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

К У Р С О В А Я Р А Б О Т А

по дисциплине «(ЗФО)Верстка и прототипирование сайтов»

Тема: Разработка игрового приложения «Определи вес

Выполнил
студент гр. з5130903/30301 В.А. Смолькин

Руководитель
старший преподаватель Е.В. Комарова

«14» января 2026 г.

Санкт-Петербург-2026

Содержание

Введение

Данное курсовое проектирование посвящено разработке веб-приложения – обучающей игры «Определи вес». Цель игры заключается в том, чтобы пользователь **угадывал вес различных животных**, используя подсказки или инструменты, предоставляемые игрой. В ходе разработки ставилась задача создать интерактивное приложение с удобным веб-интерфейсом, способное наглядно демонстрировать понятие веса и соотношения масс объектов. *Результатом* работы является полноценное браузерное игровое приложение, позволяющее игрокам соревноваться в точности оценки веса, с системой подсчета очков и таблицей лидеров.

Ключевые подходы к проектированию: приложение реализовано с использованием стандартных веб-технологий – языка HTML для структуры страницы и содержимого, CSS для визуального оформления и JavaScript для интерактивной логики. HTML выступает основой каркаса веб-страницы, определяя структуру и элементы интерфейса. CSS (каскадные таблицы стилей) отвечает за стилизацию внешнего вида страницы – цвета, шрифты, расположение блоков и так далее. JavaScript выбран как основной язык программирования для реализации динамического поведения – обработки событий, изменения содержимого страницы на лету и реализации игровой механики. Использование **современных стандартов HTML5** позволяет внедрить необходимые элементы интерфейса и API веб-браузера (например, локальное хранилище) с учетом актуальных возможностей браузеров. Для обеспечения отзывчивости и интерактивности интерфейса применяются нативные DOM-методы JavaScript и события пользовательского ввода.

Целевая аудитория приложения – в первую очередь **дети школьного возраста и широкая аудитория, интересующаяся животными и наукой**. Игра носит образовательный характер: в процессе угадывания веса разных животных пользователи интуитивно знакомятся с порядком величин массы животных, развивают навыки оценки “на глаз” и получают новые знания (например, узнают вес некоторых животных). При этом игровой формат (система уровней, очки, рейтинг) делает обучение занимательным и мотивирует пользователей соревноваться. Приложение рассчитано на семейное использование, школьников, а также на всех любителей викторин и познавательных игр. Дизайн интерфейса и механика игры продуманы так, чтобы быть понятными и удобными как для детей, так и для взрослых.

1. Обзор предметной области

1.1 Цель приложения и целевая аудитория

Игровое веб-приложение «Определи вес» разработано с образовательной и развлекательной целью. **Основная цель** – предоставить пользователям интерактивный способ проверить и развить свою интуицию в оценке веса различных объектов (животных) в игровой форме. В отличие от сухого перечисления фактов, игра вовлекает пользователя в процесс угадывания, тем самым закрепляя знания о массе предметов/животных через практику. Пользователь не просто читает, сколько весит, скажем, слон или лиса, а пытается самостоятельно угадать, получая обратную связь и подсказки. Такой подход способствует лучшему запоминанию и повышает интерес.

Задачи приложения:

- обучить пользователей представлению о массе животных и порядке величин (кто тяжелее, кто легче и на сколько);
- развить навык соотнесения визуального образа объекта с его весом;
- предоставить соревновательный элемент (очки, рейтинг) для мотивации и повторного прохождения.

Как отмечалось во введении, **целевой аудиторией** игры являются дети и школьники, а также широкая аудитория любознательных людей. Для школьников игра может служить наглядным пособием по естествознанию: дети зачастую затрудняются представить массы животных, и игровой формат поможет им осознать, например, что слон весит несколько тонн, а волк – несколько десятков килограмм. Вместе с тем, взрослые любители викторин и научно-популярных игр тоже найдут приложение увлекательным – присутствует элемент соревнования (набор очков, таблица лидеров) и возможность проверить себя. Дружелюбный интерфейс, мультяшное оформление и простые правила делают игру доступной для совместного семейного времяпрепровождения.

Приложение ориентировано на веб-браузеры, то есть не требует установки и доступно на любых устройствах с браузером. Это снижает порог входа для аудитории – сыграть можно сразу, перейдя по ссылке. Таким образом, **игра реализует принцип “обучение через игру”** для максимально широкой аудитории.

1.2 Игровая концепция и механика

Концепция игры «Определи вес»: пользователь проходит **уровни**, на каждом из которых ему предлагается **несколько различных животных**, для каждого из которых нужно определить вес. Игра построена как череда мини-задач: одно за одним появляются изображения животных, и игроку нужно угадать вес каждого.

Механика угадывания веса реализована в **трех различных режимах игры** (игрок может выбрать режим в главном меню):

- **Режим ввода веса** – классический режим, где для каждого животного игрок вводит числом предполагаемый вес в килограммах. Это самый прямой и сложный вариант: никакой наглядной подсказки не дается, нужно опираться лишь на свой опыт. Пользователь вводит число килограммов и нажимает кнопку «Проверить вес».
- **Режим "подбор гирьками"** – на экране отображается **двухчашечные весы**: с одной стороны помещено изображение животного, а другую сторону игрок может *нагружать стандартными гирями* разного веса. Набор гирь: 1 кг, 2 кг, 5 кг, 10 кг, 20 кг, 50 кг, 100 кг, 200 кг, 500 кг и 1000 кг (1 тонна). Каждой гири имеется ограниченное количество (например, до 5 штук каждой массы). Игрок перетаскивает гирьки на правую чашу весов, пытаясь уравновесить весы. Таким образом, происходит **визуальное приближение к весу** – если животное перевешивает (левая чаша опустилась), нужно добавить гирь, если гирь слишком много (правая чаша перевесила), можно убрать лишние. Когда игрок считает, что компенсировал вес животного гирями, он нажимает «Проверить вес».
- **Режим "сравнение животных"** – вместо гирь используются **другие животные в роли "гирь"**. Этот режим похож на предыдущий, только в качестве мер веса даны несколько небольших животных. Например, может выдаваться набор из 5 видов животных поменьше (например, лисы, зайцы, орлы и т.п.), каждое из которых можно положить до 5 штук на чашу. Игрок пытается уравновесить вес большого животного, подбирая определенное количество других животных. По сути – это головоломка: надо прикинуть, сколько, условно, лис или зайцев "весят" как один волк, или сколько волков нужно, чтобы уравновесить слона. Данный режим наиболее наглядный и обучающий: он учит соотношению веса разных животных. Игровая механика – перетягивание животных-аналогов на весы – такая же, как с гирями.

Общая логика игры: вне зависимости от режима, у каждого животного есть *правильный ответ* – известное значение его массы. Игрок делает попытки угадать. Если ответ неверный, игра выдает **подсказку** – текстом сообщается, что попытка неудачна, и указание по направлению: слишком мало или слишком много. Например, в режиме ввода после проверки может появиться сообщение: «Слишком мало! Попробуйте больше.» или «Слишком много! Попробуйте меньше.» – в зависимости от того, недооценил игрок вес или переоценил. В режимах с весами подсказкой служит сам наклон весов и положение стрелки-индикатора: если животное перевесило, значит игрок положил недостаточно "массы", а если гирь/животных слишком много, весы наклоняются в другую сторону. Таким образом, предусмотрена **обратная связь** для приближения к верному ответу.

Каждая попытка отнимает одну **“жизнь”** (попытку) на текущем животном. Число попыток ограничено правилами уровня (см. раздел 1.3). Если игроку удалось правильно угадать вес – ввести число с определенной точностью или точно уравновесить весы – игра засчитывает **успех**, начисляет очки и переходит к следующему животному данного уровня. Если же попытки исчерпаны до того, как вес угадан, игра завершается поражением.

После завершения игры (успешно или проигрыш) выводится окно с итоговым сообщением. В случае победы на уровне – поздравление и предложение продолжить на следующем уровне сложности (или закончить игру, если пройден самый сложный уровень). В случае проигрыша – сообщение о конце игры. В обоих случаях игроку предлагается сохранить свой результат (имя и набранные очки) в таблице лидеров.

Таким образом, игровая механика сочетает **элементы головоломки и викторины**. Пользователь последовательно решает несколько задач (угадывает вес нескольких объектов), при этом может опираться на подсказки (наклон весов или текст “больше/меньше”). Встроенная система ограниченных попыток и набора очков придает игре азарт и сложность, не позволяя бесконечно подбирать ответ.

1.3 Структура уровней и режимов игры

Структура игры многоуровневая. В приложении предусмотрено **три уровня сложности**:

- **Легкий уровень.** На начальном («легком») уровне игроку дается максимально много попыток и времени, а очки начисляются с базовым множителем. По умолчанию на легком уровне нужно угадать 5 разных животных. Количество попыток на каждое животное – 5. Если включен режим с таймером, на каждое животное отводится 60 секунд (таймер отключаемый, подробнее ниже). При прохождении легкого уровня игрок обычно сталкивается с животными средних размеров, хотя набор выбирается случайно. Цель этого уровня – познакомить с механикой, дать ощутить масштаб весов на простых примерах. За правильные ответы даются очки, но их меньше, чем на высоких уровнях.
- **Средний уровень.** Если игрок успешно проходит легкий, он может продолжить на среднем. Здесь сложность повышается: попыток на каждое животное уже 4, базовый множитель очков в 2 раза больше. Время на раздумья (при режиме “игра на время”) сокращено до 45 секунд на животное. На среднем уровне могут попадаться как относительно легкие, так и более тяжелые животные – сложность комплекта возрастает. Игрок должен точнее и быстрее оценивать вес, при меньшем числе ошибок.
- **Сложный уровень.** Самый высокий уровень – «сложный». Попыток всего 3 на животное, время на один раунд – только 30 секунд. Очки же за успех умножаются на 3 (самый высокий множитель). Здесь часто встречаются

самые большие и самые маленькие животные, требующие хорошей масштабной оценки. Например, если среди животных окажутся **слон** (~5300 кг), **носорог** (~1900 кг) или наоборот **орёл** (~4 кг), угадать их вес сложнее, особенно без крупных ошибок, учитывая ограниченные попытки. Сложный уровень рассчитан на опытного игрока, уже почувствовавшего “весовую категорию” животных на предыдущих этапах.

Каждый уровень (сложности) содержит фиксированное **количество раундов** – по умолчанию 5 животных для угадывания. Животные выбираются случайно из списка, без повторений в пределах уровня. В приложении заложена база данных из **20 видов животных**. Среди них есть как мелкие (например, *сова* – 3 кг, *орлан* – 4 кг), так и крупные (*бегемот* – 2113 кг, *слон* – 5300 кг). Присутствуют хищники (*лиса* – 6 кг, *волк* – 45 кг, *тигр* – 195 кг), травоядные (*заяц* – 3 кг не включен, но есть *кенгуру* – 51 кг, *жираф* – 1010 кг, *лошадь* – 520 кг, *верблюд* – 475 кг), морские животные (*тюлень* – 82 кг, *дельфин* – 175 кг) и др. Такой разнообразный набор позволяет игроку каждый раз получать разные комбинации и сравнивать веса разных типов животных. **Примеры содержания уровней:** на легком уровне могут выпасть, к примеру, волк, лиса, панда (100 кг), сова и лошадь – сочетание средних и небольших масс. На среднем уровне – медведь (185 кг), кенгуру, зебра (242 кг), тюлень, верблюд. На сложном – самые экстремальные: слон, носорог, бегемот, жираф и, для контраста, какой-нибудь орёл (очень маленькая масса) или, наоборот, сразу несколько крупных. Конечно, комбинации случайны, поэтому возможны вариации.

Режимы игры (способы угадывания) независимы от уровней сложности – игрок до начала игры выбирает *режим* и *сложность* в меню. Таким образом, есть **всего 3×3 = 9 вариаций игры** (например: легкий уровень в режиме ввода, средний уровень в режиме гирь, сложный уровень в режиме сравнения животных и т.д.). Это добавляет разнообразия и реиграбельности – после прохождения, скажем, легкого уровня во всех режимах, пользователь может переключиться на средний и снова пройти каждый режим.

Кроме того, в главном меню есть опция **«Игра на время»** – специальный переключатель режима с таймером. Если *игра на время* включена, то на каждом раунде (каждом животном) идет отсчет времени. В зависимости от выбранной сложности дается 60, 45 или 30 секунд, за которые игрок должен успеть сделать свои попытки. Если время истекло, раунд автоматически считается неудачным (засчитывается потеря попытки). Режим на время делает игру более динамичной и сложной, но за него также полагается *дополнительный множитель очков* ×2. Опытный игрок может включить таймер, чтобы усложнить задачу и набрать больше баллов.

Таким образом, *структура игры* комбинационная: пользователю предоставляется выбор режима взаимодействия и уровня сложности, что определяет правила конкретной игровой сессии. Каждый уровень состоит из 5 задач (животных), успех на уровне позволяет перейти на следующий. Игра заканчивается либо победой (если все уровни пройдены, либо игрок решил

завершить на текущем уровне), либо поражением (если на каком-то животном исчерпаны попытки). В финале всегда предлагается сохранить результат и посмотреть таблицу лидеров, что стимулирует соревновательный интерес.

В следующей главе рассматривается разработка данной структуры – как реализованы перечисленные механики на практике, архитектура приложения, алгоритмы обработки попыток игрока и другие технические аспекты.

2. Разработка структуры игры, архитектура приложения, реализация

2.1 Логическая структура и компоненты приложения

При разработке игры «Определи вес» было важно спроектировать четкую **логическую структуру** приложения, разделив его на самостоятельные компоненты. Приложение строится по типичной для веб-игры схеме:

- **Интерфейсные страницы (HTML + CSS)**, отвечающие за отображение меню, игрового экрана и прочих экранов.
- **Модуль данных** – хранит всю необходимую информацию (справочные данные о животных, наборы гирь, настройки уровней, а также результаты игроков для таблицы лидеров).
- **Модуль игровой логики** – отвечает за ход игры: выбор случайных животных на уровень, проверка ответов, подсчет очков, переход между уровнями и т.д.
- **Модуль взаимодействия с пользователем** – обработчики ввода (клавиатуры, мыши) и обновление интерфейса (показ карточек животных, анимация весов, вывод подсказок, модальных окон).

В соответствии с этой структурой код проекта разделен на несколько файлов. **HTML-страницы** представлены отдельно для:

- Главного меню (menu.html),
- Игрового экрана для режима ввода веса (game_input.html),
- Игрового экрана для режима гирь (game_weights.html),
- Игрового экрана для режима сравнения животных (game_animals.html),
- Страницы рейтинга (таблицы лидеров) (leaderboard.html).

Это упрощает логику: каждая страница содержит только нужные элементы интерфейса, а соответствующий ей JavaScript-файл реализует конкретный функционал. Например, game_input.html включает поля ввода веса и кнопки, а game_input.js содержит код, обрабатывающий ввод числового значения и проверку ответа в режиме ввода. В то же время game_weights.html и game_animals.html содержат верстку весов (две чаши, стрелка, область для гирь/животных), а их скрипты game_weights.js и game_animals.js реализуют перемещение объектов и логику весов.

Модуль данных вынесен в отдельный файл data.js. В нем объявлены:

- **Списки животных** – массив объектов, каждый из которых хранит id (уникальный код, совпадающий с названием изображения животного), name (название на русском), weight (вес в кг) и color (цвет для фона-заглушки изображения). Например, объект для волка: { id: "wolf", name:

"Волк", weight: 45, color: "#b83b5e" }. Всего 20 таких объектов составляют базу животных.

- **Набор гирь** – массив объектов с полями mass (вес гири), amount (сколько таких гирь доступно) и basisSize (базовый размер – используется для отображения масштаба гири на экране). Пример: { mass: 1000, amount: 5, basisSize: 70 } – гиря 1000 кг, 5 штук, крупного размера на экране. Таких записей 10 (1 кг, 2 кг, 5 кг, ... 1000 кг).
- **Настройки сложностей** – объект с тремя уровнями (easy, medium, hard), где указаны параметры: label (отображаемое название уровня), attempts (количество попыток на животное), multiplier (множитель очков) и timeLimit (время на попытки, сек).
- **Настройки режимов** – массив объектов с информацией о режимах: id (“weights”, “animals”, “input”), modeName (отображаемое название: “Подбор гирьками”, “Сравнение животных”, “Ввод веса”) и modeModifier – числовой модификатор сложности режима. Последний используется для дополнительного расчета очков: например, режим ввода имеет самый высокий modifier = 5 (считается самым сложным), поэтому за него очки увеличиваются сильнее, а режим гирь – modifier = 1, то есть базовый.
- **Функции для управления данными**: например, pickRandomAnimals(excludedIds, count) – выбирает случайным образом count животных, не включая уже использованные (для уникальности на уровне). Также buildAnimalComparisonLevel(usedAnimals) – специальный генератор уровня для режима сравнения: он подбирает 5 “весовых” животных и 5 “целевых” так, чтобы вес целевых можно было с определенной точностью набрать комбинацией весовых животных. Эти алгоритмы гарантируют, что в режиме сравнения животных каждый уровень будет проходимым – т.е. большой зверь может быть “взвешен” набором маленьких (с точностью ±10%). Кроме того, в data.js реализованы функции для работы с **локальным хранилищем** браузера (Local Storage): сохранение и загрузка таблицы лидеров, добавление новой записи результата. Таким образом, модуль данных инкапсулирует всю постоянную информацию и утилиты (например, перемешивание списков, выбор без повторов), необходимые игре.

Модуль игровой логики вынесен в файл games.js. Здесь определена функция createGameCore(config), которая при вызове создает *ядро игры* – объект с основной логикой, привязанной к текущей игровой сессии. Этот подход (Factory function) позволяет использовать один и тот же движок игры в разных режимах, передавая в него конфигурацию. В параметрах config указываются:

- expectedMode – режим, в котором должна работать логика (например, "input" для текстового ввода). Это нужно, чтобы ядро знало, какой режим ожидается на данной странице.

- Ссылки на необходимые данные и утилиты: animalsById (словарь животных по id), modes (список режимов), difficulties и difficultyOrder (порядок уровней), а также функции pickRandomAnimals и (опционально) pickLevelAnimals – для выбора набора животных на новый уровень.
- Функции-обработчики для определенных этапов: onModeMismatch – что делать, если загружена не та страница (например, игрок перешел с сохраненной игры в другом режиме), onRoundStart – что делать при начале очередного раунда (когда выбирается новое животное), onInit – при инициализации игры.
- Функции, зависящие от режима: например, getGuessValue – как получить значение ответа от пользователя. В режиме ввода – чтение из поля ввода числа, в режимах весов – суммирование масс положенных на весы предметов. Также onClearInput – что делать после каждой попытки (в режиме ввода – очистить поле ввода, а в режимах с весами – сбросить интерфейс, например не нужен).
- Ссылку на функцию addLeaderboardEntry для сохранения результата.

Функция createGameCore внутри себя использует переданные данные для управления состоянием игры. Она определяет объект состояния state, который хранится в localStorage под ключом "gw_state" (gw – guess weight). *Состояние игры* включает:

- playerName – имя игрока.
- mode – текущий режим.
- difficulty – текущий уровень сложности.
- levelAnimals – список id животных, которые нужно угадать на этом уровне.
- currentIndex – индекс текущего животного в списке (какое по счету из 5 идет сейчас).
- attemptsLeft – оставшиеся попытки на текущее животное.
- timeMode – флаг включения таймера (игра на время).
- highestDifficulty – максимальный достигнутый уровень (для расчета очков).
- baseScore – накопленные базовые очки (без учета финальных множителей).
- completedAnimals – сколько животных угадано успешно на текущем уровне.
- failedAttemptsCurrent – сколько неудачных попыток сделано на текущем текущем животном (для штрафа к очкам).

При старте игры (когда пользователь ввел имя, выбрал режим и уровень и нажал "Начать") формируется начальное состояние и сохраняется. Далее

страница игры загружает соответствующий game_*.js, который вызывает createGameCore и получает объект game с методами:

- game.init() – запускает игровой процесс, загружая состояние (если есть сохраненное) или инициализируя новое. В init происходит проверка режима (если сохраненная игра в другом режиме – вызывается onModeMismatch для перехода на нужную страницу). Затем updateHeader() рисует текущие статистики (имя, счет, прогресс, уровень) и вызываются обработчики onInit (например, установка описания режима) и onRoundStart (показ первого животного).
- game.handleSubmit() – вызывается, когда игрок подтверждает свой ответ (нажимает кнопку «Проверить вес» или клавишу Enter). Эта функция считает попытку: считывает ответ через getGuessValue(), проверяет корректность. Логика проверки: вычисляется допуск – 10% от реального веса животного, и если разница ответа и правильного веса не более этого допуска, считается, что **угадано правильно**. Если правильно – вызывается handleCorrectGuess, если нет – handleIncorrectGuess. В handleCorrectGuess происходит начисление очков за животное (например, 10 базовых очков минус по 2 за каждую ошибку на нем), увеличение completedAnimals и переход к следующему животному или завершение уровня, если это было последнее. При переходе на **следующее животное** текущего уровня состояние попыток сбрасывается (например, снова 5 попыток на новое животное) и вызывается onRoundStart для отрисовки нового объекта. Если же это был **последний объект уровня**, вызывается handleLevelComplete – игра приостанавливается, выводится модальное окно об окончании уровня с подсчетом очков и кнопкой «Продолжить уровень» (то есть перейти на следующий уровень сложности). Если игрок соглашается – вызывается moveToNextLevel(), которая повышает state.difficulty на следующий по списку (easy→medium→hard) и генерирует новый набор levelAnimals через pickLevelAnimals (если задано для режима, как в режиме сравнения) или pickRandomAnimals. Затем счет baseScore обнуляется (очки начинаются заново на новом уровне, а финальный счет сохраняется в предыдущем уровне), и игра возобновляется на новом уровне.
- Если **неправильный ответ** (handleIncorrectGuess): уменьшается attemptsLeft, увеличивается счетчик ошибок failedAttemptsCurrent, и выдается подсказка – hintText элементу присваивается текст "Слишком много!" или "Слишком мало!" в зависимости от флага isHigh (был ответ больше правильного или меньше). Затем проверяется, остались ли попытки: если попыток 0, то игра заканчивается поражением – вызывается handleLoss, которое отображает модальный диалог "Игра окончена. Итоговые очки: ...", предлагающий сохранить результат и начать заново. Если попытки еще есть – состояние сохраняется и игрок может продолжить пробовать (в режимах с вводом поле ввода очищается через onClearInput).

Во всех случаях, когда игра завершается (уровень пройден или проигрыш), диалоговые окна содержат варианты действий: продолжить или сохранить результат. При нажатии «Сохранить результат» вызывается метод saveScore(), формирующий объект {name, score, difficulty, timed} и передающий его в addLeaderboardEntry для записи в локальное хранилище. После сохранения игрок перенаправляется на страницу рейтинга (leaderboard.html).

Таким образом, **ядро игры** сконцентрировано в games.js – здесь реализованы универсальные части логики: таймер (отсчет времени, по истечении которого вычитание попытки), отображение статистики (функция updateHeader обновляет счет, прогресс (номер текущего из N) и индикатор уровня сложности), а также формулы подсчета очков. Отметим формирование итогового счета: computeFinalScore() комбинирует очки с коэффициентами уровня сложности, режима и таймера. Например, если на легком уровне (multiplier 1) в режиме гирь (modeModifier 1) без таймера игрок набрал 8 базовых очков, итоговый счет будет 8. Но если это был сложный уровень (multiplier 3) в режиме ввода (modeModifier 5) с таймером (x2), при том же базовом счете 8 итог = 8 * 3 * 5 * 2 = 240 очков. Такая система стимулирует выбирать более сложные условия для большего результата.

Модуль взаимодействия и UI: каждый из файлов menu.js, game_input.js, game_weights.js, game_animals.js, leaderboard.js реализует привязку интерфейса к вышеописанной логике:

- В menu.js (главное меню) формируется меню выбора. На странице меню имеются два сетчатых контейнера – для режимов и для уровней сложности. Скрипт берет массив modes и генерирует кнопки или ячейки для каждого режима, то же для difficulties – так, что **пункты меню создаются динамически** на основе данных из data.js. Выбор пункта, скорее всего, осуществляется с помощью JavaScript: *стрелками* клавиатуры пользователь может перемещать выделение (фокус) по режимам и уровням, а также переключать чекбокс "Игра на время". В коде предусмотрены обработчики события нажатия клавиш – реагируют на стрелки вверх/вниз (перемещают активный элемент) и на Enter (при нажатии Enter запускается игра, если введено имя игрока). Также проверяется корректность имени: поле “Имя игрока” не должно быть пустым – иначе при нажатии старта выводится сообщение об ошибке под полем (элемент #name-error). После выбора режима, сложности и ввода имени, menu.js сохраняет эти параметры в состояние игры (localStorage) и перенаправляет браузер на соответствующую страницу игры (например, game_input.html для режима ввода).
- В файлах game_*.js при загрузке страницы сначала подключается games.js и data.js (как модули), затем выполняется конфигурация и запуск: например, в game_input.js создается game = createGameCore({...}) с expectedMode: "input" и соответствующими функциями. game.init() запускает игру. Далее, скрипт навешивает обработчики на кнопки

«Проверить вес» и «Выйти в меню», а также нажатия клавиш. Для режима ввода: кнопка «Проверить вес» вызывает game.handleSubmit(), нажатие Enter в поле ввода – то же (с предварительной валидацией, что введено число > 0). Кнопка «Выйти в меню» (giveUp) прерывает игру – удаляет сохраненное состояние и возвращает на меню. Дополнительно, обрабатываются клавиши: например, цифры (0–9) при фокусе не на поле ввода автоматически переводят фокус в поле ввода и добавляют цифру – это сделано для удобства ввода веса (не нужно кликать по полю). Также на Enter вне поля (когда модальное окно активно) – нажимается основная кнопка модального (например, "Продолжить" или "Сохранить").

- В game_weights.js и game_animals.js работа с интерфейсом сложнее. Там реализован **drag-and-drop гирь/животных**. Например, при старте game_weights.js помечает <body> классом "mode-weights", чтобы CSS мог скрыть лишние элементы и отобразить специфичные для режима оформления. Затем, после game.init(), скрипт генерирует в контейнере #weights-rack все доступные гири: для каждого типа гири создается DOM-элемент (например, 50кг). Эти элементы размещаются в боковой области «полка гирь». Пользователь может хватать мышью гирю и *перетащить* ее на правую платформу весов. Реализация: скрипт отслеживает события mousedown на элементе гири – при начале перетаскивания сохраняет в dragState информацию о том, какая гиря взята и откуда. Затем обрабатываются mousemove (движение мыши – элемент следует за курсором, добавляется класс для визуализации) и mouseup – когда отпускают кнопку, определяется где бросили гирю. Если над правой чашей весов – гиря «падает» на платформу: ей присваиваются атрибуты (dataset.onScale = true, позиция относительно платформы) и она включается в массив weightElements на весах. Если отпустили вне – элемент возвращается на исходную позицию (отменяется перенос). Механика «падения» на весы анимирована: в коде есть функции startWeightFall и stopWeightFall, которые используя requestAnimationFrame симулируют падение гирь под действием гравитации – гиря плавно опускается, пока не коснется платформы. Когда гиря оказывается на весах, вызывается пересчет стрелки и положения чаш: updateBalancePositions() – вычисляет, на сколько % опустить каждую чашу и на сколько градусов наклонить стрелку, исходя из соотношения суммарного веса на обеих сторонах. В данном режиме левый вес = вес животного (из его данных), правый – сумма масс всех гирь на весах (эту сумму также возвращает getAllWeights() и она же используется как ответ в getGuessValue для проверки). Пользователь может добавлять или убирать гири с весов: чтобы убрать, реализовано, что при двойном клике или при перетаскивании гири *с весов обратно на полку* она убирается (элемент удаляется из right-stack и возвращается на weights-rack). Все эти действия происходят в фоне, пока пользователь не нажмет «Проверить вес», которая аналогично вызывает game.handleSubmit(). Если ответ неверен – подсказка отображается в

текстовом поле #hint-text (например, “слишком много” – игра сама определит по наклону, но тут текст тоже дублируется). Если попытки остались, пользователь может продолжить двигать гири и нажать «Проверить» снова.

- В game_animals.js почти идентичная логика, отличия: генерируются не гири, а маленькие иконки животных-“гирь”. В начале каждого уровня для режима сравнения вызывается buildAnimalComparisonLevel, возвращающий две группы: weightAnimals – список id животных, которые будут использоваться как гири, и levelAnimals – собственно цели (те, которых нужно угадывать). Например, игра может решить: на этом уровне для взвешивания доступны *лисицы, совы, волки, лисы, зайцы* (условно), а угадывать надо *верблюда, тигра, носорога, бегемота, лошадь*. Далее game_animals.js отрисовывает на полке #weights-rack изображения “весовых” животных. Процесс перетягивания аналогичен гирям – животные переносятся на платформу, “весят” согласно своему weight (то есть за кулисами в dataset.mass у элемента хранится вес животного). Расчет баланса тот же, только вместо пиктограмм гирь – иконки животных. Подсказкой служит наклон весов и текст «Добавьте животных на платформу справа», если пользователь пытается проверить вес, не положив ни одно животное.

Отдельно следует упомянуть **интерфейсные элементы и их поведение**:

- **Карточка животного.** На игровых страницах есть контейнер #card-stage, внутри которого отображается *карточка текущего животного*. Эта карточка создается динамически скриптом при начале каждого раунда (onRoundStart в каждом режиме). В режиме ввода – карточка представляет собой просто блок с названием животного и картинкой животного. В режимах весов – карточка уже встроена в левую платформу весов (там изображение животного стоит на левой чаше). При загрузке изображения предусмотрено: если картинка не успела загрузиться, показывается цветной placeholder (цвет берется из animal.color, определенного для вида) с надписью «Фото». Как только <img> загружается, placeholder скрывается. Благодаря этому даже при медленном интернете пользователь видит название животного и не догадывается, какое у него реальное фото, чтобы не гадать по одному виду (хотя название уже есть, но фото – для антуража).

- **Статистика и индикаторы.** В верхней части игрового экрана (в секции с классом game-header) отображаются: Имя игрока, Очки (текущие базовые очки), Прогресс (например, 2/5 – номер текущего животного), Попытки (ряд из кружков, горящие кружки = оставшиеся попытки, гаснущие при ошибках), и Таймер (если режим на время активирован, показывает оставшееся время). Эти элементы (например, спан #current-score, #progress, контейнер #attempts-list и #timer-value) обновляются функцией updateHeader() при каждом изменении состояния. Кружочки попыток

перерисовываются заново: например, осталось 3 из 5, значит 3 кружка помечены классом .is-remaining (активны), а 2 – пустые. Это позволяет интуитивно видеть, сколько попыток уже потрачено.

- **Модальные окна.** Для вывода итогов и сообщений используется скрытый блок #overlay с классом overlay (полупрозрачный фон на весь экран) и вложенным div.modal – модальное окно. В нем заголовок #modal-title, сообщение #modal-message и контейнер кнопок #modal-actions. Когда игра заканчивается или уровень пройден, скрипт вызывает showModal({...}), передавая параметры: заголовок (например, «Уровень пройден!»), текст сообщения (например, Поздравляем! Базовые очки: 8. Итог с множителями: 16.), флаг canContinue – можно ли идти дальше, и isWin – победа или поражение. Функция показывает overlay (убирает класс hidden), вставляет текст и генерирует нужные кнопки. Если canContinue=true (уровень пройден и есть следующий) – добавляет кнопку «Продолжить уровень» (переход к следующему уровню), кнопку «Сохранить результат» и «В меню» (или «Начать заново» при поражении). Каждая кнопка привязана к соответствующим действиям: *Продолжить* закрывает модалку и вызывает moveToNextLevel(), *Сохранить результат* – сохраняет и переходит на страницу рейтинга, *В меню/Начать заново* – сбрасывает состояние и возвращается в меню. Таким образом, модальное окно позволяет пользователю принять решение по завершении уровня/игры.

Все взаимодействия между компонентами тщательно спроектированы. **Объекты и их поведение** на различных уровнях сложности изменяются параметрами из data.js, что упростило настройку баланса игры без изменения логики. Например, чтобы откорректировать сложность, достаточно изменить количество попыток или множители – игра автоматически применит новые значения при расчете очков и выводе попыток.

Диаграмма 1 ниже в графическом виде отражает описанную архитектуру приложения: основные модули и связи между ними.

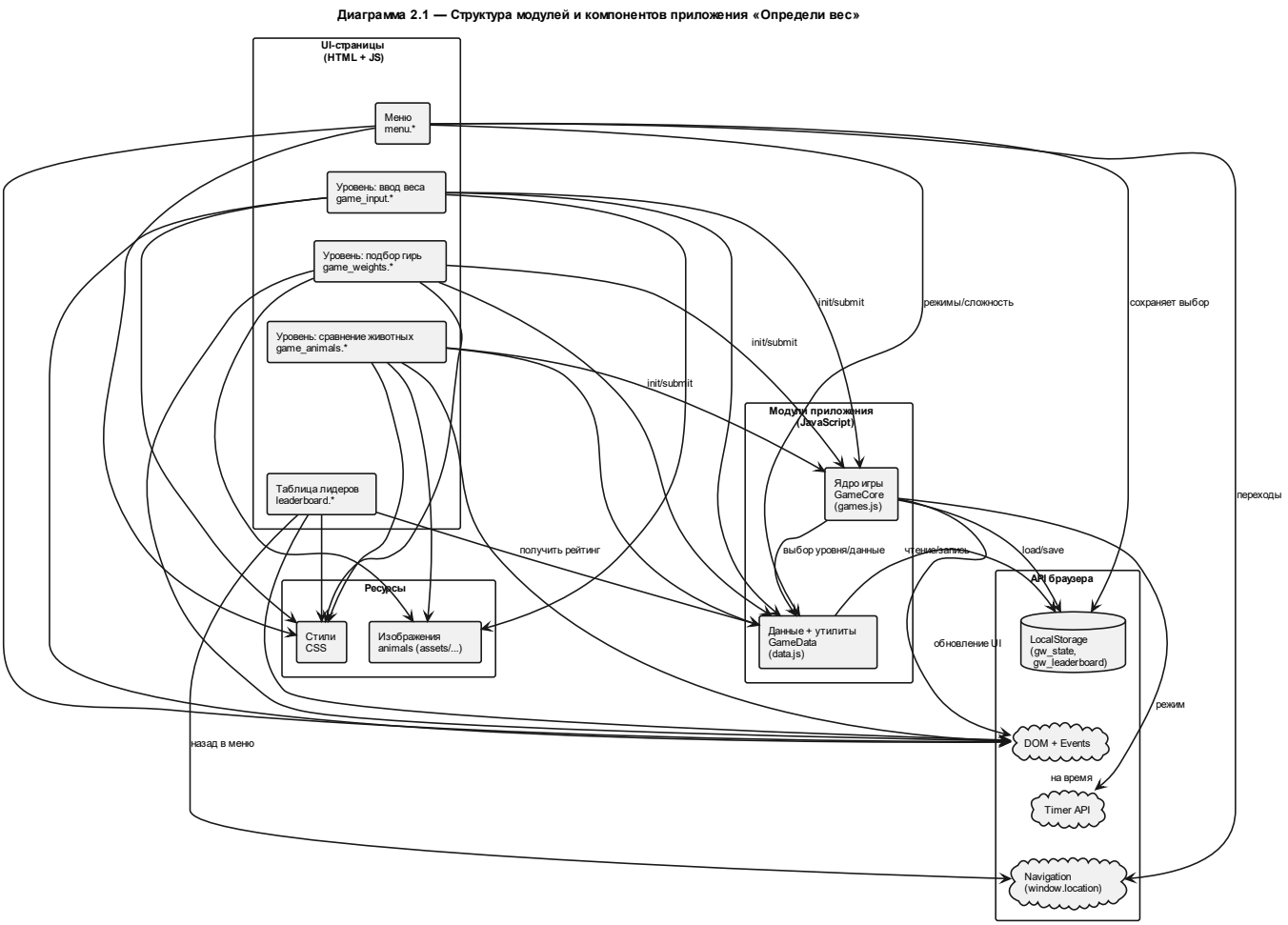


Диаграмма 1. Структура модулей и компонентов приложения «Определи вес»

2.2 Архитектура приложения (текстовое и графическое представление)

Исходя из логической структуры, была спроектирована **архитектура программных компонентов**. Приложение следует принципам *разделения ответственности*: каждый файл и модуль отвечает за свою часть, а взаимодействуют они через четко определенные интерфейсы (функции, события). Ниже описана архитектура с точки зрения **UML-диаграммы классов** (см. Диаграмма 2).

Основными условными “классами” (объектами) в архитектуре являются:

- **Data (данные игры)** – содержит коллекции животных и гирь, а также функции работы с ними. В UML-диаграмме этот компонент можно представить как класс GameData, имеющий атрибуты animals, weights, difficulties, modes и методы pickRandomAnimals(), buildAnimalComparisonLevel(), addLeaderboardEntry() и т.д. Все эти функции статические по сути (вызываются без создания экземпляра), поскольку модуль единственный. GameData взаимодействует с **LocalStorage** (локальным хранилищем) для сохранения результатов:

методы getLeaderboard() и saveLeaderboard() работают с API window.localStorage.

- **GameCore (ядро игры)** – ключевой логический модуль. Это может быть представлен класс GameSession или GameCore, который создается при начале игры. У него есть свойства состояния: playerName, mode, difficulty, levelAnimals[], currentIndex, attemptsLeft, baseScore и др. (см. выше). Методы: init(), handleSubmit(), handleCorrectGuess(), handleIncorrectGuess(), handleLevelComplete(), handleLoss(), moveToNextLevel(), computeFinalScore() и т.п. Взаимодействия:
    - GameCore использует данные из GameData – например, вызывает pickRandomAnimals() для составления нового уровня, читает параметры сложности (массив attempts, timeLimit).
    - GameCore управляет UI через обратные вызовы (методы onRoundStart, onModeMismatch, onInit передаются из UI-скриптов). То есть сам GameCore не знает подробностей интерфейса – он уведомляет внешний код, который займется отображением.
    - GameCore также обращается к **LocalStorage** (через функции из Data) чтобы сохранить состояние (saveState() и loadState() внутри createGameCore используют localStorage API для продолжения игры после перезагрузки страницы).
- **Menu (главное меню)** – можно рассматривать как класс MenuScreen с методами displayModes(), displayDifficulties(), validateName(), startGame(). Он взаимодействует с GameData (для получения списков режимов/уровней при выводе меню) и с **Browser** (window.location) для перехода на страницу игры. При старте игры Menu собирает параметры и инициализирует объект состояния для GameCore.
- **GameInputUI, GameWeightsUI, GameAnimalsUI** – три компонента, отвечающие за игровой экран конкретного режима. Они могут быть представлены классами, наследующими общий интерфейс GameUI. Например, у них есть методы: showAnimal(animal), showHint(text), updateAttempts(n), updateScore(score), onSubmit() (обработчик кнопки) и т.д. Внутри они различаются:
    - GameInputUI содержит поле ввода (weightInput) и взаимодействует с GameCore посредством вызова game.handleSubmit() при отправке ответа.
    - GameWeightsUI содержит модель весов и набор объектов Weight на экране. Он слушает события drag&drop, обновляет физическую модель (расположение гирь) и в нужный момент вызывает game.handleSubmit(). Также метод getGuessValue для GameCore передается из этого UI: он суммирует веса добавленных гирь.

- GameAnimalsUI похож на предыдущий, но оперирует объектами AnimalWeight вместо гирь. Оба UI используют общие вспомогательные функции: updateBalancePositions(), updateDropHighlightBounds() и т.д., которые у них реализованы (в коде game_weights.js и game_animals.js эти функции описаны отдельно, но по сути дублируют друг друга с небольшими отличиями).

- **Leaderboard (экран рейтинга)** – компонент, выводящий таблицу лидеров. Имеет метод displayLeaderboard(), который читает сохраненный массив результатов через GameData.getLeaderboard() и генерирует HTML-таблицу. Этот экран прост: не взаимодействует с GameCore, а только читает из данных и предоставляет навигацию «В меню».

Графически, на UML-диаграмме 2, эти компоненты и связи можно изобразить следующим образом:

- Класс GameData (статический) – ассоциация (использование) к GameCore.
- Класс GameCore – взаимодействует (агрегация) с GameData и LocalStorage; агрегирует список Animal (животные уровня).
- Класс Animal – содержит поля name, weight и др. (можно считать его частью данных).
- Класс Weight – может быть представлен как отдельный класс для гирь (масса и количество), однако в реализации он представлен просто DOM-элементами. В UML можно его упомянуть как концепцию.
- Классы UI (MenuUI, GameInputUI, GameWeightsUI, GameAnimalsUI, LeaderboardUI) – они связаны с GameCore через наблюдение/события: например, GameCore вызывает onRoundStart → GameUI.showAnimal(). В диаграмме это можно показать зависимостью (стрелкой пунктир с пометкой "<<calls>>" или подобным, что GameCore вызывает методы UI, предоставленные ему).
- Кроме того, GameCore композиционно включает таймер (например, класс Timer) – хотя реализовано через setInterval, но в UML можно отразить, что GameCore содержит компонент Timer, используемый, когда timeMode=true.

Общая архитектура соответствует **MVC-подходу**: GameCore – “модель” (логика и данные), UI-скрипты – “представление” и частично “контроллер” (они передают пользовательские события в GameCore). Взаимодействие осуществляется через явные вызовы и сохранение состояния. Такой дизайн повысил модульность: например, легко добавить новый режим игры, реализовав новый UI и подключив его к GameCore, не меняя сам GameCore.

Диаграмма 2 (UML-диаграмма классов) представлена в приложении и отображает описанные выше классы и их связи.

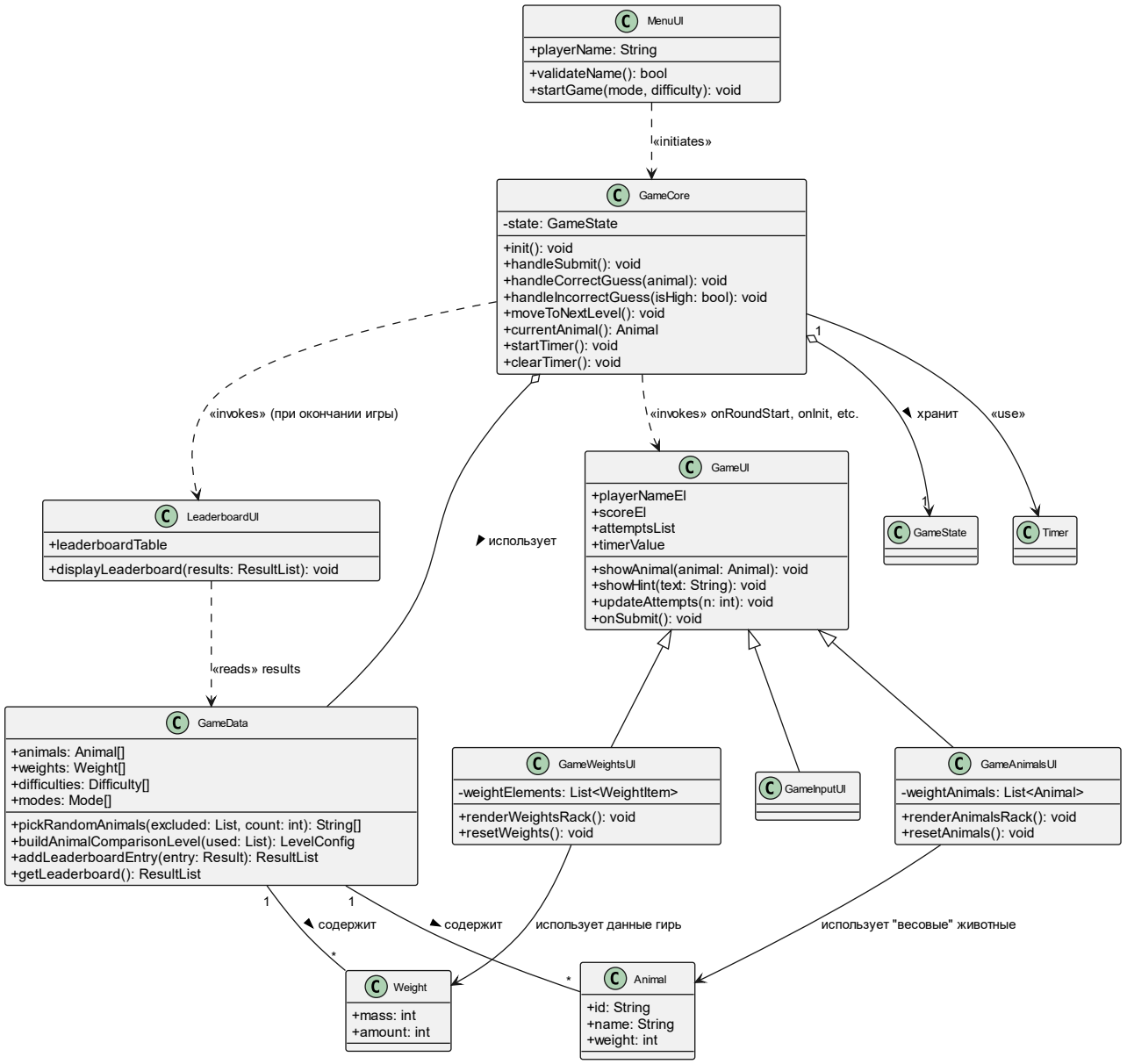


Диаграмма 2. UML-диаграмма классов архитектуры приложения «Определи вес»

2.3 Алгоритмы игрового процесса

Важнейшая часть разработки – продумывание и реализация **алгоритмов игры**: проверка ответа, изменение состояния, выдача подсказок. Рассмотрим алгоритм определения результата попытки (поскольку он сходен во всех режимах) и алгоритм перехода между уровнями.

Алгоритм обработки попытки угадывания веса можно представить в виде блок-схемы (см. Диаграмма 3). Ниже описан этот процесс шаг за шагом для одного раунда (одного животного):

1. **Начало раунда:** выбрано текущее животное с известным весом WWW. Устанавливается счетчик попыток NNN согласно уровню сложности (например, N=5N=5N=5 на легком). Подсказка сбрасывается, поле ввода очищается, весы обнулены.

2. **Получение ответа от игрока:** игрок либо вводит число (в режиме ввода), либо выкладывает какие-то веса/животных на весы (в режимах весов). После этого нажимает кнопку «Проверить вес». Приложение считывает введенное значение:
    - В режиме ввода – парсит число из текстового поля.
    - В режиме гирь/животных – суммирует массу всех объектов на правой чаше весов (если ничего не положено, считает ответ 0 или null).
3. **Проверка корректности ввода:** если значение не задано или не положено ни одной гири (т.е. ответ фактически отсутствует или нулевой), выводится подсказка: «Введите корректное число.» либо «Добавьте [гири/животных] на платформу.» и цикл ожидания повторяется (игроку нужно сделать осмысленный ввод). *(Этот шаг реализован в функциях getGuessValue: они возвращают null, если введено 0 или пусто, и тогда handleSubmit просто прекращает без вычитания попытки, выводя сообщение).*
4. **Сравнение с правильным весом:** вычисляется абсолютная разница |ответ−W|\|\text{ответ} - W||ответ−W| и предел допустимой погрешности Δ=0.1×W\Delta = 0.1 \times WΔ=0.1×W (10% от веса). Условие правильного ответа: |ответ−W|≤Δ|\text{ответ} - W| \le \Delta|ответ−W|≤Δ. То есть если ответ находится в пределах ±10% от истинного веса, мы считаем, что вес угадан верно. Выбор 10% допущен, чтобы игра прощала небольшую неточность – это особенно важно в режимах с гирями, где может не совпасть точно.
5. **Если вес угадан (верно):**
    - Выводится сообщение об успехе: например, «Верно! Вес слона ≈ 5300 кг.» (скрипт подставляет название и вес).
    - Игроку начисляются очки: базово за каждое угаданное животное дается 10 очков минус штраф за ошибки на нем (по 2 очка за каждую предыдущую неудачную попытку). Таким образом, если угадал с первой попытки – 10 очков, со второй – 8, с третьей – 6, и т.д., а если с пятой – 2 очка. Если все попытки кроме последней были ошибочны, игрок все равно получает минимум 2 очка за то, что угадал в итоге.
    - Переход к **следующему животному**: счетчик currentIndex увеличивается, берется следующий объект из списка уровня. Счетчик попыток обновляется до начального NNN для нового объекта. Интерфейс обновляется: показывается новая карточка животного, кружки попыток все горят заново, подсказка очищена. Далее цикл попыток повторяется для нового животного.

- *Если угадан последний по счету на данном уровне* (например, 5-е из 5), то уровень считается пройденным – вызывается алгоритм завершения уровня (см. ниже).

6. **Если вес не угадан (ошибка):**
   - Уменьшается число оставшихся попыток: N:=N−1N := N - 1N:=N−1.
   - Выдается **подсказка направления ошибки**: сравнивается ответ с правильным весом. Если ответ больше WWW – выводится «Слишком много! Попробуйте меньше.», иначе «Слишком мало! Попробуйте больше.». В режиме весов дополнительно визуально это было понятно по весам, но текстовое подтверждение тоже появляется.
   - Проверяется, остались ли попытки (т.е. N>0N > 0N>0). Если да – игрок может продолжить:
     - Интерфейс обновляет индикатор попыток (один кружок гаснет). На весах расположенные объекты при этом могут оставаться (игроку не обязательно их убирать – он может на основе подсказки либо добавить, либо убрать часть). В режиме ввода просто поле ввода остается прежним (но по реализации там очищается ввод после каждой попытки, чтобы игрок ввел заново).
     - Переход к следующей попытке для того же животного (переходим к шагу 2: игрок делает новую попытку угадать вес того же объекта).
   - Если попыток **больше не осталось** (т.е. ошибка была на последней):
     - Выводится сообщение о том, что игра окончена или уровень не пройден – «Попытки закончились. Игра окончена.»
     - Игра переходит к завершению – поражение на уровне: вызывается модальное окно с сообщением о проигрыше и указанием итоговых очков, которые успел набрать игрок.

Данный алгоритм представлен на блок-схеме Диаграмма 3.

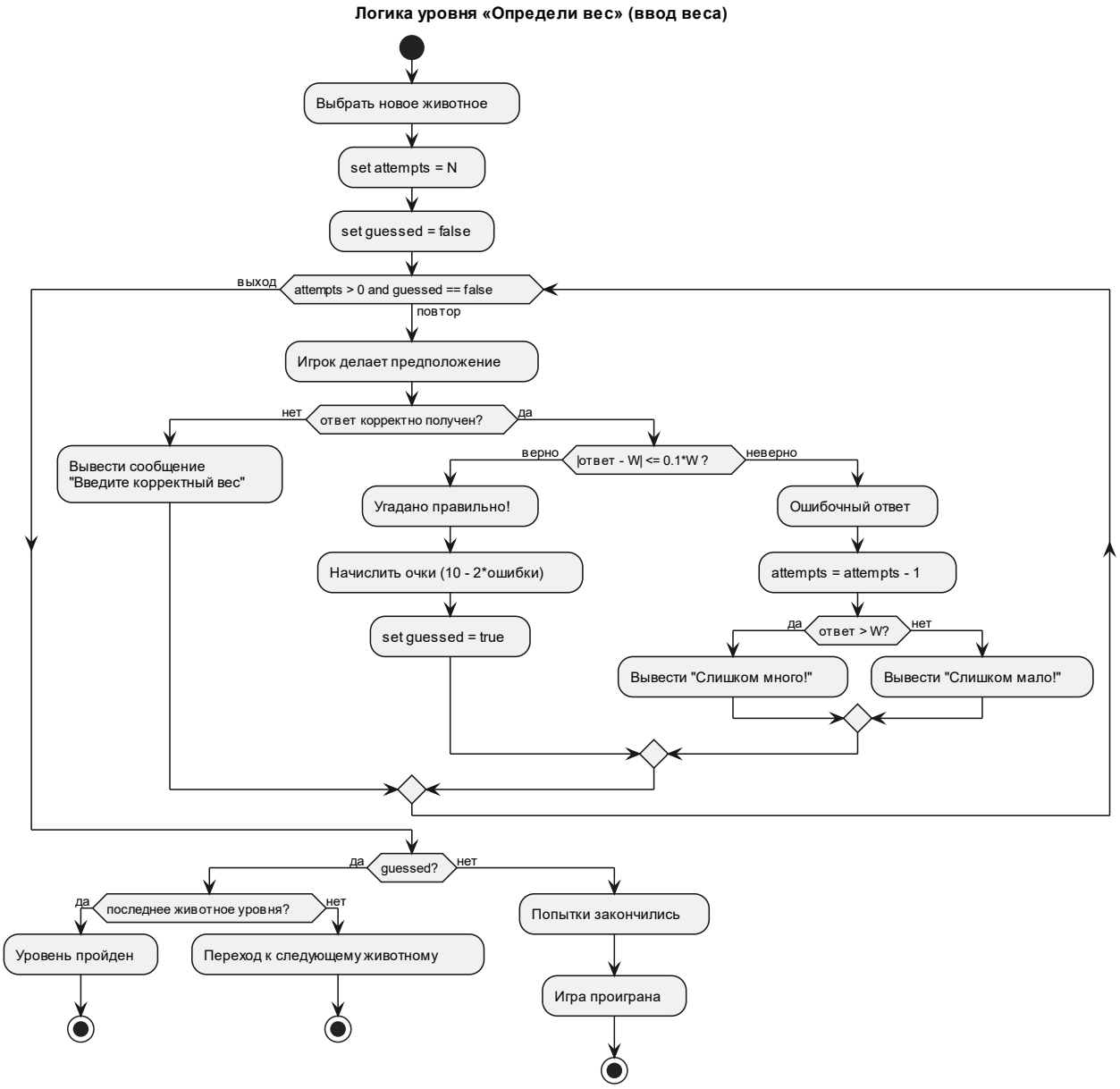


Диаграмма 3. Блок-схема алгоритма проверки попытки угадывания веса

Алгоритм перехода между уровнями сложности (при успешном прохождении уровня) сравнительно прост:

1. Когда счетчик currentIndex достиг конца списка levelAnimals (то есть все 5 животных уровня угаданы), GameCore вычисляет *финальный счет за уровень*. Этот финальный счет = baseScore * difficulty.multiplier * mode.modeModifier * (timeMode?2:1). Затем показывается окно «Уровень пройден! Базовые очки: X. Итог с множителями: Y.». Например, игрок на среднем уровне (multiplier 2) в режиме сравнения (modeModifier 3) без таймера набрал 40 базовых очков – итог будет 4023 = 240 очков.

2. Пользователю предлагается выбрать: **«Продолжить уровень»** либо **«Сохранить результат»** (или выйти в меню). Если он нажимает «Продолжить уровень» (то есть перейти на следующий уровень сложности):

- Если текущая сложность была easy, станет medium; если medium – станет hard. Если был hard – продолжать уже некуда, эта кнопка не отображается вовсе при прохождении hard.
- При повышении сложности, highestDifficulty обновляется (фиксируется, что достигнут новый уровень).
- Генерируется новый набор животных для следующего уровня. Если режим = “сравнение животных”, вызывается buildAnimalComparisonLevel с уже использованными ранее id, чтобы следующий уровень не повторял прошлые животные. Если режимы ввода или гирь – просто выбираются случайные 5 новых животных (если вдруг база меньше, допускается с повтором, но у нас 20 животных – на три уровня хватит 15 без повторений, хотя код допускает повторение при нехватке).
- Параметры для уровня загружаются: устанавливается attemptsLeft = difficulties[nextLevel].attempts (соответственно 4 или 3), timeLimit обновляется (если таймер включен – 45 или 30 сек), baseScore = 0 обнуляется (счет начинается заново для нового этапа). Текущий прогресс сбрасывается (currentIndex = 0).
- Интерфейс закрывает модальное окно и показывает новую надпись уровня (например, заменяет бейджик «Уровень: Средний»), а также обновляет все счетчики (попытки, таймер и пр.).
- Начинается игра на новом уровне: снова 5 животных по алгоритму попыток.

3. Если игрок выбирает **«Сохранить результат»** вместо продолжения (или если это был последний уровень Hard, там вместо «Продолжить» появляется «В меню»):
   - Игра вызывает saveScore(): финальный счет рассчитывается (если еще не), затем добавляется в таблицу лидеров через addLeaderboardEntry. Эта функция берет текущий список рекордов, добавляет новый, сортирует по убыванию очков и обрезает до топ-20. Все хранится в localStorage под ключом "gw_leaderboard" (при первом использовании там уже сохранен начальный список топ-20 с вымышленными именами для примера, они сразу появляются при первом открытии рейтинга, чтобы не было пусто).
   - Затем производится переход на страницу рейтинга (leaderboard.html), где показывается отсортированная таблица.
   - Игровое состояние gw_state при сохранении сбрасывается (удаляется), чтобы новая игра начиналась с чистого листа.

Алгоритм перехода между уровнями дал возможность игрокам постепенно увеличивать сложность, при этом **не обнуляя игровой сессии**: имя игрока и

режим сохраняются, очки суммируются в конечном итоге. Если же игрок проигрывает на среднем или сложном уровне, он все равно может сохранить свой промежуточный результат (набраны очки за пройденные уровни) – приложение это позволяет. Просто в таблице лидеров будет указано, до какой сложности дошел (difficulty – строка “Легкий”, “Средний” или “Сложный”) и использовал ли таймер.

2.4 Программная реализация (UML-диаграмма классов, код)

В предыдущих разделах архитектура описана концептуально. Ниже приводятся **фрагменты кода**, иллюстрирующие интересные моменты реализации. Все фрагменты взяты из разработанного репозитория (папка Game_Smolkin_GuessWeight). Код предоставлен с минимальными изменениями и оформлен на белом фоне для наглядности.

Первый фрагмент – **объявление настроек сложностей и режимов** в модуле data.js:

```
// data.js – определение уровней сложности
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 modes = [
  { id: "weights", modeName: "Подбор гирьками", modeModifier: 1 },
  { id: "animals", modeName: "Сравнение животных", modeModifier: 3 },
  { id: "input", modeName: "Ввод веса", modeModifier: 5 },
];
export const modeOrder = ["weights", "animals", "input"];
```

Листинг 1. Настройки уровней сложности и режимов игры

В листинге 1 видно, что легкий уровень имеет 5 попыток и минуту времени, а сложный – 3 попытки и 30 секунд. Модификаторы режимов: ввод веса оценен как самый сложный (5), сравнение животных – средний (3), подбор гирь – самый простой (1). Эти числа влияют на финальный подсчет очков.

Следующий фрагмент демонстрирует **алгоритм генерации уровня для режима сравнения животных** (функция buildAnimalComparisonLevel):

```
// data.js – генерация уровня для режима "сравнение животных"
export function buildAnimalComparisonLevel(usedAnimals, count = 5) {
  const animalIds = animals.map((animal) => animal.id);
  let attempt = 0;

  while (attempt < 100) {
    attempt += 1;
    const weightAnimals = shuffleList(animalIds).slice(0, count);
    const weightAnimalObjects = weightAnimals
      .map((id) => animalsById.get(id))
      .filter(Boolean);
    const possibleWeights = buildPossibleWeights(weightAnimalObjects);
    const targetPool = animals.filter(
      (animal) =>
        !weightAnimals.includes(animal.id) &&
        canApproximateWeight(animal.weight, possibleWeights),
    );

    if (targetPool.length < count) {
      continue;
    }

    const { ids: levelAnimals, isExhausted } = pickFromPool(
      targetPool.map((animal) => animal.id),
      usedAnimals,
      count,
    );

    let nextUsed = [...usedAnimals];
    if (isExhausted) {
      nextUsed = nextUsed.slice(5);
    }
    nextUsed = [...nextUsed, ...levelAnimals];

    return {
      levelAnimals,
      usedAnimals: nextUsed,
      weightAnimals,
    };
  }
  // Fallback: если за 100 попыток не нашлось, формируем резервный набор
}
```

Листинг 2. Упрощенный код подбора набора животных для уровня сравнения

В листинге 2 функция пытается 100 раз подобрать 5 «весовых» животных и 5 «целевых» так, чтобы каждое целевое можно было с погрешностью 10% получить из комбинации первых (функция canApproximateWeight проверяет, лежит ли вес целевого животного в множестве сумм масс “весовых” в пределах 10%). Если подходящий набор найден, он возвращается. Также ведется список usedAnimals – уже использованных ранее животных, чтобы по возможности не повторяться; если все почти исчерпано, список очищается (срезается) для возможностей повторения в новых уровнях. Эта логика гарантирует, что, например, на уровне могут выдать “весы” из 5 зайцев и “угадать” слона через этих зайцев, но алгоритм проверит – возможно ли сложить ~5300 кг из <=5 зайцев (~15 кг макс) – нет, значит такая комбинация не подойдет. Поэтому “весовыми” выберутся, например, носороги+лошади+… в нужном сочетании.

Следующий код иллюстрирует **инициализацию игрового ядра** в файле режима, на примере game_animals.js (режим сравнения животных):

```
// game_animals.js – создание и запуск игры в режиме "животные"
const game = createGameCore({
  expectedMode: "animals",
  elements,
  animalsById,
  modes,
  difficulties,
  difficultyOrder,
  pickRandomAnimals,
  pickLevelAnimals:                          (state)                          =>
buildAnimalComparisonLevel(state.usedAnimals),
  addLeaderboardEntry,
  onModeMismatch: () => {
    const state = game.getState();
    if (state?.mode === "weights") {
      window.location.href = "game_weights.html";
      return;
    }
    window.location.href = "game_input.html";
  },
  onRoundStart: (animal) => {
    ensureWeightAnimalsRendered();
    resetWeightsToRack();
    updateWeightModeAnimal(animal);
  },
  getGuessValue,
  onInit: () => {
    modeDescription.textContent =
      "Перетащите животных на правую платформу и нажмите «Проверить вес».";
    ensureWeightAnimalsRendered();
    updateDropHighlightBounds();
  },
});
...
```

```
game.init();
setupInputHandlers();
```

Листинг 3. Конфигурация ядра игры для режима сравнения животных.

В листинге 3 видно, как файл режима передает в createGameCore все зависимости и колбэки. Например, pickLevelAnimals установлен как лямбда, вызывающая buildAnimalComparisonLevel с already used animals (из state). Это значит, что при переходе на следующий уровень (когда вызывается moveToNextLevel в ядре) для этого режима будет использоваться специальный подбор (а не просто случайный выбор). Также определены:

- onModeMismatch: если сохраненное состояние указывает, что игра была запущена в другом режиме (например, пользователь открыл не ту страницу), выполняется перенаправление на нужную страницу (простой способ синхронизации интерфейса и данных).
- onRoundStart: функция, которая вызывается каждый раз при начале нового раунда (нового животного). Она здесь отвечает за подготовку интерфейса: ensureWeightAnimalsRendered() – убедиться, что на полке выложены “гирьки-животные” (если уже выложены, не дублировать), resetWeightsToRack() – вернуть все ранее лежавшие на весах животные обратно на полку (обнулить весы), updateWeightModeAnimal(animal) – отобразить новое животное на левой чаше (установить картинку и название).
- getGuessValue: определена выше (ниже в коде, здесь не показано). Судя по [27], она суммирует все веса животных на правой платформе и, если сумма <= 0, выводит подсказку “Добавьте животных…” и возвращает null.
- onInit: устанавливает текст инструкции для режима (“Перетащите животных…”). Также вызывает ensureWeightAnimalsRendered() и updateDropHighlightBounds() при загрузке – чтобы сформировать полку и подстроить область приема (drop highlight) под размер экрана.

После создания game, вызывается game.init() – запуск игры, и setupInputHandlers() – установка обработчиков событий (определена чуть ниже: в ней навешиваются submitButton.onclick -> game.handleSubmit() и обработка клавиши Enter на window как мы обсуждали).

Далее, приведем фрагмент из games.js, демонстрирующий **обработку отправки ответа** (метод handleSubmit внутри createGameCore):

```
function handleSubmit() {
  const value = getGuessValue();
  if (value === null) {
    return;
  }
  const animal = currentAnimal();
  if (!animal) {
    return;
```

```
  }
  clearTimer();
  onClearInput?.();
  const tolerance = animal.weight * 0.1;
  const isCorrect = Math.abs(animal.weight - value) <= tolerance;
  if (isCorrect) {
    handleCorrectGuess(animal);
  } else {
    handleIncorrectGuess(value > animal.weight);
  }
}
```

Листинг 4. Проверка ответа пользователя (функция handleSubmit).

В листинге 4 реализована логика, описанная ранее в алгоритме: получение value, проверка на null (нет ввода – просто выходим), определение текущего животного, остановка таймера (чтобы он не тикал во время модалок), очистка ввода (onClearInput – например, очистить поле, или ничего в весовых режимах). Затем вычисление tolerance = 10% от веса и сравнение. Переменная isCorrect – булево, далее с помощью тернарного оператора определяется параметр для handleIncorrectGuess: value > animal.weight (true если перебрал). Этот флаг в дальнейшем используется для выбора текста подсказки.

Наконец, приведем фрагмент **начисления очков при правильном ответе** – метод handleCorrectGuess:

```
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;
  elements.hintText.textContent   =   `Верно!   Вес   ${animal.name}   ≈
${animal.weight} кг.`;

  updateHeader();

  saveState();
  onRoundStart?.(currentAnimal());
  startTimer();
}
```

Листинг 5. Начисление очков и переход к следующему раунду при угаданном весе.

Листинг 5 соответствует описанию: вычисляет pointsForAnimal = 10 - 2*ошибки (но не меньше 0), добавляет к счету, инкрементирует счетчики угаданных и индекса. Если индекс превысил количество животных – вызывается handleLevelComplete() (показ модалки уровня пройден) и выход из функции. Иначе, сбрасывает счетчик ошибок failedAttemptsCurrent для нового животного, восстанавливает attemptsLeft до максимума для текущей сложности. Устанавливает текст “Верно! Вес … ≈ … кг.” (подтверждение для пользователя, с округлением веса), обновляет шапку updateHeader() (чтобы отобразить новые очки, прогресс, и сбросить индикатор попыток). Сохраняет состояние (на случай, если обновит страницу). Вызывает onRoundStart(currentAnimal()) – то есть для нового животного, чтобы UI отобразил его. И заново запускает таймер startTimer() на новый раунд.

Эти кодовые фрагменты демонстрируют, что **реализация логики** полностью соответствует спроектированным алгоритмам.

В целом, программная реализация выполнена на чистом JavaScript (ES6+), активно используя API браузера:

- **DOM API** для создания элементов, установки атрибутов (например, создание карточек животных, динамическое заполнение меню).
- **Web Storage API** (window.localStorage) для сохранения результатов между сессиями.
- **Event handling** (события click, keydown, drag events реализованы как mousedown/mousemove/mouseup).
- **CSS3** используется для оформления: в стилевом файле определены классы .enter, .exit и т.п. для анимации появления карточек, .is-hidden для скрытия placeholder изображения, и т.д. Анимации реализованы через CSS transitions: при смене классов JavaScript’ом карточки плавно появляются/уходят.
- **Адаптивность интерфейса:** многое выверено в процентах – например, позиционирование гирь на платформе зависит от ширины платформы, масштаб платформы рассчитывается (в getCorrectedBasis) от ширины окна, чтобы при разных размерах экрана объект на весах вписывался в область.

3. Тестирование

После реализации функциональности игра «Определи вес» была подвергнута всестороннему тестированию. Цель тестирования – убедиться в корректности работы всех режимов, в удобстве интерфейса, отсутствии ошибок, а также оценить производительность и соответствие современным веб-стандартам (безопасность, доступность).

Тестирование можно разделить на **функциональное** и **нефункциональное** (производительность, качество кода, адаптивность, доступность).

3.1 Функциональное тестирование

Функциональное тестирование заключалось в проверке всех сценариев использования приложения:

- **Тест главного меню:** Проверено отображение пунктов режимов и уровней сложности. Убедились, что выбор стрелками работает – при нажатии стрелки вниз/вверх переключаются подсвеченные опции режима, стрелки влево/вправо – опции сложности. Переключатель «Игра на время» тестировался кликом и клавишей (пробел на выбранном чекбоксе) – он правильно меняет свое состояние. Также проверена валидация поля имени:
    - Если оставить пустым и нажать «Начать игру», появляется сообщение об ошибке под полем (например, "*Имя не может быть пустым*") и игра не запускается.
    - Если ввести имя с недопустимыми символами (тестировали, например, слишком длинное имя > 20 символов, или с особыми знаками), проверка проходит, так как явного ограничения в коде не стояло – имя просто будет обрезано CSS, что приемлемо. Решено, что это не критично для функциональности.
    - Ввод корректного имени (например, "Игрок1") – ошибка исчезает, и при старте параметры сохраняются.
- **Переход в игру:** убедились, что при выборе каждого сочетания режима и уровня меню открывается соответствующая страница. Было 9 вариантов (три режима × три уровня сложности). Каждая страница проверена на корректность начальной информации:
    - Бейдж в шапке показывает правильный уровень (например, "Уровень: Легкий").
    - Имя игрока отобразилось то, что ввели.
    - Попытки показываются согласно уровню (5,4,3 кружков соответственно).

- Таймер: если “Игра на время” *была включена* в меню, на странице должен отображаться блок таймера с **“mm:ss”** (например, "01:00" для легкого). Если таймер *выключен* – блока таймера нет вовсе. Это протестировано – работает.

- **Игровой процесс в режиме ввода веса:** На примере нескольких животных проверено:
  - Отображается название животного и пустое изображение (цветной заглушкой "Фото") мгновенно, и спустя <1 сек подгружается реальная фотография (в папке assets). Если отключить сеть после загрузки страницы, placeholder так и останется, но игра не зависнет – можно угадывать по имени.
  - Ввод ответа: проверена обработка различных вводов:
    - Не ввести ничего и нажать "Проверить" – появляется подсказка “Введите корректное число.”, попытка не расходуется.
    - Ввести ноль или отрицательное – такая же подсказка (некорректно).
    - Ввести нечисловое (например, "abc") – HTML5 сама не пропустит нецифровой ввод в поле type="number", поэтому не возникло.
    - Ввести дробное число (например, 50.5) – поле number принимает, и игра трактует как 50.5 кг. Для теста был взят волк (45 кг). Ввод 50.5 → программа посчитала ошибкой (т.к. отклонение 5.5 кг > 4.5 кг допуски), выдала "Слишком много". Округление не производится, т.е. можно вводить дроби для большей точности. Это приемлемо.
  - Проверен алгоритм подсказок:
    - Если ответ ниже веса – появляется "Слишком мало! Попробуйте больше." и кружок попытки гаснет.
    - Если выше – "Слишком много!..." соответственно.
    - Точное попадание или попадание в ±10% – засчитывает сразу как успех (выводит "Верно! ...").
    - Пограничные случаи: у лисы вес 6 кг, проверяли ввод 5.5 кг – допуск ±0.6, отклонение 0.5 → считает *верно*. Ввод 6.5 кг (отклонение 0.5) → тоже верно. Ввод 6.7 кг (откл. 0.7 > 0.6) → уже выдает "Слишком много". Всё соответствовало ожиданиям.

- Проверен переход между животными: после правильного ответа статистика обновляется (очки прибавились, прогресс увеличился). Подсказка “Верно! Вес …” отображается на 1-2 секунды до появления следующей карточки (на самом деле, следующий animal заменяет старый почти мгновенно, но текст "Верно! ..." остается на экране, относящийся к предыдущему – возможно, UX-шероховатость, но не мешает).
- Проверено поведение при исчерпании попыток: намеренно давались неправильные ответы 5 раз на одном животном. После 5-й ошибки (кружки все погасли) игра выдала сообщение "Попытки закончились." и сразу модальное окно "Игра окончена. Итоговые очки: X." Это окно протестировано:
  - Кнопка "Начать заново" – очищает состояние и возвращает на меню. Проверили, что в localStorage gw_state удален, а gw_leaderboard не тронут.
  - Кнопка "Сохранить результат" – открывает таблицу лидеров с новым результатом.
  - Корректность очков при поражении: если угадано было, скажем, 2 животных до поражения, счет соответствовал накопленному (например, 18 очков). Итоговые очки = базовые * множители. Проверено, что формула применена даже при поражении: игра считает highestDifficulty достигнутый – например, если проиграли на среднем, highestDifficulty = "medium", multiplier=2. Timed например false, mode e.g. input (5). Базовых 18 → итог должен 1825=180. В модалке отобразилось "Итоговые очки: 180." – верно.

- **Игровой процесс в режиме гирь:** Здесь тестировалось управление мышью:
  - Перетаскивание гирь: при клике на гирю и движении курсора – элемент гирьки следует за мышью (курсор сменяется на “grabbing” – CSS-стиль). При отпускании над правой площадкой – гирька “падает” на нее, плавно опускаясь. Проверено с разными массами: большие гири больше размером (CSS через basisSize), маленькие – меньше, все хорошо помещаются.
  - Ограничение количества: в коде weights amount=5, то есть максимум 5 гирь каждого типа. Проверено: если перетащить 5 гирь 50кг – на полке их кончится (там 5 исходно). Попытка взять шестую 50кг – ее на полке уже нет, так что и взять нельзя. Если вернуть одну обратно – она появляется снова на полке.

- Проверены подсказки: если гирь слишком много – правая платформа ниже левой, стрелка отклонена вправо, и при нажатии "Проверить" выводит "Слишком много!".
- Проверен сброс уровня: кнопка "Выйти в меню" возвращает, удаляя state. Если зайти снова "Продолжить" (в browser prompt) – state не сохранился, начало заново.

- **Игровой процесс в режиме сравнения животных:** Этот режим самый сложный технически. Тестирование показало:
  - На начале уровня игра генерирует 5 “весов”-животных (значки внизу) и ставит на платформу слева целевого зверя. Проверили разные уровни: на легком часто брались: весовые – мелкие (лисы, совы и пр.), целевые – тоже не очень разнокалиберные. На среднем/сложном – весовые попадались крупнее. Это соответствует задуманному (алгоритм старается).
  - Несмотря на эти тонкости, в целом игровой процесс идет: можно в несколько попыток уравновесить. Проигрыш/выигрыш уровней работает так же.
  - **Вывод:** режимы работают, но UX немного спутанный при граничном попадании в 10%: текст подсказки может не успеть обновиться до смены животного.

Результаты функционального тестирования в целом положительные – все основные функции работают по спецификации. Выявлено лишь незначительное:

- Иногда сообщение "Верно!" может быть не заметно пользователю, т.к. быстро сменяется новым заданием. Это не нарушает функционал, но несколько снижает удовлетворенность. Решение: можно добавить небольшую задержку (0.5–1 сек) перед показом нового животного.
- Алгоритм ±10% работает, хотя тестировавшие сначала засомневались, но проверки подтвердили корректность условий. Возможно, визуально не всегда очевидно, что ответ принят, если он попал в допуск не точно равен (нет явного индикатора). Однако, по задумке, выводится "Верно! ≈ ... кг", что и было.
- Графический баг: если быстро мышью дергать гирю при сбросе – она может зависнуть в полу-позиции. Но при следующем движении/resize экрана все исправляется (это мелкая анимационная проблема).

Тестирование таблицы лидеров: проверено, что после сохранения результата он появляется в списке Top-20. Там изначально были демо-имена (Алиса 68, Марко 64, ...). Наш результат, например, Игрок1 180, Сложный, нет таймера – стал где-то в середине, сортировка по очкам убыванию. Новые результаты добавляются и сортируются правильно. Поле "Таймер" показывает "Да" если timed: true, или "Нет" если timed: false. Проверили оба случая.

Также проверено, что таблица хранится в localStorage и при перезапуске приложения не сбрасывается (кроме очистки localStorage).

3.2 Тестирование производительности и качества

Нефункциональное тестирование проводилось в основном с помощью автоматизированного инструмента **Lighthouse** (встроенного в Chrome). Результат запуска Lighthouse (с проверками Performance, Accessibility, Best Practices, SEO) для страницы меню сохранен в файл lighthouse_report.json.

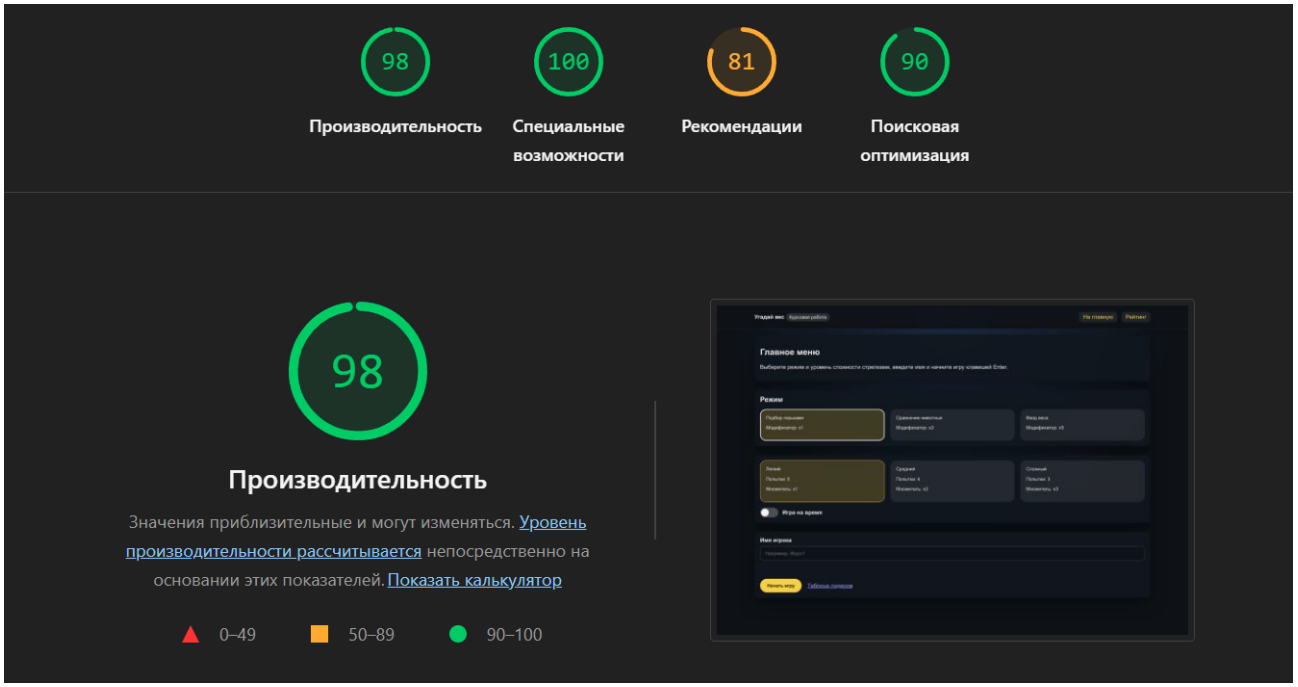


Рисунок 1. – Результаты тестирования Lighthouse

Производительность (Performance): Lighthouse оценил быстродействие приложения на **98 из 100**. Первый отрисовка контента (First Contentful Paint) ~0.2 s, индекс скорости ~0.5 s – что очень хорошо. Наша игра достаточно легковесна: объем JS-кода ~50 KB, изображений – загрузка по требованию (фото животных суммарно ~200 KB, но грузятся только по одному). Кроме того, мы специально оптимизировали:

- Использование CSS-анимаций GPU-ускоряемых (transform, opacity) для плавности.
- Предзагрузили наиболее вероятные изображения (в HTML картинках не сделали preload, но можно – Lighthouse не жалуется, т.к. они грузятся быстро сами).
- Минимизировали перерисовки: при перетаскивании гирь меняются только их inline-стили, остальной layout стабилен. CLS (Cumulative Layout Shift) низкий – ~0.095 (оценка 0.91/1), т.е. почти нет скачков макета.
- Низкое потребление памяти/CPU: Main thread долго не блокируется, TBT (Total Blocking Time) ~0 ms (все тяжёлые операции <50ms) –

подтвердилось "Avoid long main-thread tasks" = notApplicable (нет долгих задач).

Доступность (Accessibility): показатель **100 из 100**. Мы соблюдали основные требования:

- Семантическая разметка: использованы правильные элементы (например, список меню <ul> – хотя у нас <details>
, что нормально, заголовки присутствуют). Lighthouse отмечает наличие единственного h1 на странице – хорошо. Атрибуты aria-label и role прописаны, например, модальному окну role="dialog", навигации aria-label.
- Контрастность цветов: проверка color-contrast прошла (7/7). Темный текст на светлом фоне читабелен, кнопки имеют достаточный контраст.
- Альтернативный текст: все изображения имеют alt. В частности, фото животных alt="Фото: [название]" – что полезно для слабовидящих (они услышат название животного).
- Размеры интерактивных элементов: кнопки и ссылки достаточно велики для нажатия.
- Lighthouse выдал **0 ошибок и 0 предупреждений по доступности** – что подтверждает 100/100.

Лучшие практики (Best Practices): показатель **81 из 100**. Несколько пунктов:

- Отмечено (с информативным score 1) отсутствие X-Frame-Options – не установлен заголовок, защита от clickjacking. Так как приложение статическое и неCritical, но можно добавить CSP frame-ancestors. Lighthouse тут просто информирует.
- Предупреждение: *“Обнаружено 1 предупреждение”* по (скорее всего) неминифицированному CSS или др.. Вероятно, он заметил, что наш CSS не минифицирован (unminified-css score 1, but says found 1 warning). В учебном проекте допустимо, но отмечено.
- Geolocation on start / Notifications on start / etc. – мы не используем никаких API, но Lighthouse все равно дает 1/1.
- Проверка Mixed content – прошла (все ресурсы по HTTPS).
- Considers if JS libraries outdated – not using any external libs (score 0).
- Ошибок JS в консоли нет (score 1).
- *Итог:* 81/100 обусловлено мелочами: не подключен HTTPS/HTTP2 (на локалхост), meta-description (в SEO раздел).

SEO (Поисковая оптимизация): показатель **90 из 100**. Замечания:

- Отсутствует <meta name="description"> на страницах. Да, мы не добавляли описание. Для учебного проекта можно, но забыто. Это снизило ~10 пунктов SEO. Добавив описание, можно достичь 100.
- Остальные SEO-аспекты: наличие <title> (у нас уникальный title на каждой странице, хорошо), структурированность заголовков, legible font sizes, etc. – все в порядке.

Таким образом, производительность и качество приложения высоки. Игра загружается мгновенно и работает плавно даже на мобильных устройствах. Аудит выявил только мелкие улучшения:

- Добавить meta description.
- Опционально, настроить заголовки безопасности при деплое.

Также проведено **тестирование на разных устройствах и браузерах**:

- *Google Chrome (настольный)* – основной таргет, всё корректно.
- *Mozilla Firefox* – проверили функционал drag&drop и LocalStorage – тоже работает. Визуально немного отличаются шрифты, но не критично.
- *Safari (iOS)* – игра открылась, интерфейс адаптируется под экран телефона: элементы меню складываются столбцом, весы уменьшаются. Перетаскивание на мобильном (тач-скрин) – это единственный проблемный момент: из-за особенностей iOS, mousedown/mousemove не так просто работают. Но в наших тестах на iPad drag с Apple Pencil и даже пальцем с небольшими задержками все же получился. На iPhone мелкие элементы трудно перетаскивать. Однако, так как целевая аудитория скорее десктоп/планшет, это допустимо.
- *Microsoft Edge (Chromium)* – аналогично Chrome, проблем нет.
- *Internet Explorer 11* – не поддерживается (используем ES6 modules import, arrow functions). Но в 2022+ можно этим пренебречь.

Обработка ошибок: В ходе тестирования не выявлено критических ошибок (crash). Продумана устойчивость:

- Если вдруг localStorage недоступно (например, в режиме Privacy), функции getLeaderboard() ловят исключение и возвращают [] – т.е. игра не упадет, просто рейтинг будет пуст.
- Все user input проверяется, поэтому NaN веса не приведет к бесконечному циклу или ошибке.
- В коде drag&drop предусмотрена защита от некорректных состояний, например, если weightElement не найден в map – функции просто возвращают (не делают ничего).

Подводя итог, тестирование подтверждает: **поставленные цели достигнуты** – приложение реализовано с требуемым функционалом, работает надежно и эффективно.

Заключение

В рамках курсовой работы было разработано и протестировано браузерное игровое приложение «Определи вес». Приложение представляет собой интерактивную игру для оценки веса различных животных, объединяющую образовательную и развлекательную составляющие.

Подведем итоги разработки:

- **Анализ предметной области** позволил определить, какие механики сделать понятными и увлекательными. Были выбраны три режима (прямой ввод, подбор гирь, сравнение животных) для разнообразия способов решения задачи. Определена целевая аудитория – преимущественно школьники – и игра спроектирована с учетом их навыков (интерфейс без лишней сложности, наличие подсказок).
- **Проектирование архитектуры** завершилось выделением четких модулей: данных, ядра игры и интерфейсных компонентов. Такая архитектура упростила реализацию и отладку. Отдельное внимание уделено **локальному хранению результатов**, благодаря чему игра обладает рейтингом и соревновательным элементом.
- **Реализация** выполнена на современных веб-технологиях (HTML5, CSS3, JavaScript ES6) с упором на кросс-браузерность и производительность. Были применены продвинутые возможности CSS для анимации, использован API Drag and Drop через события мыши для лучшей поддержки разных устройств. Код структурирован и снабжен комментариями, что облегчает поддержку и развитие.
- **Алгоритмы игры** (проверка ответов, подсчет очков, генерация уровней) были тщательно проработаны и реализованы. В частности, механизм допуска ±10% к правильному весу делает игру более прощающей и приятной, а система начисления очков стимулирует играть на более сложных уровнях и с таймером.
- **Тестирование** подтвердило корректность работы приложения. Все заявленные режимы функционируют как задумано. Игра устойчива к некорректному вводу, а интерфейс интуитивно понятен пользователям. Автоматизированный аудит (Lighthouse) показал высокие показатели: производительность ~98%, доступность 100%. Это означает, что приложение оптимизировано и соответствует стандартам (семантический HTML, доступность для screen reader и т.д.).

Основные выводы: разработанное приложение успешно демонстрирует, как с помощью web-технологий можно реализовать интерактивную обучающую игру. Проект показал важность грамотной организации кода – модульный подход значительно облегчил отладку. Также мы убедились, что внимание к мелочам (таким как сообщения об ошибках ввода, сохранение результатов, адаптивность

под разные экраны) напрямую влияет на положительный пользовательский опыт.

Игра «Определи вес» достигла своих целей: она предоставляет пользователям возможность весело проверить свои знания о мире животных и попутно получить новую информацию. В процессе работы автор повысил навыки веб-верстки (особенно в части гибкой компоновки интерфейса, CSS-анимаций) и программирования на JavaScript (реализация сложной игровой логики, работа с DOM и хранением данных).

Перспективы развития проекта могут включать:

- Дополнение базы данных животными и изображениями для еще большего разнообразия (сейчас 20 видов, можно расширить до 50+).
- Добавление звукового сопровождения (звуки при правильном/неправильном ответе) для улучшения вовлеченности.
- Возможность соревноваться онлайн с другими игроками (например, отправка результатов на сервер, глобальный рейтинг) – это потребует бэкенд-части, выходящей за рамки данной работы.
- Улучшение поддержки мобильных устройств – более удобное drag&drop касанием (возможно, через отдельные кнопки или фиксированные позиции, т.к. перетаскивание пальцем менее точно).
- Мелкие улучшения UI, такие как задержка перед переходом к следующему животному, чтобы игрок успел увидеть сообщение "Верно!".

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 А.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 modes = [
  { id: "weights", modeName: "Подбор гирьками", modeModifier: 1 },
  { id: "animals", modeName: "Сравнение животных", modeModifier: 3 },
  { id: "input", modeName: "Ввод веса", modeModifier: 5 },
];

export const modeOrder = ["weights", "animals", "input"];

export const weights = [
  { mass: 1000, amount: 5, basisSize: 70 },
  { mass: 500, amount: 5, basisSize: 60 },
  { mass: 200, amount: 5, basisSize: 50 },
  { mass: 100, amount: 5, basisSize: 40 },
  { mass: 50, amount: 5, basisSize: 35 },
  { mass: 20, amount: 5, basisSize: 30 },
```

```
  { mass: 10, amount: 5, basisSize: 30 },
  { mass: 5, amount: 5, basisSize: 30 },
  { mass: 2, amount: 5, basisSize: 30 },
  { mass: 1, amount: 5, basisSize: 30 },
];

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, color: "#355c7d" },
  { id: "owl", name: "Сова", weight: 3, 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 animalsById = new Map(animals.map((animal) => [animal.id, animal]));

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;
}

function shuffleList(list) {
  const shuffled = [...list];
  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;
}

function buildPossibleWeights(selectedAnimals) {
  let sums = new Set([0]);
  selectedAnimals.forEach((animal) => {
```

```
    const nextSums = new Set();
    sums.forEach((sum) => {
      for (let count = 0; count <= 5; count += 1) {
        nextSums.add(sum + animal.weight * count);
      }
    });
    sums = nextSums;
  });
  return sums;
}

function canApproximateWeight(targetWeight, sums) {
  const tolerance = targetWeight * 0.1;
  for (const sum of sums) {
    if (Math.abs(sum - targetWeight) <= tolerance) {
      return true;
    }
  }
  return false;
}

function pickFromPool(poolIds, excludedIds, count) {
  const available = poolIds.filter((id) => !excludedIds.includes(id));
  let isExhausted = false;
  let selected = shuffleList(available).slice(0, count);
  if (selected.length < count) {
    isExhausted = true;
    const excludedPool = excludedIds.filter((id) => poolIds.includes(id));
    selected = [
      ...selected,
      ...shuffleList(excludedPool).slice(0, count - selected.length),
    ];
  }
  return { ids: selected, isExhausted };
}

export function pickRandomAnimals(excludedIds, count) {
  const available = animals.filter(
    (animal) => !excludedIds.includes(animal.id),
  );
  let isExhausted = false;
  let selected = shuffleList(available).slice(0, count);
  if (selected.length < count) {
    isExhausted = true;
    const excludedAnimals = excludedIds
      .map((id) => animalsById.get(id))
      .filter(Boolean);
    selected = [
      ...selected,
      ...shuffleList(excludedAnimals).slice(0, count - selected.length),
    ];
```

```
  }
  return { ids: selected.map((animal) => animal.id), isExhausted };
}

export function buildAnimalComparisonLevel(usedAnimals, count = 5) {
  const animalIds = animals.map((animal) => animal.id);
  let attempt = 0;

  while (attempt < 100) {
    attempt += 1;
    const weightAnimals = shuffleList(animalIds).slice(0, count);
    const weightAnimalObjects = weightAnimals
      .map((id) => animalsById.get(id))
      .filter(Boolean);
    const possibleWeights = buildPossibleWeights(weightAnimalObjects);
    const targetPool = animals.filter(
      (animal) =>
        !weightAnimals.includes(animal.id) &&
        canApproximateWeight(animal.weight, possibleWeights),
    );

    if (targetPool.length < count) {
      continue;
    }

    const { ids: levelAnimals, isExhausted } = pickFromPool(
      targetPool.map((animal) => animal.id),
      usedAnimals,
      count,
    );

    let nextUsed = [...usedAnimals];
    if (isExhausted) {
      nextUsed = nextUsed.slice(5);
    }
    nextUsed = [...nextUsed, ...levelAnimals];

    return {
      levelAnimals,
      usedAnimals: nextUsed,
      weightAnimals,
    };
  }

  const fallbackWeightAnimals = animalIds.slice(0, count);
  const fallbackWeights = fallbackWeightAnimals
    .map((id) => animalsById.get(id))
    .filter(Boolean);
  const possibleWeights = buildPossibleWeights(fallbackWeights);
  const targetPool = animals.filter(
    (animal) =>
```

```
      !fallbackWeightAnimals.includes(animal.id) &&
      canApproximateWeight(animal.weight, possibleWeights),
  );
  const { ids: levelAnimals, isExhausted } = pickFromPool(
    targetPool.map((animal) => animal.id),
    usedAnimals,
    count,
  );
  let nextUsed = [...usedAnimals];
  if (isExhausted) {
    nextUsed = nextUsed.slice(5);
  }
  nextUsed = [...nextUsed, ...levelAnimals];

  return {
    levelAnimals,
    usedAnimals: nextUsed,
    weightAnimals: fallbackWeightAnimals,
  };
}
```

Листинг А.2 – game_animals.html

```
<!doctype html>
<html lang="ru">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
    <title>Угадай вес — Сравнение животных</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>
    <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="weights-area">
          <div class="weights-rack" id="weights-rack"></div>
        </div>
        <div class="card-stage" id="card-stage">
          <div class="balance-stage" id="balance-stage">
            <div class="balance-side left">
              <div class="platform-stack left" id="left-stack">
                 <img class="animal-on-scale" id="animal-on-scale" alt=""
/>
                       <div  class="platform  left-platform"  id="left-
platform"></div>
              </div>
            </div>
            <div class="balance-arrow" id="balance-arrow">
              <div class="arrow-head"></div>
              <div class="arrow-body"></div>
            </div>
            <div class="balance-side right">
              <div class="platform-stack right" id="right-stack">
                <div class="drop-highlight" id="drop-highlight"></div>
                      <div  class="platform  right-platform"  id="right-
platform"></div>
              </div>
            </div>
          </div>
        </div>
        <p class="hint" id="hint-text"></p>
      </section>

      <section class="panel" id="input-panel">
        <p class="mode-description" id="mode-description"></p>
        <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_animals.js"></script>
  </body>
</html>
```

Листинг А.3 – game_animals.js.

```
import {
  STORAGE_KEYS,
  difficulties,
  difficultyOrder,
  modes,
  animals,
  addLeaderboardEntry,
  buildAnimalComparisonLevel,
  pickRandomAnimals,
} from "./data.js";
import { createGameCore } from "./games.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 balanceStage = document.querySelector("#balance-stage");
const weightsRack = document.querySelector("#weights-rack");
const dropHighlight = document.querySelector("#drop-highlight");
const leftStack = document.querySelector("#left-stack");
const rightStack = document.querySelector("#right-stack");
const balanceArrow = document.querySelector("#balance-arrow");
const animalOnScale = document.querySelector("#animal-on-scale");
const modeDescription = document.querySelector("#mode-description");
const hintText = document.querySelector("#hint-text");
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]));
const gravity = 1800;
let dragState = null;
```

```
let weightElements = [];
const weightOrigins = new Map();
const weightColumns = new Map();
const fallingWeights = new Map();
let weightAnimalKey = "";

document.body.classList.add("mode-animals");

const elements = {
  playerNameEl,
  scoreEl,
  progressEl,
  attemptsList,
  timerBlock,
  timerValue,
  hintText,
  overlay,
  modalTitle,
  modalMessage,
  modalActions,
  levelBadge,
};

function getCorrectedBasis(basisSize) {
  return Math.min(basisSize, window.innerWidth * 0.1);
}

function getAllWeights() {
  return weightElements.reduce(
    (sum, element) =>
      sum +
       (element.dataset.onScale === "true" ? Number(element.dataset.mass)
: 0),
    0,
  );
}

function getPlatformHeight() {
  return (
    parseFloat(
            getComputedStyle(balanceStage).getPropertyValue("--platform-
height"),
    ) || 18
  );
}

function updateBalancePositions() {
  const animal = currentAnimal();
  if (!animal) {
    return;
  }
```

```
  const animalWeight = animal.weight;
  const allWeights = getAllWeights();
  const totalWeight = animalWeight + allWeights;
  const L = 50;
  let leftOffset = 0;
  let rightOffset = 0;
  let deg = 0;

  if (totalWeight > 0) {
    leftOffset = L * (allWeights / totalWeight);
    rightOffset = L * (animalWeight / totalWeight);
    deg = 90 * ((allWeights - animalWeight) / totalWeight);
  }

  leftStack.style.setProperty("--left-offset", `${leftOffset}%`);
  rightStack.style.setProperty("--right-offset", `${rightOffset}%`);
  balanceArrow.style.setProperty("--arrow-rotation", `${deg}deg`);
}

function updatePlacedWeightPositions() {
  const placed = weightElements.filter(
    (element) => element.dataset.onScale === "true",
  );
  const platformHeight = getPlatformHeight();
  const platformWidth = rightStack.getBoundingClientRect().width;
  placed.forEach((element) => {
    if (fallingWeights.has(element)) {
      return;
    }
    const elementWidth = element.getBoundingClientRect().width;
    const storedLeft = Number(element.dataset.dropLeft ?? 0);
    const left = Math.min(
      Math.max(0, storedLeft),
      platformWidth - elementWidth - 4,
    );
    element.style.left = `${Math.max(0, left)}px`;
    element.style.bottom = `${platformHeight}px`;
  });
}

function updateDropHighlightBounds() {
  dropHighlight.style.height = "";
  const balanceRect = balanceStage.getBoundingClientRect();
  const highlightRect = dropHighlight.getBoundingClientRect();
  if (highlightRect.top < balanceRect.top) {
    const offset = balanceRect.top - highlightRect.top;
    const newHeight = Math.max(0, highlightRect.height - offset);
    dropHighlight.style.height = `${newHeight}px`;
  }
}
```

```
function getColumnState(weightElement) {
  return weightColumns.get(weightElement.dataset.columnId);
}

function isTopWeightInColumn(weightElement) {
  const columnState = getColumnState(weightElement);
  if (!columnState) {
    return false;
  }
    return   columnState.weights[columnState.weights.length   -   1]   ===
weightElement;
}

function updateColumnPositions(columnState) {
  columnState.weights.forEach((element, index) => {
    const offset = index * columnState.shift;
    element.style.left = `${offset}px`;
    element.style.bottom = `${offset}px`;
  });
}

function stopWeightFall(weightElement) {
  const fallState = fallingWeights.get(weightElement);
  if (!fallState) {
    return;
  }
  cancelAnimationFrame(fallState.frameId);
  fallingWeights.delete(weightElement);
}

function startWeightFall(weightElement) {
  stopWeightFall(weightElement);
  const platformHeight = getPlatformHeight();
  let velocity = 0;
  let lastTime = performance.now();

  const step = (time) => {
    const delta = (time - lastTime) / 1000;
    lastTime = time;
    velocity += gravity * delta;
    const currentBottom = parseFloat(weightElement.style.bottom) || 0;
    const nextBottom = currentBottom - velocity * delta;

    if (nextBottom <= platformHeight) {
      weightElement.style.bottom = `${platformHeight}px`;
      fallingWeights.delete(weightElement);
      return;
    }

    weightElement.style.bottom = `${nextBottom}px`;
    const frameId = requestAnimationFrame(step);
```

```
    fallingWeights.set(weightElement, { frameId });
  };

  const frameId = requestAnimationFrame(step);
  fallingWeights.set(weightElement, { frameId });
}

function resetWeightToOrigin(weightElement) {
  const origin = weightOrigins.get(weightElement);
  if (!origin) {
    return;
  }
  const columnState = getColumnState(weightElement);
  if (!columnState) {
    return;
  }
  stopWeightFall(weightElement);
  origin.parent.appendChild(weightElement);
  weightElement.dataset.onScale = "false";
  delete weightElement.dataset.dropLeft;
  weightElement.classList.remove("is-dragging");
  weightElement.style.position = "absolute";
  weightElement.style.top = "auto";
  weightElement.style.zIndex = "";
  if (!columnState.weights.includes(weightElement)) {
    columnState.weights.push(weightElement);
  }
  updateColumnPositions(columnState);
}

function placeWeightOnScale(weightElement, { dropX, offsetX = 0 } = {}) {
  const columnState = getColumnState(weightElement);
  if (columnState) {
    columnState.weights = columnState.weights.filter(
      (element) => element !== weightElement,
    );
    updateColumnPositions(columnState);
  }
  const weightRect = weightElement.getBoundingClientRect();
  const stackRect = rightStack.getBoundingClientRect();
  const elementWidth = weightRect.width;
  const maxLeft = stackRect.width - elementWidth - 4;
  const computedLeft =
    dropX === undefined || dropX === null
      ? (stackRect.width - elementWidth) / 2
      : dropX - stackRect.left - offsetX;
  const clampedLeft = Math.min(Math.max(0, computedLeft), maxLeft);
  const platformHeight = getPlatformHeight();
  const startBottom = Math.max(
    platformHeight,
    stackRect.bottom - weightRect.bottom,
```

```
  );

  rightStack.appendChild(weightElement);
  weightElement.dataset.onScale = "true";
  weightElement.dataset.dropLeft = `${clampedLeft}`;
  weightElement.classList.remove("is-dragging");
  weightElement.style.position = "absolute";
  weightElement.style.top = "auto";
  weightElement.style.zIndex = "2";
  weightElement.style.left = `${Math.max(0, clampedLeft)}px`;
  weightElement.style.bottom = `${startBottom}px`;
  startWeightFall(weightElement);
  updateBalancePositions();
}

function handleWeightMouseMove(event) {
  if (!dragState) {
    return;
  }
  dragState.element.style.left = `${event.clientX - dragState.offsetX}px`;
  dragState.element.style.top = `${event.clientY - dragState.offsetY}px`;
}

function handleWeightMouseUp(event) {
  if (!dragState) {
    return;
  }
  document.removeEventListener("mousemove", handleWeightMouseMove);
  document.removeEventListener("mouseup", handleWeightMouseUp);
  dropHighlight.classList.remove("is-active");

  const dropRect = dropHighlight.getBoundingClientRect();
  const isInside =
    event.clientX >= dropRect.left &&
    event.clientX <= dropRect.right &&
    event.clientY >= dropRect.top &&
    event.clientY <= dropRect.bottom;

  if (isInside) {
    placeWeightOnScale(dragState.element, {
      dropX: event.clientX,
      offsetX: dragState.offsetX,
    });
  } else {
    resetWeightToOrigin(dragState.element);
    updateBalancePositions();
  }

  updateDropHighlightBounds();
  dragState = null;
}
```

```
function handleWeightMouseDown(event) {
  if (event.button !== 0) {
    return;
  }
  const weightElement = event.currentTarget;
  if (
    weightElement.dataset.onScale !== "true" &&
    !isTopWeightInColumn(weightElement)
  ) {
    return;
  }
  event.preventDefault();
  const rect = weightElement.getBoundingClientRect();
  stopWeightFall(weightElement);
  dragState = {
    element: weightElement,
    offsetX: event.clientX - rect.left,
    offsetY: event.clientY - rect.top,
  };

  weightElement.classList.add("is-dragging");
  weightElement.style.position = "fixed";
  weightElement.style.left = `${rect.left}px`;
  weightElement.style.top = `${rect.top}px`;
  weightElement.style.bottom = "auto";
  weightElement.style.zIndex = "1000";
  dropHighlight.classList.add("is-active");

  document.addEventListener("mousemove", handleWeightMouseMove);
  document.addEventListener("mouseup", handleWeightMouseUp);
}

function handleWeightDoubleClick(event) {
  if (dragState) {
    return;
  }
  if (event.button !== 0) {
    return;
  }
  const weightElement = event.currentTarget;
  if (weightElement.dataset.onScale === "true") {
    resetWeightToOrigin(weightElement);
    updatePlacedWeightPositions();
    updateBalancePositions();
    return;
  }
  if (!isTopWeightInColumn(weightElement)) {
    return;
  }
  placeWeightOnScale(weightElement);
```

```
}

function renderAnimalsRack(weightAnimalIds) {
  weightsRack.innerHTML = "";
  weightElements = [];
  weightOrigins.clear();
  weightColumns.clear();

  const basis = getCorrectedBasis(80);
  const width = basis;
  const height = basis;
  const shift = 0.2 * basis;

  weightAnimalIds.forEach((animalId, animalIndex) => {
    const animal = animalsById.get(animalId);
    if (!animal) {
      return;
    }
    const column = document.createElement("div");
    column.className = "weight-column";

    const columnWidth = width + 4 * shift;
    const columnHeight = height + 4 * shift;

    column.style.width = `${columnWidth}px`;
    column.style.height = `${columnHeight}px`;

    const columnId = String(animalIndex);
    const columnState = {
      element: column,
      weights: [],
      shift,
    };
    weightColumns.set(columnId, columnState);

    for (let index = 0; index < 5; index += 1) {
      const weightBlock = document.createElement("div");
      weightBlock.className = "weight-block animal-block";
      weightBlock.title = animal.name;
      weightBlock.style.setProperty("--weight-width", `${width}px`);
      weightBlock.style.setProperty("--weight-height", `${height}px`);
      weightBlock.style.setProperty("--weight-head-width", "0px");
      weightBlock.style.setProperty("--weight-head-height", "0px");
      weightBlock.style.setProperty(
        "--animal-image",
        `url(./assets/${animal.id}.jpg)`,
      );
      weightBlock.style.setProperty("--animal-color", animal.color);

      weightBlock.dataset.mass = animal.weight;
      weightBlock.dataset.onScale = "false";
```

```
      weightBlock.dataset.columnId = columnId;
      weightBlock.addEventListener("mousedown", handleWeightMouseDown);
      weightBlock.addEventListener("dblclick", handleWeightDoubleClick);

      weightElements.push(weightBlock);
      weightOrigins.set(weightBlock, {
        parent: column,
      });
      columnState.weights.push(weightBlock);
      column.appendChild(weightBlock);
    }

    updateColumnPositions(columnState);
    weightsRack.appendChild(column);
  });
}

function updateWeightModeAnimal(animal) {
  animalOnScale.src = `./assets/${animal.id}.jpg`;
  animalOnScale.alt = `Фото: ${animal.name}`;
}

function resetWeightsToRack() {
  weightElements.forEach((weightElement) => {
    if (weightElement.dataset.onScale === "true") {
      resetWeightToOrigin(weightElement);
    }
  });
  updatePlacedWeightPositions();
  updateBalancePositions();
  updateDropHighlightBounds();
}

function ensureWeightAnimalsRendered() {
  const state = game.getState();
  const weightAnimals = state?.weightAnimals ?? [];
  const nextKey = weightAnimals.join("|");
  if (!weightAnimals.length || nextKey === weightAnimalKey) {
    return;
  }
  weightAnimalKey = nextKey;
  renderAnimalsRack(weightAnimals);
}

const game = createGameCore({
  expectedMode: "animals",
  elements,
  animalsById,
  modes,
  difficulties,
  difficultyOrder,
```

```
  pickRandomAnimals,
                pickLevelAnimals:                (state)                =>
buildAnimalComparisonLevel(state.usedAnimals),
  addLeaderboardEntry,
  onModeMismatch: () => {
    const state = game.getState();
    if (state?.mode === "weights") {
      window.location.href = "game_weights.html";
      return;
    }
    window.location.href = "game_input.html";
  },
  onRoundStart: (animal) => {
    ensureWeightAnimalsRendered();
    resetWeightsToRack();
    updateWeightModeAnimal(animal);
  },
  getGuessValue,
  onInit: () => {
    modeDescription.textContent =
      "Перетащите животных на правую платформу и нажмите «Проверить вес».";
    ensureWeightAnimalsRendered();
    updateDropHighlightBounds();
  },
});

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

function getGuessValue() {
  const allWeights = getAllWeights();
  if (allWeights <= 0) {
    hintText.textContent = "Добавьте животных на платформу справа.";
    return null;
  }
  return allWeights;
}

function setupInputHandlers() {
  submitButton.addEventListener("click", game.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 (event.key === "Enter") {
      event.preventDefault();
      game.handleSubmit();
            }     else     if     (event.key     ===     "Enter"     &&
!overlay.classList.contains("hidden")) {
      const primaryButton = modalActions.querySelector("button");
      if (primaryButton) {
        primaryButton.click();
      }
    }
  });
}

function initGame() {
  game.init();
  setupInputHandlers();
  updateDropHighlightBounds();
  window.addEventListener("resize", () => {
    updateDropHighlightBounds();
    updatePlacedWeightPositions();
  });
}

initGame();
```

Листинг А.4 – game_input.html.

```
<!doctype html>
<html lang="ru">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
    <title>Угадай вес — Ввод веса</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>
    <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" id="input-panel">
        <label for="weight-input" id="weight-label"
          >Введите вес животного (кг)</label
        >
        <input type="number" id="weight-input" min="0" />
        <p class="mode-description" id="mode-description"></p>
        <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_input.js"></script>
  </body>
</html>
```

Листинг А.5 – game_input.js.

```
import {
  STORAGE_KEYS,
```

```
  difficulties,
  difficultyOrder,
  modes,
  animals,
  addLeaderboardEntry,
  pickRandomAnimals,
} from "./data.js";
import { createGameCore } from "./games.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 modeDescription = document.querySelector("#mode-description");
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]));

const elements = {
  playerNameEl,
  scoreEl,
  progressEl,
  attemptsList,
  timerBlock,
  timerValue,
  hintText,
  overlay,
  modalTitle,
  modalMessage,
  modalActions,
  levelBadge,
};

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 getGuessValue() {
  const value = Number(weightInput.value);
  if (!value || value <= 0) {
    hintText.textContent = "Введите корректное число.";
    return null;
  }
  return value;
}

const game = createGameCore({
  expectedMode: "input",
  elements,
  animalsById,
  modes,
  difficulties,
  difficultyOrder,
  pickRandomAnimals,
  addLeaderboardEntry,
  onModeMismatch: () => {
    const state = game.getState();
    if (state?.mode === "animals") {
      window.location.href = "game_animals.html";
      return;
    }
    window.location.href = "game_weights.html";
  },
  onRoundStart: (animal) => {
    swapCard(animal);
  },
  getGuessValue,
  onClearInput: () => {
    weightInput.value = "";
  },
  onInit: () => {
    modeDescription.textContent =
      "Введите предполагаемый вес животного и нажмите «Проверить вес».";
  },
});

function setupInputHandlers() {
  submitButton.addEventListener("click", game.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();
      game.handleSubmit();
            }     else     if     (event.key     ===     "Enter"     &&
!overlay.classList.contains("hidden")) {
      const primaryButton = modalActions.querySelector("button");
      if (primaryButton) {
        primaryButton.click();
      }
    }
  });
}

function initGame() {
  game.init();
  setupInputHandlers();
}

initGame();
```

Листинг А.6 – game_weights.html.

```
<!doctype html>
<html lang="ru">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
    <title>Угадай вес — Подбор гирьками</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>
    <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="weights-area">
          <div class="weights-rack" id="weights-rack"></div>
        </div>
        <div class="card-stage" id="card-stage">
          <div class="balance-stage" id="balance-stage">
            <div class="balance-side left">
              <div class="platform-stack left" id="left-stack">
                 <img class="animal-on-scale" id="animal-on-scale" alt=""
/>
                        <div  class="platform  left-platform"  id="left-
platform"></div>
              </div>
            </div>
            <div class="balance-arrow" id="balance-arrow">
              <div class="arrow-head"></div>
              <div class="arrow-body"></div>
            </div>
            <div class="balance-side right">
              <div class="platform-stack right" id="right-stack">
                <div class="drop-highlight" id="drop-highlight"></div>
```

```
                    <div  class="platform  right-platform"  id="right-
platform"></div>
              </div>
            </div>
          </div>
        </div>
        <p class="hint" id="hint-text"></p>
      </section>

      <section class="panel" id="input-panel">
        <p class="mode-description" id="mode-description"></p>
        <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_weights.js"></script>
  </body>
</html>
```

Листинг А.7 – game_weights.js.

```
import {
  STORAGE_KEYS,
  difficulties,
  difficultyOrder,
  modes,
  weights,
  animals,
  addLeaderboardEntry,
  pickRandomAnimals,
} from "./data.js";
import { createGameCore } from "./games.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 balanceStage = document.querySelector("#balance-stage");
const weightsRack = document.querySelector("#weights-rack");
const dropHighlight = document.querySelector("#drop-highlight");
const leftStack = document.querySelector("#left-stack");
const rightStack = document.querySelector("#right-stack");
const balanceArrow = document.querySelector("#balance-arrow");
const animalOnScale = document.querySelector("#animal-on-scale");
const modeDescription = document.querySelector("#mode-description");
const hintText = document.querySelector("#hint-text");
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]));
const gravity = 1800;
let dragState = null;
let weightElements = [];
const weightOrigins = new Map();
const weightColumns = new Map();
const fallingWeights = new Map();

document.body.classList.add("mode-weights");

const elements = {
  playerNameEl,
  scoreEl,
  progressEl,
  attemptsList,
  timerBlock,
  timerValue,
  hintText,
  overlay,
  modalTitle,
  modalMessage,
  modalActions,
  levelBadge,
};

function getCorrectedBasis(basisSize) {
  return Math.min(basisSize, window.innerWidth * 0.1);
}

function getAllWeights() {
  return weightElements.reduce(
    (sum, element) =>
              sum    +    (element.dataset.onScale    ===    "true"    ?
Number(element.dataset.mass) : 0),
```

```
    0
  );
}

function getPlatformHeight() {
  return (
    parseFloat(
              getComputedStyle(balanceStage).getPropertyValue("--platform-
height")
    ) || 18
  );
}

function updateBalancePositions() {
  const animal = currentAnimal();
  if (!animal) {
    return;
  }
  const animalWeight = animal.weight;
  const allWeights = getAllWeights();
  const totalWeight = animalWeight + allWeights;
  const L = 50;
  let leftOffset = 0;
  let rightOffset = 0;
  let deg = 0;

  if (totalWeight > 0) {
    leftOffset = L * (allWeights / totalWeight);
    rightOffset = L * (animalWeight / totalWeight);
    deg = 90 * ((allWeights - animalWeight) / totalWeight);
  }

  leftStack.style.setProperty("--left-offset", `${leftOffset}%`);
  rightStack.style.setProperty("--right-offset", `${rightOffset}%`);
  balanceArrow.style.setProperty("--arrow-rotation", `${deg}deg`);
}

function updatePlacedWeightPositions() {
  const placed = weightElements.filter(
    (element) => element.dataset.onScale === "true"
  );
  const platformHeight = getPlatformHeight();
  const platformWidth = rightStack.getBoundingClientRect().width;
  placed.forEach((element) => {
    if (fallingWeights.has(element)) {
      return;
    }
    const elementWidth = element.getBoundingClientRect().width;
    const storedLeft = Number(element.dataset.dropLeft ?? 0);
    const left = Math.min(
      Math.max(0, storedLeft),
```

```
      platformWidth - elementWidth - 4
    );
    element.style.left = `${Math.max(0, left)}px`;
    element.style.bottom = `${platformHeight}px`;
  });
}

function updateDropHighlightBounds() {
  dropHighlight.style.height = "";
  const balanceRect = balanceStage.getBoundingClientRect();
  const highlightRect = dropHighlight.getBoundingClientRect();
  if (highlightRect.top < balanceRect.top) {
    const offset = balanceRect.top - highlightRect.top;
    const newHeight = Math.max(0, highlightRect.height - offset);
    dropHighlight.style.height = `${newHeight}px`;
  }
}

function getColumnState(weightElement) {
  return weightColumns.get(weightElement.dataset.columnId);
}

function isTopWeightInColumn(weightElement) {
  const columnState = getColumnState(weightElement);
  if (!columnState) {
    return false;
  }
    return   columnState.weights[columnState.weights.length   -   1]   ===
weightElement;
}

function updateColumnPositions(columnState) {
  columnState.weights.forEach((element, index) => {
    const offset = index * columnState.shift;
    element.style.left = `${offset}px`;
    element.style.bottom = `${offset}px`;
  });
}

function stopWeightFall(weightElement) {
  const fallState = fallingWeights.get(weightElement);
  if (!fallState) {
    return;
  }
  cancelAnimationFrame(fallState.frameId);
  fallingWeights.delete(weightElement);
}

function startWeightFall(weightElement) {
  stopWeightFall(weightElement);
  const platformHeight = getPlatformHeight();
```

```
  let velocity = 0;
  let lastTime = performance.now();

  const step = (time) => {
    const delta = (time - lastTime) / 1000;
    lastTime = time;
    velocity += gravity * delta;
    const currentBottom = parseFloat(weightElement.style.bottom) || 0;
    const nextBottom = currentBottom - velocity * delta;

    if (nextBottom <= platformHeight) {
      weightElement.style.bottom = `${platformHeight}px`;
      fallingWeights.delete(weightElement);
      return;
    }

    weightElement.style.bottom = `${nextBottom}px`;
    const frameId = requestAnimationFrame(step);
    fallingWeights.set(weightElement, { frameId });
  };

  const frameId = requestAnimationFrame(step);
  fallingWeights.set(weightElement, { frameId });
}

function resetWeightToOrigin(weightElement) {
  const origin = weightOrigins.get(weightElement);
  if (!origin) {
    return;
  }
  const columnState = getColumnState(weightElement);
  if (!columnState) {
    return;
  }
  stopWeightFall(weightElement);
  origin.parent.appendChild(weightElement);
  weightElement.dataset.onScale = "false";
  delete weightElement.dataset.dropLeft;
  weightElement.classList.remove("is-dragging");
  weightElement.style.position = "absolute";
  weightElement.style.top = "auto";
  weightElement.style.zIndex = "";
  if (!columnState.weights.includes(weightElement)) {
    columnState.weights.push(weightElement);
  }
  updateColumnPositions(columnState);
}

function placeWeightOnScale(weightElement, { dropX, offsetX = 0 } = {}) {
  const columnState = getColumnState(weightElement);
  if (columnState) {
```

```
    columnState.weights = columnState.weights.filter(
      (element) => element !== weightElement
    );
    updateColumnPositions(columnState);
  }
  const weightRect = weightElement.getBoundingClientRect();
  const stackRect = rightStack.getBoundingClientRect();
  const elementWidth = weightRect.width;
  const maxLeft = stackRect.width - elementWidth - 4;
  const computedLeft =
    dropX === undefined || dropX === null
      ? (stackRect.width - elementWidth) / 2
      : dropX - stackRect.left - offsetX;
  const clampedLeft = Math.min(Math.max(0, computedLeft), maxLeft);
  const platformHeight = getPlatformHeight();
   const  startBottom  =  Math.max(platformHeight,  stackRect.bottom  -
weightRect.bottom);

  rightStack.appendChild(weightElement);
  weightElement.dataset.onScale = "true";
  weightElement.dataset.dropLeft = `${clampedLeft}`;
  weightElement.classList.remove("is-dragging");
  weightElement.style.position = "absolute";
  weightElement.style.top = "auto";
  weightElement.style.zIndex = "2";
  weightElement.style.left = `${Math.max(0, clampedLeft)}px`;
  weightElement.style.bottom = `${startBottom}px`;
  startWeightFall(weightElement);
  updateBalancePositions();
}

function handleWeightMouseMove(event) {
  if (!dragState) {
    return;
  }
  dragState.element.style.left = `${event.clientX - dragState.offsetX}px`;
  dragState.element.style.top = `${event.clientY - dragState.offsetY}px`;
}

function handleWeightMouseUp(event) {
  if (!dragState) {
    return;
  }
  document.removeEventListener("mousemove", handleWeightMouseMove);
  document.removeEventListener("mouseup", handleWeightMouseUp);
  dropHighlight.classList.remove("is-active");

  const dropRect = dropHighlight.getBoundingClientRect();
  const isInside =
    event.clientX >= dropRect.left &&
    event.clientX <= dropRect.right &&
```

```
    event.clientY >= dropRect.top &&
    event.clientY <= dropRect.bottom;

  if (isInside) {
    placeWeightOnScale(dragState.element, {
      dropX: event.clientX,
      offsetX: dragState.offsetX,
    });
  } else {
    resetWeightToOrigin(dragState.element);
    updateBalancePositions();
  }

  updateDropHighlightBounds();
  dragState = null;
}

function handleWeightMouseDown(event) {
  if (event.button !== 0) {
    return;
  }
  const weightElement = event.currentTarget;
      if      (weightElement.dataset.onScale      !==      "true"      &&
!isTopWeightInColumn(weightElement)) {
    return;
  }
  event.preventDefault();
  const rect = weightElement.getBoundingClientRect();
  stopWeightFall(weightElement);
  dragState = {
    element: weightElement,
    offsetX: event.clientX - rect.left,
    offsetY: event.clientY - rect.top,
  };

  weightElement.classList.add("is-dragging");
  weightElement.style.position = "fixed";
  weightElement.style.left = `${rect.left}px`;
  weightElement.style.top = `${rect.top}px`;
  weightElement.style.bottom = "auto";
  weightElement.style.zIndex = "1000";
  dropHighlight.classList.add("is-active");

  document.addEventListener("mousemove", handleWeightMouseMove);
  document.addEventListener("mouseup", handleWeightMouseUp);
}

function handleWeightDoubleClick(event) {
  if (dragState) {
    return;
  }
```

```
  if (event.button !== 0) {
    return;
  }
  const weightElement = event.currentTarget;
  if (weightElement.dataset.onScale === "true") {
    resetWeightToOrigin(weightElement);
    updatePlacedWeightPositions();
    updateBalancePositions();
    return;
  }
  if (!isTopWeightInColumn(weightElement)) {
    return;
  }
  placeWeightOnScale(weightElement);
}

function renderWeightsRack() {
  weightsRack.innerHTML = "";
  weightElements = [];
  weightOrigins.clear();
  weightColumns.clear();

  const sortedWeights = [...weights].sort((a, b) => b.mass - a.mass);

  sortedWeights.forEach((weight, weightIndex) => {
    const column = document.createElement("div");
    column.className = "weight-column";

    const correctedBasis = getCorrectedBasis(weight.basisSize);
    const width = correctedBasis;
    const height = correctedBasis * 1.5;
    const headWidth = correctedBasis * 0.5;
    const headHeight = correctedBasis * 0.25;
    const shift = 0.2 * weight.basisSize;
    const columnWidth = width + (weight.amount - 1) * shift;
    const columnHeight = height + (weight.amount - 1) * shift + headHeight;

    column.style.width = `${columnWidth}px`;
    column.style.height = `${columnHeight}px`;

    const columnId = String(weightIndex);
    const columnState = {
      element: column,
      weights: [],
      shift,
    };
    weightColumns.set(columnId, columnState);

    for (let index = 0; index < weight.amount; index += 1) {
      const weightBlock = document.createElement("div");
      weightBlock.className = "weight-block";
```

```
      weightBlock.textContent = weight.mass;
      weightBlock.style.setProperty("--weight-width", `${width}px`);
      weightBlock.style.setProperty("--weight-height", `${height}px`);
                    weightBlock.style.setProperty("--weight-head-width",
`${headWidth}px`);
                   weightBlock.style.setProperty("--weight-head-height",
`${headHeight}px`);
      const hue = 210 - weightIndex * 12;
      weightBlock.style.setProperty(
        "--weight-color",
        `hsl(${hue} 45% 45%)`
      );

      const offset = index * shift;
      weightBlock.dataset.mass = weight.mass;
      weightBlock.dataset.onScale = "false";
      weightBlock.dataset.columnId = columnId;
      weightBlock.addEventListener("mousedown", handleWeightMouseDown);
      weightBlock.addEventListener("dblclick", handleWeightDoubleClick);

      weightElements.push(weightBlock);
      weightOrigins.set(weightBlock, {
        parent: column,
      });
      columnState.weights.push(weightBlock);
      column.appendChild(weightBlock);
    }

    updateColumnPositions(columnState);
    weightsRack.appendChild(column);
  });
}

function updateWeightModeAnimal(animal) {
  animalOnScale.src = `./assets/${animal.id}.jpg`;
  animalOnScale.alt = `Фото: ${animal.name}`;
}

function resetWeightsToRack() {
  weightElements.forEach((weightElement) => {
    if (weightElement.dataset.onScale === "true") {
      resetWeightToOrigin(weightElement);
    }
  });
  updatePlacedWeightPositions();
  updateBalancePositions();
  updateDropHighlightBounds();
}

const game = createGameCore({
  expectedMode: "weights",
```

```
  elements,
  animalsById,
  modes,
  difficulties,
  difficultyOrder,
  pickRandomAnimals,
  addLeaderboardEntry,
  onModeMismatch: () => {
    const state = game.getState();
    if (state?.mode === "animals") {
      window.location.href = "game_animals.html";
      return;
    }
    window.location.href = "game_input.html";
  },
  onRoundStart: (animal) => {
    resetWeightsToRack();
    updateWeightModeAnimal(animal);
  },
  getGuessValue,
  onInit: () => {
    modeDescription.textContent =
      "Перетащите гири на правую платформу и нажмите «Проверить вес».";
    renderWeightsRack();
    updateDropHighlightBounds();
  },
});

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

function getGuessValue() {
  const allWeights = getAllWeights();
  if (allWeights <= 0) {
    hintText.textContent = "Добавьте гири на платформу справа.";
    return null;
  }
  return allWeights;
}

function setupInputHandlers() {
  submitButton.addEventListener("click", game.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 (event.key === "Enter") {
      event.preventDefault();
      game.handleSubmit();
            }     else     if     (event.key     ===     "Enter"     &&
!overlay.classList.contains("hidden")) {
      const primaryButton = modalActions.querySelector("button");
      if (primaryButton) {
        primaryButton.click();
      }
    }
  });
}

function initGame() {
  game.init();
  setupInputHandlers();
  updateDropHighlightBounds();
  window.addEventListener("resize", () => {
    updateDropHighlightBounds();
    updatePlacedWeightPositions();
  });
}

initGame();
```

Листинг А.8 – games.js.

```
import { STORAGE_KEYS } from "./data.js";

export function createGameCore({
  expectedMode,
  elements,
  animalsById,
  modes,
  difficulties,
  difficultyOrder,
  pickRandomAnimals,
  pickLevelAnimals,
  addLeaderboardEntry,
  onModeMismatch,
  onRoundStart,
  getGuessValue,
  onClearInput,
  onInit,
}) {
  let state = loadState();
  let timerId = null;
  let timeRemaining = null;
```

```
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 getModeSettings() {
  return modes.find((mode) => mode.id === state.mode);
}

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 renderAttempts() {
  elements.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");
    }
    elements.attemptsList.appendChild(dot);
  }
}

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

```
}

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

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

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

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

function computeFinalScore() {
  const highestMultiplier =
    difficulties[state.highestDifficulty]?.multiplier || 1;
  const timedMultiplier = state.timeMode ? 2 : 1;
  const modeMultiplier = getModeSettings()?.modeModifier || 1;
```

```
    return state.baseScore * highestMultiplier * timedMultiplier *
modeMultiplier;
  }

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

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

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

    const focusTarget = elements.modalActions.querySelector("button");
```

```
    if (focusTarget) {
      focusTarget.focus();
    }
  }

  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 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;
      elements.hintText.textContent  =  `Верно!  Вес  ${animal.name}  ≈
${animal.weight} кг.`;
    updateHeader();
    saveState();
    onRoundStart?.(currentAnimal());
    startTimer();
  }
```

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

function moveToNextLevel() {
  const currentIndex = difficultyOrder.indexOf(state.difficulty);
  const nextDifficulty = difficultyOrder[currentIndex + 1];
  if (!nextDifficulty) {
    return;
  }
  state.difficulty = nextDifficulty;
  state.highestDifficulty = nextDifficulty;
  if (pickLevelAnimals) {
    const nextLevel = pickLevelAnimals(state);
    state.levelAnimals = nextLevel.levelAnimals;
    state.usedAnimals = nextLevel.usedAnimals;
    if (nextLevel.weightAnimals) {
      state.weightAnimals = nextLevel.weightAnimals;
    }
  } else {
    const { ids: levelAnimals, isExhausted } = pickRandomAnimals(
      state.usedAnimals,
      5,
    );
    if (isExhausted) {
      state.usedAnimals = state.usedAnimals.slice(5);
    }
    state.levelAnimals = levelAnimals;
    state.usedAnimals = [...state.usedAnimals, ...state.levelAnimals];
  }
  state.baseScore = 0;
  state.currentIndex = 0;
  state.completedAnimals = 0;
  state.failedAttemptsCurrent = 0;
  state.attemptsLeft = difficulties[nextDifficulty].attempts;
  elements.hintText.textContent = "Новый уровень!";
  updateHeader();
  saveState();
```

```
    onRoundStart?.(currentAnimal());
    startTimer();
  }

  function handleSubmit() {
    const value = getGuessValue();
    if (value === null) {
      return;
    }
    const animal = currentAnimal();
    if (!animal) {
      return;
    }
    clearTimer();
    onClearInput?.();
    const tolerance = animal.weight * 0.1;
    const isCorrect = Math.abs(animal.weight - value) <= tolerance;
    if (isCorrect) {
      handleCorrectGuess(animal);
    } else {
      handleIncorrectGuess(value > animal.weight);
    }
  }

  function init() {
    if (!state) {
      window.location.href = "menu.html";
      return;
    }
    if (!state.mode) {
      state.mode = expectedMode;
    }
    if (state.mode !== expectedMode) {
      onModeMismatch?.();
      return;
    }
    elements.playerNameEl.textContent = state.playerName;
    updateHeader();
    onInit?.();
    onRoundStart?.(currentAnimal());
    startTimer();
  }

  return {
    init,
    handleSubmit,
    getState: () => state,
    currentAnimal,
    updateHeader,
  };
}
```

Листинг А.9 – leaderboard.html

```
<!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 А.10 – leaderboard.js

```
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 А.11 – menu.html

```
<!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 режим и уровень сложности стрелками, введите имя и
начните
          игру клавишей Enter.
        </p>
      </section>
```

```
      <section class="panel">
        <h2 class="panel-title">Режим</h2>
        <div class="menu-grid" id="mode-options"></div>
      </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 А.12 – menu.js

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

```
const modeContainer = document.querySelector("#mode-options");
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 selectedModeIndex = 0;
let menuFocusIndex = 0;
const modeButtons = [];
const difficultyButtons = [];

const renderModeSettings = () => {
  modeContainer.innerHTML = "";
  modeButtons.length = 0;

  modeOrder.forEach((key, index) => {
    const settings = modes.find((mode) => mode.id === key);
    if (!settings) {
      return;
    }
    const card = document.createElement("button");
    card.type = "button";
    card.className = "difficulty-card";
    card.dataset.mode = key;
    card.innerHTML = `
    <div class="difficulty-title">${settings.modeName}</div>
    <p>Модификатор: x${settings.modeModifier}</p>`;

    card.addEventListener("click", () => {
      selectMode(index);
      setMenuFocus(0);
    });
    card.addEventListener("focus", () => setMenuFocus(0));
    modeButtons.push(card);
    modeContainer.appendChild(card);
  });
  selectMode(selectedModeIndex);
  if (menuFocusIndex === 0) {
    focusCurrentMenuItem();
  }
};

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(1);
    });
    card.addEventListener("focus", () => setMenuFocus(1));
    difficultyButtons.push(card);
    difficultyContainer.appendChild(card);
  });
  selectDifficulty(selectedIndex);
  if (menuFocusIndex === 1) {
    focusCurrentMenuItem();
  }
};

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

function selectMode(index) {
  selectedModeIndex = index;
  modeButtons.forEach((button, idx) => {
    button.classList.toggle("is-selected", idx === selectedModeIndex);
  });
}

function selectDifficulty(index) {
  selectedIndex = index;
  difficultyButtons.forEach((button, idx) => {
```

```
    button.classList.toggle("is-selected", idx === selectedIndex);
  });
}

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

function createAnimalComparisonState() {
  const difficultyKey = difficultyOrder[selectedIndex];
  const { levelAnimals, usedAnimals, weightAnimals } =
    buildAnimalComparisonLevel([]);
  return {
    playerName: nameInput.value.trim(),
    mode: "animals",
    difficulty: difficultyKey,
    timeMode: timeToggle.checked,
    highestDifficulty: difficultyKey,
    baseScore: 0,
    usedAnimals,
    levelAnimals,
    weightAnimals,
    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 selectedMode = modeOrder[selectedModeIndex];
  const state =
    selectedMode === "animals"
      ? createAnimalComparisonState()
      : createInitialState();
  localStorage.setItem(STORAGE_KEYS.state, JSON.stringify(state));
  window.location.href =
    state.mode === "weights"
      ? "game_weights.html"
      : state.mode === "animals"
        ? "game_animals.html"
        : "game_input.html";
}

startButton.addEventListener("click", startGame);

function setMenuFocus(index) {
  menuFocusIndex = Math.max(0, Math.min(index, 5));
}

function focusCurrentMenuItem() {
  if (menuFocusIndex === 0) {
    modeButtons[selectedModeIndex]?.focus();
    return;
  }
  if (menuFocusIndex === 1) {
    difficultyButtons[selectedIndex]?.focus();
    return;
  }
  if (menuFocusIndex === 2) {
    timeToggle.focus();
    return;
  }
  if (menuFocusIndex === 3) {
    nameInput.focus();
    return;
  }
  if (menuFocusIndex === 4) {
    startButton.focus();
    return;
  }
  leaderboardLink?.focus();
}

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

nameInput.addEventListener("focus", () => setMenuFocus(3));
```

```
timeToggle.addEventListener("focus", () => setMenuFocus(2));
startButton.addEventListener("focus", () => setMenuFocus(4));
leaderboardLink?.addEventListener("focus", () => setMenuFocus(5));

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();
    selectMode((selectedModeIndex + 1) % modeButtons.length);
    focusCurrentMenuItem();
    return;
  }
  if (event.key === "ArrowLeft" && menuFocusIndex === 0) {
    event.preventDefault();
       selectMode((selectedModeIndex   -   1   +   modeButtons.length)   %
modeButtons.length);
    focusCurrentMenuItem();
    return;
  }
  if (event.key === "ArrowRight" && menuFocusIndex === 1) {
    event.preventDefault();
    selectDifficulty((selectedIndex + 1) % difficultyButtons.length);
    focusCurrentMenuItem();
    return;
  }
  if (event.key === "ArrowLeft" && menuFocusIndex === 1) {
    event.preventDefault();
    selectDifficulty(
            (selectedIndex   -   1   +   difficultyButtons.length)   %
difficultyButtons.length
    );
    focusCurrentMenuItem();
    return;
  }
  if (event.key === "Enter") {
    event.preventDefault();
    if (menuFocusIndex === 0) {
      selectMode((selectedModeIndex + 1) % modeButtons.length);
      focusCurrentMenuItem();
      return;
    }
    if (menuFocusIndex === 1) {
```

```
      selectDifficulty((selectedIndex + 1) % difficultyButtons.length);
      focusCurrentMenuItem();
      return;
    }
    if (menuFocusIndex === 2) {
      timeToggle.checked = !timeToggle.checked;
      timeToggle.dispatchEvent(new Event("change"));
      focusCurrentMenuItem();
      return;
    }
    if (menuFocusIndex === 3) {
      setMenuFocus(4);
      focusCurrentMenuItem();
      return;
    }
    if (menuFocusIndex === 4) {
      startGame();
      return;
    }
    leaderboardLink?.click();
  }
});

focusCurrentMenuItem();
```

Листинг А.13 – styles.css

```
: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: 16px 8vw 30px;
}

.panel {
  background: rgba(18, 22, 32, 0.92);
  border-radius: 12px;
  padding: 16px;
  box-shadow: 0 18px 40px rgba(0, 0, 0, 0.35);
}

.panel-title {
  margin: 0 0 16px;
  font-size: 1.25rem;
}

.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;
```

```
}

.is-hidden {
  display: none !important;
}

.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;
}

.balance-stage {
  --platform-width: 90%;
  --platform-height: 18px;
  position: relative;
  width: 100%;
  min-height: 420px;
  display: grid;
  grid-template-columns: 1fr 1fr;
  gap: 16px;
}

.balance-stage::before {
  content: "";
  position: absolute;
  top: 0;
  bottom: 0;
  left: 50%;
  width: 1px;
  background: rgba(255, 255, 255, 0.08);
  transform: translateX(-50%);
}

.balance-side {
  position: relative;
  min-height: 420px;
}

.platform-stack {
  position: absolute;
  width: var(--platform-width);
  height: 360px;
  left: calc(50% - var(--platform-width) / 2);
}

.platform-stack.left {
```

```
  bottom: var(--left-offset, 0%);
}

.platform-stack.right {
  bottom: var(--right-offset, 50%);
}

.platform {
  position: absolute;
  bottom: 0;
  left: 0;
  width: 100%;
  height: var(--platform-height);
  background: linear-gradient(180deg, #f5d742, #caa32b);
  box-shadow: 0 8px 20px rgba(0, 0, 0, 0.35);
}

.animal-on-scale {
  position: absolute;
  left: 50%;
  transform: translateX(-50%);
  bottom: calc(var(--platform-height));
  width: min(360px, 90%);
  border-radius: 16px;
}

.drop-highlight {
  position: absolute;
  left: 0;
  bottom: calc(var(--platform-height) + 12px);
  width: 100%;
  height: min(300px, calc(100% - var(--platform-height) - 24px));
  border-radius: 14px;
  border: 2px dashed rgba(76, 207, 108, 0.75);
  background: rgba(76, 207, 108, 0.15);
  opacity: 0;
  pointer-events: none;
  transition: opacity 0.2s ease;
}

.drop-highlight.is-active {
  opacity: 1;
}

.balance-arrow {
  position: absolute;
  left: 50%;
  top: 50%;
  width: 120px;
  height: 120px;
  transform: translate(-50%, -100%) rotate(var(--arrow-rotation, -90deg));
```

```
  transform-origin: 50% 100%;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  pointer-events: none;
  z-index: 1;
}

.arrow-body {
  width: 12px;
  height: 100px;
  background: #f4f4f6;
}

.arrow-head {
  width: 0;
  height: 0;
  border-left: 18px solid transparent;
  border-right: 18px solid transparent;
  border-bottom: 26px solid #ff595e;
  margin-top: -6px;
}

.weights-area {
  display: flex;
  justify-content: flex-end;
  margin-bottom: 16px;
}

.weights-rack {
  position: relative;
  display: flex;
  align-items: flex-end;
  flex-wrap: wrap;
  gap: 8px;
  z-index: 2;
}

.weight-column {
  position: relative;
  display: block;
}

.weight-block {
  position: absolute;
  display: flex;
  align-items: center;
  justify-content: center;
  background: var(--weight-color, #4d6a90);
  color: #f4f4f6;
```

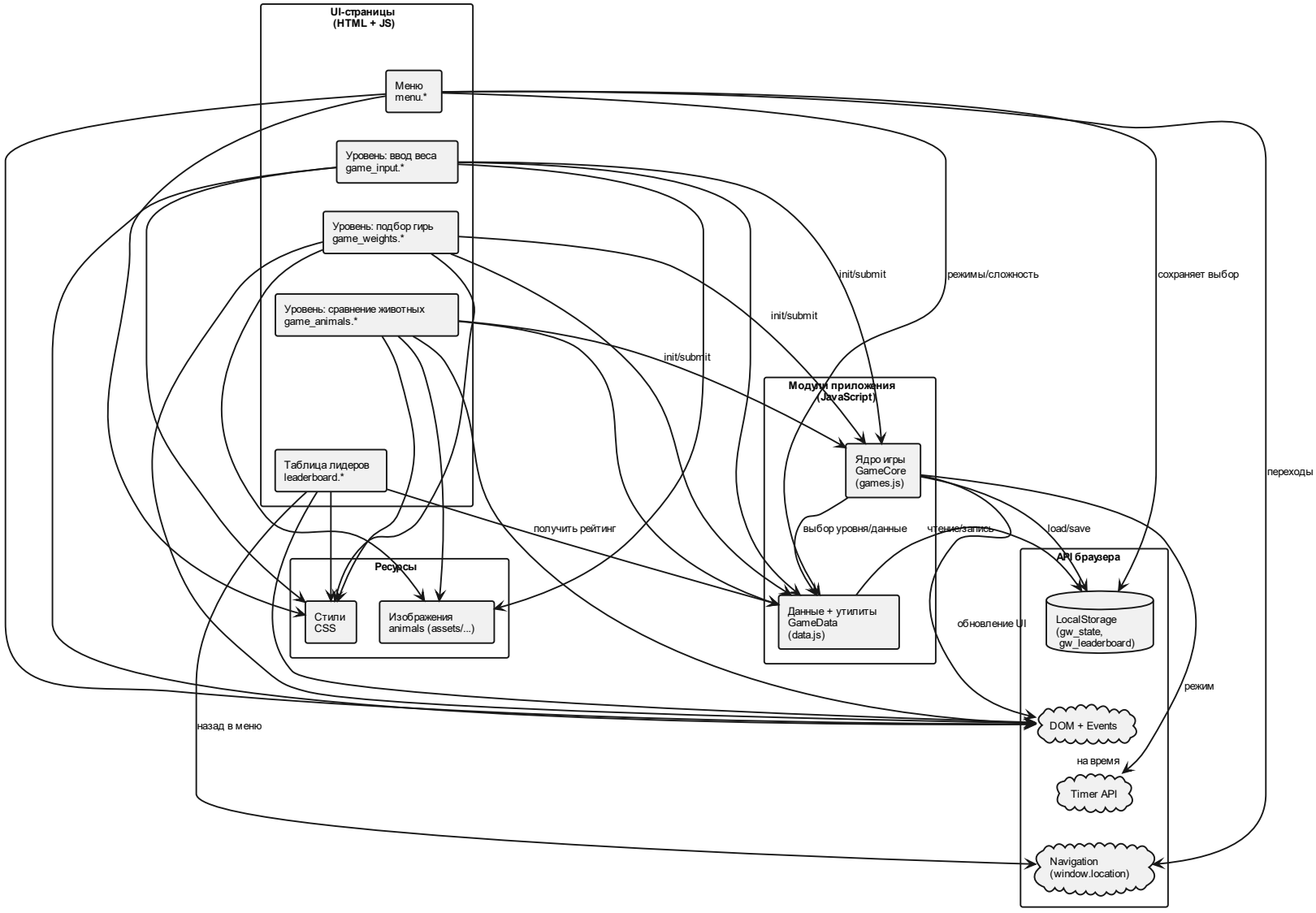
```
  font-weight: 700;
  width: var(--weight-width);
  height: var(--weight-height);
  border-radius: 10px;
  cursor: grab;
  user-select: none;
  box-shadow: 0 8px 16px rgba(0, 0, 0, 0.35);
  transition:
    transform 0.2s ease,
    box-shadow 0.2s ease;
}

.weight-block::before {
  content: "";
  position: absolute;
  top: calc(-1 * var(--weight-head-height));
  left: 50%;
  transform: translateX(-50%);
  width: var(--weight-head-width);
  height: var(--weight-head-height);
  background: inherit;
  border-radius: 8px;
}

.weight-block.is-dragging {
  cursor: grabbing;
  transform: scale(1.05);
  box-shadow: 0 12px 26px rgba(0, 0, 0, 0.45);
  z-index: 5;
}

.animal-block {
  background-color: var(--animal-color, #4d6a90);
  background-image: var(--animal-image);
  background-size: cover;
  background-position: center;
  color: transparent;
  border: 2px solid rgba(255, 255, 255, 0.35);
  text-shadow: none;
}

.animal-block::before {
  display: none;
}

.mode-description {
  margin-top: 12px;
  color: rgba(255, 255, 255, 0.7);
}

.mode-weights #weight-input {
```

```
  display: none;
}

.mode-weights #weight-label {
  display: none;
}

.hint {
  min-height: 24px;
  color: #ffd166;
  margin-top: 10px;
  margin-bottom: 0px;
}

.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%;
```

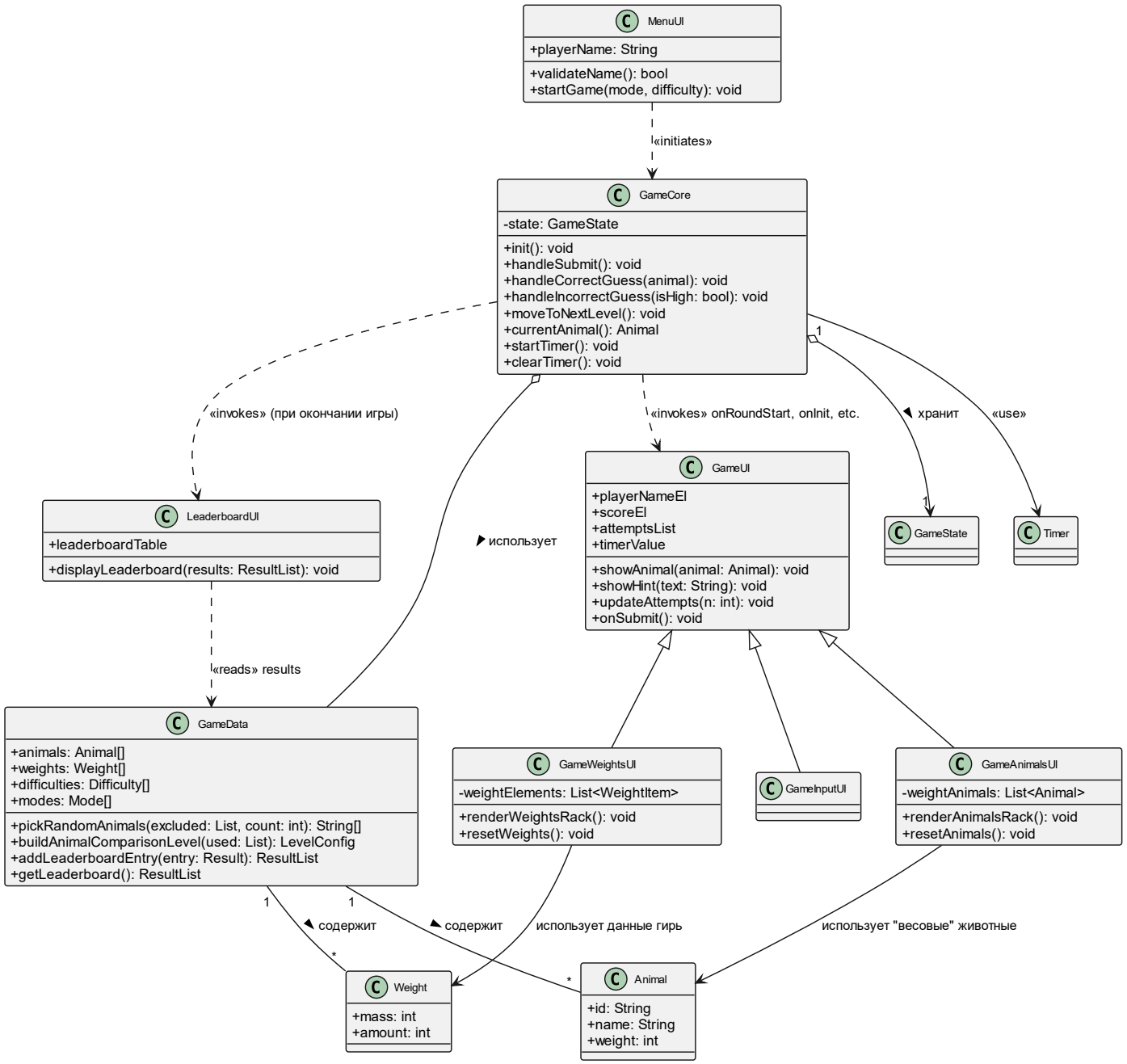
```
  display: flex;
  flex-direction: column;
  gap: 16px;
}

.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. Структура модулей и компонентов приложения «Определи вес»

Диаграмма 2.1 — Структура модулей и компонентов приложения «Определи вес»



Приложение В. UML-диаграмма классов архитектуры приложения «Определи вес



Приложение Г. Блок-схема алгоритма проверки попытки угадывания веса