    const title = document.createElement("h2");
    title.textContent = animal.name;

    const media = document.createElement("div");

```

```

media.className = "animal-media";

const image = document.createElement("img");
image.className = "animal-image";
image.src = `./assets/${animal.id}.jpg`;
image.alt = `Фото: ${animal.name}`;

const placeholder = document.createElement("div");
placeholder.className = "animal-placeholder";
placeholder.style.background = animal.color;
placeholder.textContent = "Фото";

image.addEventListener("load", () => {
  placeholder.classList.add("is-hidden");
  adjustStageHeight(card);
});
image.addEventListener("error", () => {
  image.remove();
  placeholder.textContent = "Фото недоступно";
  adjustStageHeight(card);
});

media.append(image, placeholder);
card.append(title, media);
return card;
}

function swapCard(animal) {
  const existing = cardStage.querySelector(".animal-card");
  if (existing) {
    existing.classList.remove("enter", "enter-active");

    existing.classList.add("exit");
    existing.addEventListener(
      "transitionend",
      () => {
        existing.remove();
      },
      { once: true }
    );
  }
  const newCard = createAnimalCard(animal);
  newCard.classList.add("enter");
  cardStage.appendChild(newCard);
  requestAnimationFrame(() => {
    newCard.classList.add("enter-active");
    adjustStageHeight(newCard);
  });
}

function startTimer() {

```

```

clearTimer();
if (!state.timeMode) {
    timerValue.textContent = "---";
    return;
}
timeRemaining = getDifficultySettings().timeLimit;
timerValue.textContent = formatTime(timeRemaining);
timerId = setInterval(() => {
    timeRemaining -= 1;
    if (timeRemaining <= 0) {
        clearTimer();
        handleTimeout();
        return;
    }
    timerValue.textContent = formatTime(timeRemaining);
}, 1000);
}

function clearTimer() {
    if (timerId) {
        clearInterval(timerId);
        timerId = null;
    }
}

function handleTimeout() {
    state.attemptsLeft -= 1;
    state.failedAttemptsCurrent += 1;
    hintText.textContent = "Время вышло. Попытка потрачена.";
    if (state.attemptsLeft <= 0) {
        handleLoss();
        return;
    }
    updateHeader();
    saveState();
    startTimer();
}

function currentAnimal() {
    return animalsById.get(state.levelAnimals[state.currentIndex]);
}

function handleCorrectGuess(animal) {
    const pointsForAnimal = Math.max(0, 10 - state.failedAttemptsCurrent * 2);
    state.baseScore += pointsForAnimal;
    state.completedAnimals += 1;
    state.currentIndex += 1;
    if (state.currentIndex >= state.levelAnimals.length) {
        handleLevelComplete();
        return;
    }
}

```

```

    }
    state.failedAttemptsCurrent = 0;
    state.attemptsLeft = getDifficultySettings().attempts;
    hintText.textContent = `Верно! Вес ${animal.name} ≈ ${animal.weight}
кг.`;
    updateHeader();
    saveState();
    swapCard(currentAnimal());
    startTimer();
}

function handleIncorrectGuess(isHigh) {
    state.attemptsLeft -= 1;
    state.failedAttemptsCurrent += 1;
    if (state.attemptsLeft <= 0) {
        hintText.textContent = "Попытки закончились.";
        handleLoss();
        return;
    }
    hintText.textContent = isHigh
        ? "Слишком много! Попробуйте меньше."
        : "Слишком мало! Попробуйте больше.";
    updateHeader();
    saveState();
    startTimer();
}

function handleLevelComplete() {
    clearTimer();
    updateHeader();
    const canContinue = state.difficulty !== "hard";
    const finalScore = computeFinalScore();
    showModal({
        title: "Уровень пройден!",
        message: `Поздравляем! Базовые очки: ${state.baseScore}. Итог с
множителями: ${finalScore}.`,
        canContinue,
        isWin: true,
    });
}

function handleLoss() {
    clearTimer();
    updateHeader();
    const finalScore = computeFinalScore();
    showModal({
        title: "Игра окончена",
        message: `Вы не угадали вес животного. Итоговые очки: ${finalScore}.`,
        canContinue: false,
        isWin: false,
    });
}

```

```

}

function computeFinalScore() {
  const highestMultiplier =
    difficulties[state.highestDifficulty]?.multiplier || 1;
  const timedMultiplier = state.timeMode ? 2 : 1;
  return state.baseScore * highestMultiplier * timedMultiplier;
}

function showModal({ title, message, canContinue, isWin }) {
  overlay.classList.remove("hidden");
  modalTitle.textContent = title;
  modalMessage.textContent = message;
  modalActions.innerHTML = "";

  if (canContinue) {
    const continueButton = document.createElement("button");
    continueButton.className = "primary";
    continueButton.textContent = "Продолжить уровень";
    continueButton.addEventListener("click", () => {
      overlay.classList.add("hidden");
      moveToNextLevel();
    });
    modalActions.appendChild(continueButton);
  }

  const saveButton = document.createElement("button");
  saveButton.className = canContinue ? "secondary" : "primary";
  saveButton.textContent = "Сохранить результат";
  saveButton.addEventListener("click", () => {
    saveScore();
    window.location.href = "leaderboard.html";
  });
  modalActions.appendChild(saveButton);

  const exitButton = document.createElement("button");
  exitButton.className = "secondary";
  exitButton.textContent = isWin ? "В меню" : "Начать заново";
  exitButton.addEventListener("click", () => {
    localStorage.removeItem(STORAGE_KEYS.state);
    window.location.href = "menu.html";
  });
  modalActions.appendChild(exitButton);

  const focusTarget = modalActions.querySelector("button");
  if (focusTarget) {
    focusTarget.focus();
  }
}

function moveToNextLevel() {

```

```

const currentIndex = difficultyOrder.indexOf(state.difficulty);
const nextDifficulty = difficultyOrder[currentIndex + 1];
if (!nextDifficulty) {
    return;
}
state.difficulty = nextDifficulty;
state.highestDifficulty = nextDifficulty;
state.levelAnimals = pickRandomAnimals(state.usedAnimals, 5);
state.usedAnimals = [...state.usedAnimals, ...state.levelAnimals];
state.baseScore = 0;
state.currentIndex = 0;
state.completedAnimals = 0;
state.failedAttemptsCurrent = 0;
state.attemptsLeft = difficulties[nextDifficulty].attempts;
hintText.textContent = "Новый уровень!";
updateHeader();
saveState();
swapCard(currentAnimal());
startTimer();
}

function saveScore() {
    const highestLabel = difficulties[state.highestDifficulty]?.label || "";
    addLeaderboardEntry({
        name: state.playerName,
        score: computeFinalScore(),
        difficulty: highestLabel,
        timed: state.timeMode,
    });
}

function handleSubmit() {
    const value = Number(weightInput.value);
    console.log("handleSubmit");
    if (!value || value <= 0) {
        hintText.textContent = "Введите корректное число.";
        return;
    }
    const animal = currentAnimal();
    if (!animal) {
        return;
    }
    clearTimer();
    weightInput.value = "";
    const tolerance = animal.weight * 0.1;
    const isCorrect = Math.abs(animal.weight - value) <= tolerance;
    if (isCorrect) {
        handleCorrectGuess(animal);
    } else {
        handleIncorrectGuess(value > animal.weight);
    }
}

```

```

}

function setupInputHandlers() {
    submitButton.addEventListener("click", handleSubmit);
    giveUpButton.addEventListener("click", () => {
        localStorage.removeItem(STORAGE_KEYS.state);
        window.location.href = "menu.html";
    });

    window.addEventListener("keydown", (event) => {
        if (!overlay.classList.contains("hidden")) {
            return;
        }
        if (/^\d/.test(event.key)) {
            if (document.activeElement !== weightInput) {
                event.preventDefault();
                weightInput.focus();
                weightInput.value += event.key;
            }
            return;
        }
        if (event.key === "Enter" && document.activeElement === weightInput)
        {
            event.preventDefault();
            console.log("setupInputHandlers_handleSubmit");
            handleSubmit();
            } else if (event.key === "Enter" &&
!overlay.classList.contains("hidden")) {
            const primaryButton = modalActions.querySelector("button");
            if (primaryButton) {
                console.log("setupInputHandlers_primaryButton");
                primaryButton.click();
            }
        }
    });
}

function initGame() {
    updateHeader();
    swapCard(currentAnimal());
    setupInputHandlers();
    startTimer();
}

initGame();

```

Листинг A.6 – Файл leaderboard.html (страница отображения таблицы лидеров).

```

<!DOCTYPE html>
<html lang="ru">
    <head>
```

```

        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
        <title>Угадай вес – Рейтинг</title>
        <link rel="stylesheet" href="styles.css" />
</head>
<body>
    <header>
        <div>
            <strong>Угадай вес</strong>
            <div class="badge">ТОП-20</div>
        </div>
        <nav>
            <a href="menu.html">Главное меню</a>
            <a href="../index.html">На главную</a>
        </nav>
    </header>
    <main>
        <section class="panel">
            <h1>Таблица лидеров</h1>
            <table class="leaderboard-table">
                <thead>
                    <tr>
                        <th>#</th>
                        <th>Игрок</th>
                        <th>Очки</th>
                        <th>Сложность</th>
                        <th>Таймер</th>
                    </tr>
                </thead>
                <tbody id="leaderboard-body"></tbody>
            </table>
        </section>
    </main>
    <script type="module" src="leaderboard.js"></script>
</body>
</html>

```

Листинг A.7 – Файл leaderboard.js (скрипт заполнения таблицы лидеров из хранилища).

```

import { getLeaderboard, initLeaderboard } from "./data.js";

const tableBody = document.querySelector("#leaderboard-body");

initLeaderboard();

const entries = getLeaderboard();

tableBody.innerHTML = "";
entries.forEach((entry, index) => {
    const row = document.createElement("tr");

```

```

row.innerHTML =
    <td>${index + 1}</td>
    <td>${entry.name}</td>
    <td>${entry.score}</td>
    <td>${entry.difficulty}</td>
    <td>${entry.timed ? "Да" : "Нет"}</td>
`;
tableBody.appendChild(row);
});

```

Листинг A.8 – Файл styles.css (таблицы стилей для оформления всех страниц приложения).

```

:root {
    color-scheme: dark;
    font-family: "Segoe UI", Roboto, Helvetica, Arial, sans-serif;
    background: #0f1218;
    color: #f4f4f6;
}

* {
    box-sizing: border-box;
}

body {
    margin: 0;
    min-height: 100vh;
    background: radial-gradient(circle at top, #1f2a44, #0f1218 55%);
    display: flex;
    flex-direction: column;
}

header {
    padding: 18px 8vw;
    background: rgba(15, 18, 24, 0.9);
    display: flex;
    justify-content: space-between;
    align-items: center;
    border-bottom: 1px solid rgba(255, 255, 255, 0.08);

    nav {
        display: flex;
        gap: 12px;
        flex-wrap: wrap;
    }
}

a {
    color: #f5d742;
    text-decoration: none;
    padding: 6px 12px;
    border-radius: 8px;
    background: rgba(255, 255, 255, 0.08);
}

```

```
        transition: background 0.2s ease;

        &:hover,
        &:focus-visible {
            background: rgba(255, 255, 255, 0.18);
        }
    }

main {
    flex: 1;
    display: flex;
    flex-direction: column;
    gap: 24px;
    padding: 32px 8vw 60px;
}

.panel {
    background: rgba(18, 22, 32, 0.92);
    border-radius: 18px;
    padding: 24px;
    box-shadow: 0 18px 40px rgba(0, 0, 0, 0.35);
}

.menu-grid {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(220px, 1fr));
    gap: 16px;
}

.difficulty-card {
    border: 1px solid transparent;
    border-radius: 16px;
    padding: 16px;
    background: rgba(255, 255, 255, 0.08);
    color: inherit;
    text-align: left;
    cursor: pointer;
    transition: transform 0.2s ease, border 0.2s ease, background 0.2s ease;

    &:hover {
        transform: translateY(-3px);
        background: rgba(255, 255, 255, 0.14);
    }

    &.is-selected {
        border-color: #f5d742;
        background: rgba(245, 215, 66, 0.2);
    }
}
```

```
.menu-actions {
  display: flex;
  flex-wrap: wrap;
  gap: 16px;
  align-items: center;
}

.game-actions {
  margin-top: 16px;
}

button {
  &.primary {
    background: #f5d742;
    border: none;
    padding: 12px 20px;
    border-radius: 999px;
    font-weight: 700;
    cursor: pointer;
    color: #1a1e26;
  }

  &.secondary {
    background: transparent;
    border: 1px solid rgba(255, 255, 255, 0.3);
    padding: 12px 20px;
    border-radius: 999px;
    color: inherit;
    cursor: pointer;
  }
}

label {
  display: block;
  font-weight: 600;
  margin-bottom: 8px;

  &.switch {
    display: inline-flex;
    align-items: center;
    gap: 12px;
    margin-bottom: 0;
    cursor: pointer;
  }
}

.toggle-row {
  margin-top: 16px;
  display: flex;
  align-items: center;
  gap: 12px;
```

```
}

.switch {
  input {
    position: absolute;
    opacity: 0;
    pointer-events: none;

    &:checked + .switch-track {
      background: rgba(245, 215, 66, 0.8);

      &::after {
        transform: translateX(24px);
      }
    }

    &:focus-visible + .switch-track {
      outline: 2px solid #f5d742;
      outline-offset: 3px;
    }
  }
}

.switch-track {
  width: 52px;
  height: 28px;
  border-radius: 999px;
  background: rgba(255, 255, 255, 0.25);
  position: relative;
  transition: background 0.2s ease;

  &::after {
    content: "";
    position: absolute;
    width: 22px;
    height: 22px;
    border-radius: 50%;
    background: #f4f4f6;
    top: 3px;
    left: 3px;
    transition: transform 0.2s ease;
  }
}

.switch-label {
  font-weight: 600;
}

input[type="text"],
input[type="number"] {
  width: 100%;
```

```
padding: 12px 14px;
border-radius: 10px;
border: 1px solid rgba(255, 255, 255, 0.2);
background: rgba(15, 18, 24, 0.6);
color: inherit;
}

.game-header {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(180px, 1fr));
  gap: 16px;
  align-items: center;
}

.stat {
  background: rgba(255, 255, 255, 0.08);
  border-radius: 12px;
  padding: 12px 14px;
}

.card-stage {
  position: relative;
  overflow: visible;
  min-height: 320px;
  display: flex;
  align-items: center;
  justify-content: center;
}

.animal-card {
  width: min(520px, 90%);
  background: rgb(40, 46, 68);
  border-radius: 20px;
  padding: 24px;
  text-align: center;
  display: flex;
  flex-direction: column;
  gap: 16px;
  position: absolute;
  transition: transform 0.5s ease, opacity 0.5s ease;
}

&.enter {
  transform: translateX(120%);
  opacity: 0;

  &.enter-active {
    transform: translateX(0);
    opacity: 1;
  }
}
```

```
&.exit {
    transform: translateX(-140%);
    opacity: 0;
}
}

.animal-placeholder {
    min-height: 160px;
    border-radius: 16px;
    display: flex;
    align-items: center;
    justify-content: center;
    font-weight: 700;
    color: rgba(0, 0, 0, 0.6);
    text-transform: uppercase;
    letter-spacing: 0.08em;

    &.is-hidden {
        display: none;
    }
}

.animal-media {
    width: 100%;
    display: flex;
    flex-direction: column;
    gap: 8px;
}

.animal-image {
    width: 100%;
    height: auto;
    display: block;
    border-radius: 16px;
    object-fit: contain;
}

.hint {
    min-height: 24px;
    color: #ffd166;
}

.attempts {
    display: flex;
    gap: 6px;
}

.attempt-dot {
    width: 14px;
    height: 14px;
    border-radius: 50%;
```

```
background: rgba(255, 255, 255, 0.2);

&.is-remaining {
  background: #4cc9f0;
}

.overlay {
  position: fixed;
  inset: 0;
  background: rgba(8, 10, 14, 0.7);
  display: flex;
  align-items: center;
  justify-content: center;
  padding: 20px;
  z-index: 10;

  &.hidden {
    display: none;
  }
}

.modal {
  background: #141a26;
  border-radius: 18px;
  padding: 24px;
  max-width: 560px;
  width: 100%;
  display: flex;
  flex-direction: column;
  gap: 16px;
}

.modal-actions {
  display: flex;
  flex-wrap: wrap;
  gap: 12px;
}

.leaderboard-table {
  width: 100%;
  border-collapse: collapse;

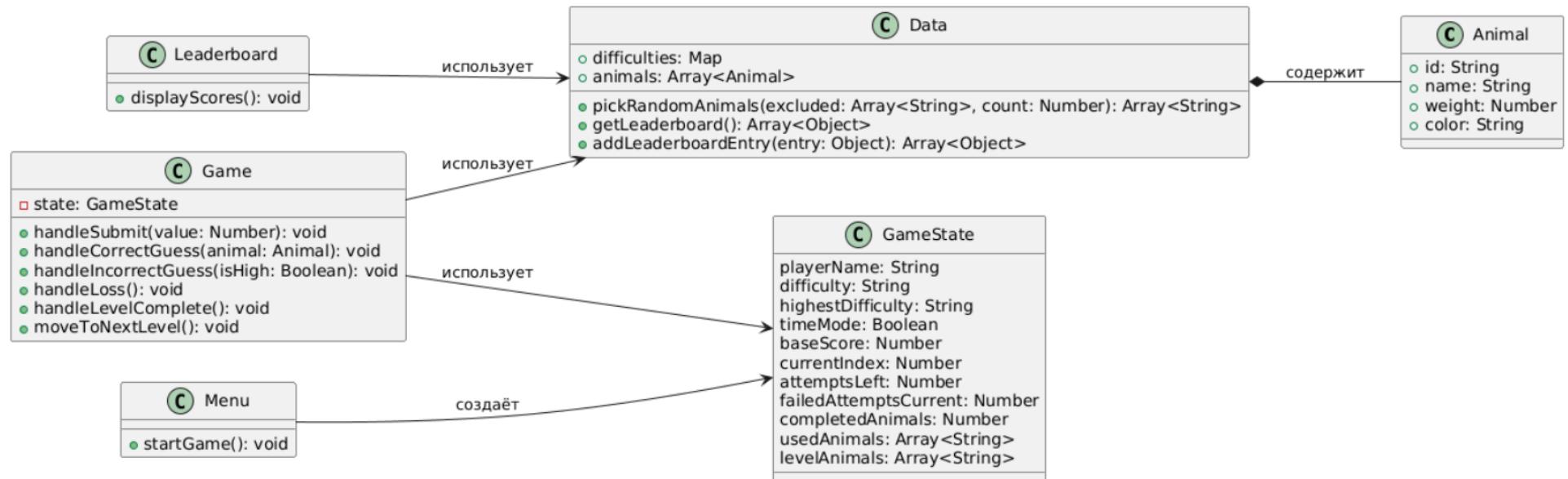
  th,
  td {
    padding: 10px 12px;
    border-bottom: 1px solid rgba(255, 255, 255, 0.1);
    text-align: left;
  }
}
```

```
.badge {
  display: inline-flex;
  align-items: center;
  gap: 6px;
  padding: 4px 10px;
  border-radius: 999px;
  font-size: 0.85rem;
  background: rgba(255, 255, 255, 0.12);
}

.error {
  color: #ff7b7b;
  min-height: 20px;
}

@media (max-width: 720px) {
  header {
    flex-direction: column;
    gap: 12px;
  }
}
```

Приложение Б. UML-диаграмма приложения



Приложение В. Блок схема игрового процесса

