**ПОЛНАЯ МАТЕМАТИЧЕСКАЯ ЛОГИКА РАСЧЁТОВ**

**1. БАЗОВЫЕ ПОНЯТИЯ И КОНСТАНТЫ**

**Константы системы**

**const ACTIVE\_PV\_BASE = 330; // Базовое значение для одной "единицы"**

**const GRID\_SIZE = 70; // Размер сетки (для справки)**

**Основные термины**

**PV (Personal Volume) — личный объём партнёра**

* **Отображается как X/330pv (например, 150/330pv)**
* **Минимум: 0**
* **Максимум: 330 (полный)**
* **Карточка считается "полной", если PV ≥ 330**

**Баланс — количество баллов в "ногах" (левой и правой ветках)**

* **Отображается как L / R (например, 15 / 23)**
* **Левая нога: сумма баллов от всех партнёров в левой ветке**
* **Правая нога: сумма баллов от всех партнёров в правой ветке**

**Актив-PV — активные заказы, ручное управление баллами**

* **Отображается как L / R (например, 220 / 110)**
* **Пользователь добавляет вручную кнопками ±1, ±10**
* **При достижении 330 PV → создаётся 1 единица → добавляется к балансу**

**Этапы и Циклы — прогрессия в системе**

* **1 цикл = 72 балла**
* **Этапы внутри цикла: 6 → 12 → 18 → 36 (сумма = 72)**

**2. ИЕРАРХИЧЕСКАЯ СТРУКТУРА**

**Определение родителя и ребёнка**

**Правило определения родителя:**

**Родитель — это карточка, которая выше на экране (меньшая координата Y)**

**function pickParent(byId, line) {**

**const startCard = byId.get(line.startId);**

**const endCard = byId.get(line.endId);**

**let parent, child, sideOnParent;**

**// Сравниваем Y-координаты**

**if (startCard.y <= endCard.y) {**

**parent = startCard;**

**child = endCard;**

**sideOnParent = line.startSide; // сторона на родителе**

**} else {**

**parent = endCard;**

**child = startCard;**

**sideOnParent = line.endSide;**

**}**

**return {**

**parentId: parent.id,**

**childId: child.id,**

**parentSide: normalizeSide(sideOnParent)**

**};**

**}**

**Нормализация стороны (ноги)**

**Правило:**

* **right → правая нога**
* **left, bottom, top, или любое другое → левая нога**

**function normalizeSide(side) {**

**const s = String(side || '').toLowerCase();**

**if (s === 'right') return 'right';**

**return 'left'; // всё остальное считается левой**

**}**

**Важно: сторона определяется на родителе, не на ребёнке!**

**3. РАСЧЁТ PV**

**Парсинг PV из HTML карточки**

**Формат в HTML: <span>150/330pv</span>**

**function parsePv330(html) {**

**// Ищем паттерн: число/330pv**

**const regex = /(\d+)\s\*\/\s\*330\s\*pv/i;**

**const match = regex.exec(html || '');**

**const value = match ? parseInt(match[1], 10) : 0;**

**const isFull = value >= 330;**

**return { value, isFull };**

**}**

**Примеры:**

* **"150/330pv" → { value: 150, isFull: false }**
* **"330/330pv" → { value: 330, isFull: true }**
* **"0/330pv" → { value: 0, isFull: false }**
* **Нет паттерна → { value: 0, isFull: false }**

**4. РАСЧЁТ БАЛЛОВ В НОГАХ**

**Основной алгоритм**

**Принцип: от каждой полной карточки (PV ≥ 330) поднимается 1 балл вверх по иерархии до корня.**

**Шаги алгоритма:**

**function calculateBalls(cards, lines) {**

**// 1. Определяем иерархию**

**const parentOf = {}; // child -> { parentId, side }**

**lines.forEach(line => {**

**const { parentId, childId, parentSide } = pickParent(line);**

**// Если у ребёнка уже есть родитель - выбираем того, кто выше**

**if (parentOf[childId]) {**

**const oldParent = cards.find(c => c.id === parentOf[childId].parentId);**

**const newParent = cards.find(c => c.id === parentId);**

**// Выбираем родителя с меньшим Y**

**if (newParent.y < oldParent.y) {**

**parentOf[childId] = { parentId, side: parentSide };**

**}**

**} else {**

**parentOf[childId] = { parentId, side: parentSide };**

**}**

**});**

**// 2. Парсим PV всех карточек**

**const pv = {};**

**const isFull = {};**

**cards.forEach(card => {**

**const parsed = parsePv330(card.bodyHTML);**

**pv[card.id] = parsed.value;**

**isFull[card.id] = parsed.isFull;**

**});**

**// 3. Инициализируем баллы**

**const balls = {};**

**cards.forEach(card => {**

**balls[card.id] = { L: 0, R: 0, total: 0 };**

**});**

**// 4. ПОДНИМАЕМ БАЛЛЫ ВВЕРХ**

**cards.forEach(card => {**

**// Только полные карточки генерируют баллы**

**if (!isFull[card.id]) return;**

**let currentId = card.id;**

**let iterations = 0;**

**const MAX\_ITERATIONS = 5000; // защита от бесконечного цикла**

**// Поднимаемся вверх по цепочке родителей**

**while (parentOf[currentId] && iterations < MAX\_ITERATIONS) {**

**iterations++;**

**const { parentId, side } = parentOf[currentId];**

**// Добавляем 1 балл к соответствующей ноге родителя**

**if (side === 'right') {**

**balls[parentId].R += 1;**

**} else { // 'left'**

**balls[parentId].L += 1;**

**}**

**balls[parentId].total += 1;**

**// Переходим к следующему уровню**

**currentId = parentId;**

**}**

**});**

**return balls;**

**}**

**Пример расчёта**

**Структура:**

**A (корень)**

**/ \**

**/ \**

**B C**

**/ \ / \**

**D E F G**

**PV карточек:**

**A: 100/330 (неполная)**

**B: 330/330 (полная) ✓**

**C: 200/330 (неполная)**

**D: 330/330 (полная) ✓**

**E: 150/330 (неполная)**

**F: 330/330 (полная) ✓**

**G: 0/330 (неполная)**

**Связи:**

**B → левая нога A**

**C → правая нога A**

**D → левая нога B**

**E → правая нога B**

**F → левая нога C**

**G → правая нога C**

**Расчёт:**

**1. От D (полная):**

**- B.L += 1 → B: { L: 1, R: 0 }**

**- A.L += 1 → A: { L: 1, R: 0 }**

**2. От E: не полная, пропускаем**

**3. От B (полная):**

**- A.L += 1 → A: { L: 2, R: 0 }**

**4. От F (полная):**

**- C.L += 1 → C: { L: 1, R: 0 }**

**- A.R += 1 → A: { L: 2, R: 1 }**

**5. От G: не полная, пропускаем**

**Итоговый баланс:**

**A: L=2, R=1, total=3**

**B: L=1, R=0, total=1**

**C: L=1, R=0, total=1**

**D: L=0, R=0, total=0**

**E: L=0, R=0, total=0**

**F: L=0, R=0, total=0**

**G: L=0, R=0, total=0**

**5. РАСЧЁТ ЭТАПОВ И ЦИКЛОВ**

**Таблица этапов**

| **Этап** | **Диапазон баллов** | **Необходимо баллов** | **Накопленно** |
| --- | --- | --- | --- |
| **0** | **0 – 5** | **6** | **0** |
| **1** | **6 – 17** | **12** | **6** |
| **2** | **18 – 35** | **18** | **18** |
| **3** | **36 – 71** | **36** | **36** |
| **4** | **72+** | **новый цикл** | **72** |

**1 цикл = 72 балла (6 + 12 + 18 + 36)**

**Алгоритм расчёта**

**function calcStagesAndCycles(totalBalls) {**

**const thresholds = [6, 12, 18, 36];**

**// Количество полных циклов**

**const cycles = Math.floor(totalBalls / 72);**

**// Остаток баллов в текущем цикле**

**let within = totalBalls % 72;**

**// Определяем текущий этап**

**let stage = 0; // текущий этап (0-4)**

**let toNext = 6; // баллов до следующего этапа**

**let accumulated = 0; // накоплено баллов в текущем цикле**

**for (let i = 0; i < thresholds.length; i++) {**

**// Если баллов достаточно для прохождения этапа**

**if (within >= accumulated + thresholds[i]) {**

**accumulated += thresholds[i];**

**stage = i + 1;**

**} else {**

**// Не хватает баллов - вычисляем до следующего**

**toNext = accumulated + thresholds[i] - within;**

**break;**

**}**

**}**

**// Если прошли все 4 этапа**

**if (stage === 4) {**

**toNext = 72 - within; // до следующего цикла**

**}**

**return { cycles, stage, toNext };**

**}**

**Примеры расчёта этапов**

**// Пример 1: 5 баллов**

**calcStagesAndCycles(5);**

**// → { cycles: 0, stage: 0, toNext: 1 }**

**// Объяснение: 0 циклов, этап 0, до этапа 1 нужно ещё 1 балл**

**// Пример 2: 6 баллов**

**calcStagesAndCycles(6);**

**// → { cycles: 0, stage: 1, toNext: 12 }**

**// Объяснение: 0 циклов, этап 1, до этапа 2 нужно ещё 12 баллов**

**// Пример 3: 20 баллов**

**calcStagesAndCycles(20);**

**// → { cycles: 0, stage: 2, toNext: 16 }**

**// Объяснение: 0 циклов, этап 2, до этапа 3 нужно ещё 16 баллов**

**// Пример 4: 72 балла**

**calcStagesAndCycles(72);**

**// → { cycles: 1, stage: 0, toNext: 6 }**

**// Объяснение: 1 цикл пройден, начался новый цикл**

**// Пример 5: 150 баллов**

**calcStagesAndCycles(150);**

**// → { cycles: 2, stage: 0, toNext: 6 }**

**// Объяснение: 2 цикла, 6 баллов в новом цикле (150 = 72\*2 + 6)**

**6. СИСТЕМА АКТИВ-PV**

**Концепция**

**Актив-PV — это ручная система добавления баллов, которая имитирует активные заказы партнёров.**

**Структура данных**

**На каждой карточке есть скрытый элемент с данными:**

**<span class="active-pv-hidden" style="display: none;"**

**data-btnl="0" <!-- кнопки нажатые слева -->**

**data-btnr="0" <!-- кнопки нажатые справа -->**

**data-abonusl="0" <!-- устаревшее, не используется -->**

**data-abonusr="0" <!-- устаревшее, не используется -->**

**data-locall="0" <!-- единицы слева (330pv = 1 единица) -->**

**data-localr="0"> <!-- единицы справа -->**

**</span>**

**Механизм работы**

**Отображаемое значение = остаток PV Единицы (units) = скрытые единицы в data-locall/localr**

**// Формула:**

**Актив-PV(отображаемый) = остаток от деления на 330**

**Единицы = целая часть от деления на 330**

**// Пример:**

**// Пользователь нажал кнопки на 450 PV слева**

**450 / 330 = 1 (целая часть) → 1 единица → добавляется к балансу**

**450 % 330 = 120 (остаток) → отображается как "120 / X"**

**Алгоритм propagateActivePvUp**

**Это ключевая функция, которая распространяет Актив-PV вверх по иерархии.**

**function propagateActivePvUp(cardEl, side, amount) {**

**const BASE = 330; // ACTIVE\_PV\_BASE**

**let currentElement = cardEl;**

**let currentSide = side; // 'L' или 'R'**

**let carry = amount; // сколько несём вверх**

**while (currentElement) {**

**// 1. ПОЛУЧАЕМ ТЕКУЩИЕ ЗНАЧЕНИЯ**

**const activePV = parseActivePV(currentElement);**

**let L = activePV.L; // отображаемое значение слева**

**let R = activePV.R; // отображаемое значение справа**

**const hidden = currentElement.querySelector('.active-pv-hidden');**

**const localKey = currentSide === 'L' ? 'locall' : 'localr';**

**// Получаем скрытые единицы**

**let storedUnits = parseInt(hidden.dataset[localKey] || '0', 10);**

**if (!Number.isFinite(storedUnits) || storedUnits < 0) {**

**storedUnits = 0;**

**}**

**// 2. ВЫЧИСЛЯЕМ ТЕКУЩЕЕ СОСТОЯНИЕ**

**const currentRemainder = (currentSide === 'L') ? L : R;**

**const totalBefore = currentRemainder + storedUnits \* BASE;**

**// 3. ПРИМЕНЯЕМ ИЗМЕНЕНИЕ**

**let delta = carry;**

**// Защита от отрицательных значений**

**if (delta < 0) {**

**const minDelta = -currentRemainder;**

**if (delta < minDelta) {**

**delta = minDelta; // можем убрать только до 0**

**}**

**}**

**// Новый остаток**

**let newRemainder = currentRemainder + delta;**

**if (newRemainder < 0) newRemainder = 0;**

**// 4. ПРОВЕРЯЕМ СОЗДАНИЕ ЕДИНИЦ**

**let newUnits = storedUnits;**

**if (newRemainder >= BASE) {**

**// Создаём новые единицы**

**const extraUnits = Math.floor(newRemainder / BASE);**

**newUnits += extraUnits;**

**newRemainder -= extraUnits \* BASE;**

**}**

**const totalAfter = newRemainder + newUnits \* BASE;**

**const actuallyApplied = totalAfter - totalBefore;**

**// 5. СОХРАНЯЕМ НОВЫЕ ЗНАЧЕНИЯ**

**if (currentSide === 'L') {**

**L = newRemainder;**

**} else {**

**R = newRemainder;**

**}**

**setActivePV(currentElement, L, R);**

**hidden.dataset[localKey] = String(newUnits);**

**// 6. ПОДСВЕТКА (если создали единицы)**

**if (newUnits > storedUnits) {**

**highlightBalance(currentElement, currentSide);**

**}**

**// 7. НАХОДИМ РОДИТЕЛЯ**

**const currentCard = findCardByElement(currentElement);**

**const parentInfo = getParentInfo(currentCard.id);**

**// Если нет родителя или ничего не изменилось - стоп**

**if (!parentInfo || actuallyApplied === 0) {**

**break;**

**}**

**// 8. ПЕРЕХОДИМ К РОДИТЕЛЮ**

**const parentElement = findCardElementById(parentInfo.parentId);**

**if (!parentElement) break;**

**currentElement = parentElement;**

**currentSide = parentInfo.side; // сторона на родителе**

**carry = actuallyApplied; // несём то, что реально применили**

**}**

**}**

**Пример работы propagateActivePvUp**

**Структура:**

**A**

**/ \**

**/ \**

**B C**

**Начальное состояние:**

**A: Актив-PV = 0/0, units = 0/0**

**B: Актив-PV = 0/0, units = 0/0**

**C: Актив-PV = 0/0, units = 0/0**

**B - левая нога A**

**C - правая нога A**

**Действие: Пользователь нажимает "+1" 450 раз на левой стороне карточки B**

**Шаг 1: Карточка B**

**- amount = 450**

**- currentRemainder = 0**

**- storedUnits = 0**

**- newRemainder = 0 + 450 = 450**

**- 450 >= 330 → создаём 1 единицу**

**- newUnits = 0 + 1 = 1**

**- newRemainder = 450 - 330 = 120**

**- Отображаем: "120 / X"**

**- Сохраняем: locall = 1**

**- actuallyApplied = (120 + 1\*330) - 0 = 450**

**- carry = 450 → идём к родителю**

**Шаг 2: Карточка A (родитель B)**

**- B - левая нога A → currentSide = 'L'**

**- amount = 450**

**- currentRemainder = 0**

**- storedUnits = 0**

**- newRemainder = 0 + 450 = 450**

**- 450 >= 330 → создаём 1 единицу**

**- newUnits = 0 + 1 = 1**

**- newRemainder = 450 - 330 = 120**

**- Отображаем: "120 / X"**

**- Сохраняем: locall = 1**

**- actuallyApplied = 450**

**- У A нет родителя → стоп**

**Итог:**

**A: Актив-PV = 120/0, units = 1/0, Баланс L+1**

**B: Актив-PV = 120/0, units = 1/0, Баланс L+1 (от созданной единицы)**

**Кнопки управления Актив-PV**

**На каждой карточке есть кнопки:**

**Для левой ноги:**

**[+1] [+10] [-10] [-1]**

**Для правой ноги:**

**[-1] [-10] [+10] [+1]**

**Центральная кнопка:**

**[Очистить]**

**Обработка кликов:**

**canvas.addEventListener('click', (e) => {**

**const btn = e.target.closest('.active-btn');**

**if (!btn) return;**

**const cardEl = btn.closest('.card');**

**// Кнопка "Очистить"**

**if (btn.classList.contains('active-clear')) {**

**setActivePV(cardEl, 0, 0);**

**const hidden = cardEl.querySelector('.active-pv-hidden');**

**hidden.dataset.btnL = '0';**

**hidden.dataset.btnR = '0';**

**hidden.dataset.locall = '0';**

**hidden.dataset.localr = '0';**

**saveState();**

**return;**

**}**

**// Кнопки +/- 1, 10**

**const dir = btn.dataset.dir; // 'L' или 'R'**

**const rawStep = parseInt(btn.dataset.step, 10); // 1, 10, -1, -10**

**let step = rawStep;**

**// Защита от отрицательного шага больше текущего значения**

**if (step < 0) {**

**const apv = parseActivePV(cardEl);**

**const current = (dir === 'L') ? apv.L : apv.R;**

**const available = Math.max(0, current);**

**if (-step > available) {**

**step = -available; // можем убрать максимум до 0**

**}**

**}**

**if (step === 0) return;**

**// Сохраняем нажатия кнопок**

**const hidden = cardEl.querySelector('.active-pv-hidden');**

**if (dir === 'L') {**

**hidden.dataset.btnL = String(parseInt(hidden.dataset.btnL || '0', 10) + step);**

**} else {**

**hidden.dataset.btnR = String(parseInt(hidden.dataset.btnR || '0', 10) + step);**

**}**

**// Распространяем вверх**

**propagateActivePvUp(cardEl, dir, step);**

**saveState();**

**});**

**7. ФИНАЛЬНЫЙ БАЛАНС КАРТОЧКИ**

**Формула итогового баланса**

**ФинальныйБаланс = РасчётныйБаланс + БонусАктивPV + МануальныйБаланс**

**Где:**

**- РасчётныйБаланс: баллы от полных карточек (из Engine.recalc)**

**- БонусАктивPV: единицы из locall/localr**

**- МануальныйБаланс: если пользователь вручную отредактировал баланс**

**Код расчёта**

**function calculateFinalBalance(cardData, engineResult) {**

**const cardEl = cardData.element;**

**const cardId = cardData.id;**

**// 1. Расчётный баланс (от Engine)**

**const calculated = engineResult[cardId] || { L: 0, R: 0, total: 0 };**

**// 2. Бонус Актив-PV (единицы)**

**const hidden = cardEl.querySelector('.active-pv-hidden');**

**const aBonusL = hidden ? parseInt(hidden.dataset.abonusl || '0', 10) : 0;**

**const aBonusR = hidden ? parseInt(hidden.dataset.abonusr || '0', 10) : 0;**

**// 3. Локальные единицы (из Актив-PV)**

**let localL = hidden ? parseInt(hidden.dataset.locall || '0', 10) : 0;**

**let localR = hidden ? parseInt(hidden.dataset.localr || '0', 10) : 0;**

**if (!Number.isFinite(localL) || localL < 0) localL = 0;**

**if (!Number.isFinite(localR) || localR < 0) localR = 0;**

**// 4. ФИНАЛЬНЫЙ РАСЧЁТ**

**const leftBalance = Math.max(0, calculated.L + aBonusL + localL);**

**const rightBalance = Math.max(0, calculated.R + aBonusR + localR);**

**return { L: leftBalance, R: rightBalance };**

**}**

**Пример расчёта финального баланса**

**Карточка A:**

**1. Расчётный баланс (от полных карточек):**

**L = 5, R = 3**

**2. Бонус Актив-PV (устаревшее, обычно 0):**

**aBonusL = 0, aBonusR = 0**

**3. Локальные единицы (из Актив-PV):**

**locall = 2, localr = 1**

**4. Финальный баланс:**

**L = 5 + 0 + 2 = 7**

**R = 3 + 0 + 1 = 4**

**Отображение: "7 / 4"**

**8. РУЧНОЕ РЕДАКТИРОВАНИЕ БАЛАНСА**

**Концепция**

**Пользователь может вручную отредактировать поле "Баланс:" на карточке, введя произвольные значения.**

**Формат ввода**

**Допустимые форматы:**

* **"10 / 20" → L=10, R=20**
* **"10/20" → L=10, R=20**
* **"10 - 20" → L=10, R=20**
* **"произвольный текст" → сохраняется как есть (нерасчётный режим)**

**Структура данных**

**// Хранится в скрытом элементе:**

**hidden.dataset.manualBalanceLeft = "10";**

**hidden.dataset.manualBalanceRight = "20";**

**hidden.dataset.manualBalanceRaw = "10 / 20";**

**// Флаг на элементе значения:**

**valueElement.dataset.manualBalance = "true";**

**Алгоритм обработки**

**function setupBalanceManualEditing(cardEl) {**

**const rows = cardEl.querySelectorAll('.card-row');**

**rows.forEach(row => {**

**const label = row.querySelector('.label');**

**const value = row.querySelector('.value');**

**if (!label || !value) return;**

**const labelText = (label.textContent || '').trim().toLowerCase();**

**if (labelText.startsWith('баланс')) {**

**// 1. Делаем поле редактируемым**

**value.setAttribute('contenteditable', 'true');**

**// 2. При потере фокуса - обработка**

**value.addEventListener('blur', () => {**

**handleBalanceManualBlur(value);**

**});**

**}**

**});**

**}**

**function handleBalanceManualBlur(valueEl) {**

**const raw = valueEl.textContent.trim();**

**// Пытаемся распарсить как L/R**

**const match = /^(\d+)\s\*[\/\-]\s\*(\d+)$/.exec(raw);**

**const cardEl = valueEl.closest('.card');**

**const hidden = cardEl.querySelector('.active-pv-hidden');**

**if (match) {**

**// Успешно распарсили**

**const left = parseInt(match[1], 10);**

**const right = parseInt(match[2], 10);**

**if (Number.isFinite(left) && Number.isFinite(right)) {**

**// Сохраняем как числа**

**hidden.dataset.manualBalanceLeft = String(left);**

**hidden.dataset.manualBalanceRight = String(right);**

**hidden.dataset.manualBalanceRaw = '';**

**valueEl.dataset.manualBalance = 'true';**

**// Отображаем в стандартном формате**

**renderSplitValue(valueEl, left, right);**

**return;**

**}**

**}**

**// Не смогли распарсить - сохраняем как текст**

**if (raw.length > 0) {**

**hidden.dataset.manualBalanceLeft = '';**

**hidden.dataset.manualBalanceRight = '';**

**hidden.dataset.manualBalanceRaw = raw;**

**valueEl.dataset.manualBalance = 'true';**

**valueEl.textContent = raw;**

**} else {**

**// Пустое - убираем ручное редактирование**

**delete valueEl.dataset.manualBalance;**

**hidden.dataset.manualBalanceLeft = '';**

**hidden.dataset.manualBalanceRight = '';**

**hidden.dataset.manualBalanceRaw = '';**

**}**

**}**

**Приоритет отображения**

**При рендеринге баланса:**

**if (valueEl.dataset.manualBalance === 'true') {**

**// РУЧНОЙ режим - показываем то, что ввёл пользователь**

**const manual = readStoredManualBalance(cardEl);**

**if (manual.left !== null && manual.right !== null) {**

**// Числовой формат**

**renderSplitValue(valueEl, manual.left, manual.right);**

**} else {**

**// Текстовый формат**

**valueEl.textContent = manual.raw;**

**}**

**} else {**

**// АВТОМАТИЧЕСКИЙ режим - показываем рассчитанный баланс**

**const finalBalance = calculateFinalBalance(cardData, engineResult);**

**renderSplitValue(valueEl, finalBalance.L, finalBalance.R);**

**}**

**9. ОТОБРАЖЕНИЕ ЗНАЧЕНИЙ**

**Функция renderSplitValue**

**Отображает значение в формате "L / R" с отдельными span-элементами для каждой части.**

**function renderSplitValue(valueEl, left, right) {**

**// Очищаем предыдущее содержимое**

**valueEl.innerHTML = '';**

**// Создаём три элемента: левый / разделитель / правый**

**const spanL = document.createElement('span');**

**spanL.className = 'value-left value-part';**

**spanL.textContent = String(left);**

**const spanSep = document.createElement('span');**

**spanSep.className = 'value-sep';**

**spanSep.textContent = ' / ';**

**const spanR = document.createElement('span');**

**spanR.className = 'value-right value-part';**

**spanR.textContent = String(right);**

**valueEl.appendChild(spanL);**

**valueEl.appendChild(spanSep);**

**valueEl.appendChild(spanR);**

**}**

**Зачем отдельные span-элементы?**

**Это нужно для анимации подсветки при изменении значений:**

**.value-part.balance-highlight {**

**color: #d93025;**

**font-weight: 900;**

**animation: balancePulse 1.6s ease-in-out infinite alternate;**

**transform-origin: center;**

**text-shadow: 0 4px 12px rgba(217,48,37,.35);**

**}**

**@keyframes balancePulse {**

**0% { transform: scale(1.8); }**

**50% { transform: scale(2.1); }**

**100% { transform: scale(1.8); }**

**}**

**10. СИСТЕМА ПОДСВЕТКИ**

**Триггеры подсветки**

**Баланс подсвечивается, когда:**

1. **Значение увеличилось по сравнению с предыдущим расчётом**
2. **Создалась новая единица в Актив-PV**

**Линия подсвечивается, когда:**

1. **Баланс родителя увеличился через эту ногу**
2. **Актив-PV распространился через эту линию**

**Алгоритм подсветки баланса**

**function applyBalanceHighlights(highlightCandidates, idToElementMap) {**

**highlightCandidates.forEach((sides, cardId) => {**

**const cardEl = idToElementMap.get(cardId);**

**const valueEl = getBalanceValueElement(cardEl);**

**// Пропускаем ручной баланс**

**if (valueEl.dataset.manualBalance === 'true') {**

**return;**

**}**

**// Подсветка левой части**

**if (sides.left) {**

**setPartHighlight(valueEl, 'L', true, {**

**autoRemoveDuration: 2000, // автоудаление через 2 сек**

**animationDuration: 1600 // длительность анимации**

**});**

**}**

**// Подсветка правой части**

**if (sides.right) {**

**setPartHighlight(valueEl, 'R', true, {**

**autoRemoveDuration: 2000,**

**animationDuration: 1600**

**});**

**}**

**});**

**}**

**function setPartHighlight(valueEl, side, shouldHighlight, options = {}) {**

**const selector = side === 'L' ? '.value-left' : '.value-right';**

**const part = valueEl.querySelector(selector);**

**if (!part) return false;**

**if (!shouldHighlight) {**

**part.classList.remove('balance-highlight');**

**return false;**

**}**

**// Устанавливаем длительность анимации**

**const duration = options.animationDuration || 1600;**

**part.style.setProperty('--balance-animation-duration', `${duration}ms`);**

**// Перезапускаем анимацию**

**part.classList.remove('balance-highlight');**

**void part.offsetWidth; // force reflow**

**part.classList.add('balance-highlight');**

**// Таймер автоудаления**

**if (options.autoRemoveDuration > 0) {**

**setTimeout(() => {**

**part.classList.remove('balance-highlight');**

**}, options.autoRemoveDuration);**

**}**

**return true;**

**}**

**Определение кандидатов для подсветки**

**function findHighlightCandidates(previousBalances, currentBalances) {**

**const candidates = new Map();**

**currentBalances.forEach((current, cardId) => {**

**const previous = previousBalances.get(cardId);**

**if (!previous) return; // нет предыдущего значения**

**const highlight = { left: false, right: false };**

**// Проверяем увеличение**

**if (current.L > previous.L) {**

**highlight.left = true;**

**}**

**if (current.R > previous.R) {**

**highlight.right = true;**

**}**

**if (highlight.left || highlight.right) {**

**candidates.set(cardId, highlight);**

**}**

**});**

**return candidates;**

**}**

**Подсветка линий**

**function highlightLineBetween(parentId, childId, side, type) {**

**const normalizedSide = (side === 'L') ? 'left' : 'right';**

**for (const line of lines) {**

**const relation = getLineRelation(line);**

**if (!relation) continue;**

**if (relation.parent.id !== parentId) continue;**

**if (relation.child.id !== childId) continue;**

**if (relation.side !== normalizedSide) continue;**

**applyLineHighlight(line.element, type);**

**break;**

**}**

**}**

**function applyLineHighlight(lineEl, type) {**

**const className = type === 'pv' ? 'line--pv-highlight' : 'line--balance-highlight';**

**// Устанавливаем длительность анимации**

**lineEl.style.setProperty('--line-animation-duration', '1600ms');**

**// Перезапускаем анимацию**

**lineEl.classList.remove(className);**

**void lineEl.getBBox(); // force reflow для SVG**

**lineEl.classList.add(className);**

**// Таймер автоудаления**

**setTimeout(() => {**

**lineEl.classList.remove(className);**

**}, 2000);**

**}**

**11. ГЛАВНАЯ ФУНКЦИЯ ПЕРЕСЧЁТА**

**Engine.recalc - точка входа**

**Engine.recalc(state) {**

**// 1. ПОСТРОЕНИЕ ИНДЕКСА КАРТОЧЕК**

**const byId = buildIndex(state);**

**// 2. ОПРЕДЕЛЕНИЕ ИЕРАРХИИ**

**const parentOf = {}; // child -> { parentId, side }**

**const children = {}; // parent -> { left: Set, right: Set }**

**state.cards.forEach(c => {**

**children[c.id] = { left: new Set(), right: new Set() };**

**});**

**state.lines.forEach(line => {**

**const pr = pickParent(byId, line);**

**if (!pr) return;**

**const { parentId, childId, parentSide } = pr;**

**// Добавляем ребёнка к родителю**

**children[parentId][parentSide].add(childId);**

**// Для ребёнка выбираем самого верхнего родителя**

**const existed = parentOf[childId];**

**if (!existed) {**

**parentOf[childId] = { parentId, side: parentSide };**

**} else {**

**const oldP = byId.get(existed.parentId);**

**const newP = byId.get(parentId);**

**// Выбираем родителя с меньшим Y**

**if (newP && oldP && newP.y < oldP.y) {**

**parentOf[childId] = { parentId, side: parentSide };**

**}**

**}**

**});**

**// 3. ПАРСИНГ PV**

**const pv = {};**

**const isFull = {};**

**state.cards.forEach(c => {**

**const parsed = parsePv330(c.bodyHTML);**

**pv[c.id] = parsed.value;**

**isFull[c.id] = parsed.isFull;**

**});**

**// 4. РАСЧЁТ БАЛЛОВ**

**const balls = {};**

**state.cards.forEach(c => {**

**balls[c.id] = { L: 0, R: 0, total: 0 };**

**});**

**// Поднимаем баллы от полных карточек**

**state.cards.forEach(c => {**

**if (!isFull[c.id]) return;**

**let cur = c.id;**

**let iterations = 0;**

**while (parentOf[cur] && iterations < 5000) {**

**iterations++;**

**const { parentId, side } = parentOf[cur];**

**if (side === 'right') {**

**balls[parentId].R += 1;**

**} else {**

**balls[parentId].L += 1;**

**}**

**balls[parentId].total += 1;**

**cur = parentId;**

**}**

**});**

**// 5. РАСЧЁТ ЭТАПОВ И ЦИКЛОВ**

**const result = {};**

**state.cards.forEach(c => {**

**const b = balls[c.id];**

**const { cycles, stage, toNext } = calcStagesAndCycles(b.total);**

**result[c.id] = {**

**L: b.L,**

**R: b.R,**

**total: b.total,**

**cycles,**

**stage,**

**toNext**

**};**

**});**

**// 6. ВОЗВРАТ РЕЗУЛЬТАТА**

**return {**

**result, // балансы и этапы**

**meta: { // метаданные**

**pv, // PV карточек**

**isFull, // флаги полноты**

**parentOf, // иерархия child->parent**

**children // иерархия parent->children**

**}**

**};**

**}**

**12. ПОЛНЫЙ ЦИКЛ ОБНОВЛЕНИЯ**

**Последовательность действий**

**1. Пользователь изменяет данные**

**↓**

**2. saveState() вызывается**

**↓**

**3. serializeState() создаёт снимок**

**↓**

**4. undoStack.push(snapshot)**

**↓**

**5. recalculateAndRender(state)**

**↓**

**6. Engine.recalc(state)**

**├─ Определение иерархии**

**├─ Парсинг PV**

**├─ Расчёт баллов**

**└─ Расчёт этапов**

**↓**

**7. Обновление DOM**

**├─ Обновление цвета монеток**

**├─ Расчёт финального баланса**

**├─ Рендеринг значений**

**└─ Применение подсветки**

**↓**

**8. Сохранение в кеш lastCalculatedBalances**

**Код recalculateAndRender**

**function recalculateAndRender(stateOverride = null) {**

**const state = stateOverride ?? serializeState();**

**// 1. ВЫЗОВ ENGINE**

**const { result, meta } = Engine.recalc(state);**

**lastEngineMeta = meta;**

**const id2el = new Map(cards.map(c => [c.id, c.element]));**

**const highlightCandidates = new Map();**

**// 2. ОБНОВЛЕНИЕ КАЖДОЙ КАРТОЧКИ**

**cards.forEach(cd => {**

**const el = cd.element;**

**const cardResult = result[cd.id];**

**ensureActiveControls(el);**

**// 2.1. Обновление монетки**

**const circle = el.querySelector('.coin-icon circle');**

**if (circle) {**

**const full = meta.isFull[cd.id];**

**circle.setAttribute('fill', full ? '#ffd700' : '#3d85c6');**

**}**

**// 2.2. Получение бонусов Актив-PV**

**const hidden = el.querySelector('.active-pv-hidden');**

**const aBonusL = hidden ? parseInt(hidden.dataset.abonusl || '0', 10) : 0;**

**const aBonusR = hidden ? parseInt(hidden.dataset.abonusr || '0', 10) : 0;**

**// 2.3. Обновление строк**

**const rows = el.querySelectorAll('.card-row');**

**rows.forEach(row => {**

**const label = row.querySelector('.label');**

**const value = row.querySelector('.value');**

**if (!label || !value) return;**

**const name = (label.textContent || '').trim().toLowerCase();**

**if (name.startsWith('баланс')) {**

**// Проверяем ручное редактирование**

**const manual = value.dataset.manualBalance ? readStoredManualBalance(el) : null;**

**if (manual && manual.left !== null && manual.right !== null) {**

**// РУЧНОЙ режим**

**renderSplitValue(value, manual.left, manual.right);**

**} else if (manual && manual.raw) {**

**// ТЕКСТОВЫЙ режим**

**value.textContent = manual.raw;**

**} else {**

**// АВТОМАТИЧЕСКИЙ режим**

**delete value.dataset.manualBalance;**

**// Получаем локальные единицы**

**let localL = hidden ? parseInt(hidden.dataset.locall || '0', 10) : 0;**

**let localR = hidden ? parseInt(hidden.dataset.localr || '0', 10) : 0;**

**if (!Number.isFinite(localL) || localL < 0) localL = 0;**

**if (!Number.isFinite(localR) || localR < 0) localR = 0;**

**// Расчёт финального баланса**

**const r = cardResult || { L: 0, R: 0, total: 0 };**

**const leftBalance = Math.max(0, r.L + aBonusL + localL);**

**const rightBalance = Math.max(0, r.R + aBonusR + localR);**

**// Определение необходимости подсветки**

**if (cardResult) {**

**const prev = lastCalculatedBalances.get(cd.id);**

**if (prev) {**

**const prevL = prev.L;**

**const prevR = prev.R;**

**const nextL = cardResult.L;**

**const nextR = cardResult.R;**

**const entry = { left: false, right: false };**

**if (nextL > prevL) entry.left = true;**

**if (nextR > prevR) entry.right = true;**

**if (entry.left || entry.right) {**

**highlightCandidates.set(cd.id, entry);**

**}**

**}**

**}**

**// Отображение**

**renderSplitValue(value, leftBalance, rightBalance);**

**}**

**}**

**});**

**});**

**// 3. ПРИМЕНЕНИЕ ПОДСВЕТКИ**

**applyBalanceHighlights(highlightCandidates, id2el);**

**// 4. ОБНОВЛЕНИЕ КЕША**

**Object.entries(result || {}).forEach(([cardId, data]) => {**

**const nextL = data && Number.isFinite(data.L) ? data.L : 0;**

**const nextR = data && Number.isFinite(data.R) ? data.R : 0;**

**lastCalculatedBalances.set(cardId, { L: nextL, R: nextR });**

**});**

**// Очистка удалённых карточек**

**Array.from(lastCalculatedBalances.keys()).forEach(cardId => {**

**if (!result[cardId]) {**

**lastCalculatedBalances.delete(cardId);**

**}**

**});**

**}**

**13. КРАЕВЫЕ СЛУЧАИ И ОСОБЕННОСТИ**

**13.1. Множественные родители**

**Проблема: карточка может иметь несколько входящих линий.**

**Решение: выбирается родитель с наименьшим Y (самый верхний).**

**if (parentOf[childId]) {**

**const oldParent = byId.get(parentOf[childId].parentId);**

**const newParent = byId.get(parentId);**

**if (newParent.y < oldParent.y) {**

**parentOf[childId] = { parentId, side: parentSide };**

**}**

**}**

**13.2. Циклические ссылки**

**Защита: максимум 5000 итераций в цикле подъёма баллов.**

**let iterations = 0;**

**while (parentOf[cur] && iterations < 5000) {**

**iterations++;**

**// ...**

**}**

**13.3. Отрицательные значения Актив-PV**

**Защита: нельзя убрать больше, чем есть.**

**if (step < 0) {**

**const current = (dir === 'L') ? apv.L : apv.R;**

**const available = Math.max(0, current);**

**if (-step > available) {**

**step = -available;**

**}**

**}**

**13.4. Ручное редактирование баланса**

**Приоритет: ручной баланс не пересчитывается автоматически.**

**if (valueEl.dataset.manualBalance === 'true') {**

**// Показываем ручное значение**

**// НЕ запускаем автоматический расчёт**

**return;**

**}**

**13.5. Удаление карточек**

**При удалении карточки:**

1. **Удаляются все связанные линии**
2. **Пересчитывается иерархия**
3. **Баллы пересчитываются без этой карточки**

**14. ФОРМУЛЫ И КОНСТАНТЫ**

**Все формулы в одном месте**

**// БАЗОВЫЕ КОНСТАНТЫ**

**const ACTIVE\_PV\_BASE = 330;**

**const CYCLE\_BALLS = 72;**

**const STAGE\_THRESHOLDS = [6, 12, 18, 36]; // сумма = 72**

**// PV**

**PV\_полная = PV >= 330**

**PV\_неполная = PV < 330**

**// БАЛАНС РАСЧЁТНЫЙ**

**Балл\_от\_карточки = 1 // если PV >= 330**

**Балл\_от\_карточки = 0 // если PV < 330**

**Баланс\_родителя.L = сумма баллов от детей в левой ноге**

**Баланс\_родителя.R = сумма баллов от детей в правой ноге**

**Баланс\_родителя.total = L + R**

**// АКТИВ-PV**

**Отображаемое\_значение = кнопки\_нажаты % 330**

**Скрытые\_единицы = floor(кнопки\_нажаты / 330)**

**// ФИНАЛЬНЫЙ БАЛАНС**

**Финальный\_L = Расчётный\_L + Бонус\_L + Локальные\_единицы\_L**

**Финальный\_R = Расчётный\_R + Бонус\_R + Локальные\_единицы\_R**

**// ЭТАПЫ И ЦИКЛЫ**

**Циклы = floor(total\_баллов / 72)**

**Остаток = total\_баллов % 72**

**Этап\_0: 0-5 баллов (нужно 6)**

**Этап\_1: 6-17 баллов (нужно 12)**

**Этап\_2: 18-35 баллов (нужно 18)**

**Этап\_3: 36-71 баллов (нужно 36)**

**Этап\_4: 72+ баллов (новый цикл)**

**До\_следующего\_этапа = порог\_этапа - накоплено\_в\_цикле**

**15. ИТОГОВАЯ СХЕМА РАСЧЁТОВ**

**┌─────────────────────────────────────────────────────┐**

**│ ВХОДНЫЕ ДАННЫЕ │**

**├─────────────────────────────────────────────────────┤**

**│ • Карточки (x, y, bodyHTML) │**

**│ • Линии (startId, endId, startSide, endSide) │**

**│ • Актив-PV (btnL, btnR, locall, localr) │**

**└─────────────────────────────────────────────────────┘**

**↓**

**┌─────────────────────────────────────────────────────┐**

**│ ШАГ 1: ОПРЕДЕЛЕНИЕ ИЕРАРХИИ │**

**├─────────────────────────────────────────────────────┤**

**│ • Сравнение Y-координат │**

**│ • Родитель = карточка выше (меньший Y) │**

**│ • Нормализация стороны (left/right) │**

**│ • Построение parentOf и children │**

**└─────────────────────────────────────────────────────┘**

**↓**

**┌─────────────────────────────────────────────────────┐**

**│ ШАГ 2: ПАРСИНГ PV │**

**├─────────────────────────────────────────────────────┤**

**│ • Regex: /(\d+)\s\*\/\s\*330\s\*pv/i │**

**│ • pv[cardId] = число │**

**│ • isFull[cardId] = (число >= 330) │**

**└─────────────────────────────────────────────────────┘**

**↓**

**┌─────────────────────────────────────────────────────┐**

**│ ШАГ 3: РАСЧЁТ РАСЧЁТНЫХ БАЛЛОВ │**

**├─────────────────────────────────────────────────────┤**

**│ • Для каждой полной карточки: │**

**│ - Поднять 1 балл вверх по цепочке родителей │**

**│ - Добавить к соответствующей ноге │**

**│ • balls[parentId].L или .R += 1 │**

**│ • balls[parentId].total += 1 │**

**└─────────────────────────────────────────────────────┘**

**↓**

**┌─────────────────────────────────────────────────────┐**

**│ ШАГ 4: РАСЧЁТ ЭТАПОВ И ЦИКЛОВ │**

**├─────────────────────────────────────────────────────┤**

**│ • cycles = floor(total / 72) │**

**│ • within = total % 72 │**

**│ • Определение stage по thresholds [6,12,18,36] │**

**│ • Расчёт toNext до следующего этапа │**

**└─────────────────────────────────────────────────────┘**

**↓**

**┌─────────────────────────────────────────────────────┐**

**│ ШАГ 5: ДОБАВЛЕНИЕ АКТИВ-PV │**

**├─────────────────────────────────────────────────────┤**

**│ • Получение locall и localr из hidden │**

**│ • Финальный\_L = Расчётный\_L + locall │**

**│ • Финальный\_R = Расчётный\_R + localr │**

**└─────────────────────────────────────────────────────┘**

**↓**

**┌─────────────────────────────────────────────────────┐**

**│ ШАГ 6: ОТОБРАЖЕНИЕ │**

**├─────────────────────────────────────────────────────┤**

**│ • Обновление монеток (золото/синий) │**

**│ • Рендеринг баланса "L / R" │**

**│ • Определение кандидатов для подсветки │**

**│ • Применение анимации │**

**└─────────────────────────────────────────────────────┘**

**↓**

**┌─────────────────────────────────────────────────────┐**

**│ ВЫХОДНЫЕ ДАННЫЕ │**

**├─────────────────────────────────────────────────────┤**

**│ result[cardId] = { │**

**│ L: число, // левая нога │**

**│ R: число, // правая нога │**

**│ total: число, // всего баллов │**

**│ cycles: число, // циклов │**

**│ stage: 0-4, // текущий этап │**

**│ toNext: число // до следующего этапа │**

**│ } │**

**└─────────────────────────────────────────────────────┘**

**РЕЗЮМЕ**

**Система расчётов состоит из 5 взаимосвязанных подсистем:**

1. **Иерархия — определение родитель-ребёнок по Y-координатам**
2. **PV — парсинг личного объёма из HTML, флаг полноты (≥330)**
3. **Расчётные баллы — подъём 1 балла от каждой полной карточки вверх**
4. **Актив-PV — ручное управление с распространением единиц (330pv = 1 ед.)**
5. **Этапы/Циклы — прогрессия по схеме 6→12→18→36 = 72 (цикл)**

**Финальный баланс = Расчётный + Актив-единицы + Ручные правки**

**Вся математика детерминирована — одинаковые входные данные всегда дают одинаковый результат.**