// ==== main.js ====

let dropEvents = [];

const MAX\_DROPS = 20;

let floatingTexts = [];

let hoverEquip = null;

let playerAttackTimer = 0;

let monsterAttackTimers = [];

let bossAttackTimer = 0;

const ATTACK\_COOLDOWN = 1.0; // seconds between attacks

const BG\_W = 1920;

const BG\_H = 1024;

let BG\_X = 0;

let BG\_Y = 0;

let showBag = false;

const bagBtn = { x:0, y:0, w:64, h:64, hover:false };

let equipPanelX = 0, equipPanelY = 0;

// ==== Inventory (Bag) ====

const BAG\_CAPACITY = 100;

let bag = [];  // each entry: { type: 'hat'|'glove'|…, tier: n, level: m, stats: {...} }

// — Button hit-boxes for Rank-Up & Change-Name —

const rankBtn = { x: 0, y: 0, w: 64, h: 64, hover: false };

const nameBtn = { x: 0, y: 0, w: 64, h: 64, hover: false };

// auto‐sell flags for tiers 1 through 8

let autoSellTiers = Array(8).fill(false);

function generateEquipmentStats(tier) {

  const randFloat = (min, max) => Math.random() \* (max - min) + min;

  const randInt   = (min, max) => Math.floor(randFloat(min, max));

  // explicit ranges for each tier

  const ranges = {

    1: { attack:[0,10],   def:[0,10],    hp:[0,100],     regen:[0,0.5],    atkSpeed:[0.001,0.1] },

    2: { attack:[10,50],  def:[10,50],   hp:[100,500],   regen:[0.5,2.5],  atkSpeed:[0.05,0.5] },

    3: { attack:[50,200], def:[50,200],  hp:[500,2000],  regen:[2.5,10],   atkSpeed:[0.15,1.5] },

    4: { attack:[200,500],def:[200,500], hp:[2000,5000], regen:[10,25],    atkSpeed:[1.5,3]   },

    5: { attack:[500,1000],def:[500,1000],hp:[5000,10000],regen:[25,50],   atkSpeed:[3,5]     },

    6: { attack:[5000,10000],def:[5000,100000],hp:[5000,20000],regen:[500,10000],atkSpeed:[5,10]    },

    7: { attack:[200000,500000],def:[200000,500000],hp:[200000,500000],regen:[1000,2000],atkSpeed:[10,20]   },

    8: { attack:[500000,1000000],def:[500000,1000000],hp:[500000,10000000],regen:[20000,50000],atkSpeed:[20,50] }

  };

  // pick the right range object

  const r = ranges[tier] || ranges[8];

  return {

    attack:   randInt(r.attack[0],   r.attack[1]),

    def:      randInt(r.def[0],      r.def[1]),

    hp:       randInt(r.hp[0],       r.hp[1]),

    regen:    parseFloat(randFloat(r.regen[0],    r.regen[1]).toFixed(2)),

    atkSpeed: parseFloat(randFloat(r.atkSpeed[0], r.atkSpeed[1]).toFixed(2))

  };

}

function formatNumber(num) {

  return num.toLocaleString();

}

// ==== Canvas Setup ====

const canvas = document.getElementById('gameCanvas');

const ctx = canvas.getContext('2d');

let mouseX = 0, mouseY = 0;

function resizeCanvas() {

  canvas.width = window.innerWidth;

  canvas.height = window.innerHeight;

  // Center background

  BG\_X = (canvas.width - BG\_W) / 2;

  BG\_Y = (canvas.height - BG\_H) / 2;

}

window.addEventListener('resize', resizeCanvas);

canvas.addEventListener('mousemove', e => { mouseX = e.offsetX; mouseY = e.offsetY; });

resizeCanvas();

const RANKS = [

  { rank:1, name:'Novice', color:'#BBBBBB', cost: 0, statBonus:{atk:10, def:10, hp:100, regen:5} },

  { rank:2, name:'Apprentice', color:'#DDDD55', cost: 100, statBonus:{atk:20, def:20, hp:200, regen:10} },

  { rank:3, name:'Fighter', color:'#77DD77', cost: 500, statBonus:{atk:500, def:500, hp:500, regen:50} },

  { rank:4, name:'Elite', color:'#55BBBB', cost:1000, statBonus:{atk:1000,def:1000,hp:10000,regen:1000} },

  { rank:5, name:'Champion', color:'#FFAA00', cost:50000, statBonus:{atk:5000,def:5000,hp:50000,regen:5000} },

  { rank:6, name:'Master', color:'#FF5555', cost:1000000, statBonus:{atk:10000,def:10000,hp:100000,regen:10000} },

  { rank:7, name:'Legend', color:'#AA00FF', cost:50000000, statBonus:{atk:50000,def:50000,hp:500000,regen:50000} },

  { rank:8, name:'Mythic', color:'#FFD700', cost:100000000,statBonus:{atk:100000,def:100000,hp:1000000,regen:100000} }

];

let powerBalls = 0;

// ==== Assets ====

// ==== Asset Loading ====

const images = {};

images['atkSpeed'] = new Image();

images['atkSpeed'].src = 'assets/speedattack.png';

// 1) Load all your base assets

const assetNames = [

  'character', 'monster', 'boss', 'stage', 'level', 'xp',

  'attack', 'hp', 'def', 'hpregen',

  'gold', 'crystal', 'pts',

  'upgrade', 'pointUpgrade', 'add',

  'hat', 'glove', 'shoe', 'armor', 'weapon', 'upequip',

  'powerball', 'rankbutton','background', 'namechange'

];

images['addall'] = new Image();

images['addall'].src = `assets/addall.png`;

assetNames.forEach(name => {

  const img = new Image();

  img.src = `assets/${name}.png`;

  images[name] = img;

});

images['bag'] = new Image();

images['bag'].src = 'assets/bag.png';

images['sell'] = new Image();

images['sell'].src = 'assets/sell.png';

// 2) Load one character sprite per rank (1 through RANKS.length)

for (let r = 1; r <= RANKS.length; r++) {

  const key = `character\_r${r}`;               // e.g. "character\_r1"

  const img = new Image();

  img.src = `assets/character\_rank\_${r}.png`;   // e.g. "assets/character\_rank\_1.png"

  images[key] = img;

}

// ==== Notifications ====

let notifications = [];

function addNotification(text, x, y, color='black', duration=1.5) {

  notifications.push({ text, x, y, color, timer: duration });

}

// ==== Save/Load/Reset ====

// — Save Game State, including rankIndex —

// —— SAVE GAME STATE ——

function saveGame() {

  // Figure out which RANKS entry we're on

  const rankIndex = RANKS.findIndex(r => r.rank === player.rank.rank);

  // Build the state object

  const state = {

    xp,

    points,

    gold,

    crystals,

    kills,

    stage,

    playerLevel,

    xpToLevel,

    powerBalls,

    statUpgrades,

    statCosts,

    equipment,

    bag,

    player: {

  x:         player.x,

  y:         player.y,

  size:      player.size,

  speed:     player.speed,

  attack:    player.attack,

  atkSpeed:  player.atkSpeed,

  def:       player.def,

  maxHp:     player.maxHp,

  hp:        player.hp,

  regen:     player.regen,

  name:      player.name,

  rankIndex: rankIndex >= 0 ? rankIndex : 0

}

  };

  localStorage.setItem('rpgSave', JSON.stringify(state));

}

// —— LOAD GAME STATE ——

function loadGame() {

  const data = JSON.parse(localStorage.getItem('rpgSave') || '{}');

  if (!data.player) return;  // nothing to load yet

  // Restore simple globals

  xp          = data.xp          ?? 0;

  points      = data.points      ?? 0;

  gold        = data.gold        ?? 0;

  crystals    = data.crystals    ?? 0;

  powerBalls  = data.powerBalls  ?? 0;

  kills       = data.kills       ?? 0;

  stage       = data.stage       ?? 1;

  playerLevel = data.playerLevel ?? 1;

  xpToLevel   = data.xpToLevel   ?? 2;

  bag = data.bag || [];

  // Restore your stat‐upgrade trackers and costs

Object.assign(statUpgrades, data.statUpgrades ?? { attack: 0, hp: 0, def: 0, regen: 0, speed: 0 });

Object.assign(statCosts,    data.statCosts    ?? { attack: 5, hp: 8, def: 6, regen: 10, speed: 12 });

  // Restore equipment

  Object.assign(equipment, data.equipment || {});

  // Restore raw player props

  Object.assign(player, {

    x:      data.player.x    ?? (canvas.width/2 - player.size/2),

    y:      data.player.y    ?? (canvas.height/2 - player.size/2),

    size:   data.player.size ?? player.size,

    speed:  data.player.speed?? player.speed,

    name:   data.player.name ?? player.name

  });

  // Restore fully‐bumped stats

  player.attack = data.player.attack ?? player.attack;

  player.def    = data.player.def    ?? player.def;

  player.maxHp  = data.player.maxHp  ?? player.maxHp;

  player.regen  = data.player.regen  ?? player.regen;

  player.hp     = data.player.hp     ?? player.maxHp;

  player.atkSpeed = data.player.atkSpeed ?? 1;  // default if missing

  // Restore rank

  const idx = Number.isInteger(data.player.rankIndex)

            && data.player.rankIndex >= 0

            && data.player.rankIndex < RANKS.length

            ? data.player.rankIndex

            : 0;

  player.rank = RANKS[idx];

  // Finally, re‐apply everything in one go to ensure consistency

  applyEquipmentStats();

  // (This will clamp player.hp <= player.maxHp automatically)

}

function updateButtonPositions() {

  // match your upgrade / pointUpgrade icons:

  const ux = 10;

  const uy = canvas.height - 74;

  rankBtn.x = ux + 74 \* 2;   // third slot

  rankBtn.y = uy;

  nameBtn.x = ux + 74 \* 3;   // fourth slot

  nameBtn.y = uy;

    bagBtn.x  = nameBtn.x + nameBtn.w + 10; // 10px gap to the right

  bagBtn.y  = uy;

  // compute equipment‐slots panel just above the toolbar, left of bagBtn

const toolbarY = canvas.height - 74;

equipPanelX = nameBtn.x - 56;                         // 48px slot + 8px padding

equipPanelY = toolbarY - types.length \* 60 - 10;      // 60px per slot, +10px gap

}

// call it now and on every resize

window.addEventListener('resize', () => {

  resizeCanvas();

  updateButtonPositions();

});

// ==== Equipment ====

const equipment = { hat: null, glove: null, shoe: null, armor: null, weapon: null };

const types = ['hat','glove','shoe','armor','weapon'];

types.forEach(t => {

  for (let tier = 1; tier <= RANKS.length; tier++) {

    const key = `${t}t${tier}`;            // ex: "glovet3"

    const img = new Image();

    img.src   = `assets/${key}.png`;       // ex: "assets/glovet3.png"

    images[key] = img;

  }

});

// ==== Rarity ====

const tiers = [

  { tier: 1, w: 30 },

  { tier: 2, w: 15 },

  { tier: 3, w: 7.5},

  { tier: 4, w: 3.0 },

  { tier: 5, w: 0.2 },

  { tier: 6, w: 0.002 },

  { tier: 7, w: 0.0005 },

  { tier: 8, w: 0.00001 }

];

function drawBagPanel() {

  const bx = canvas.width - 200,

        by =  20,

        bw = 180,

        bh = 300;

  // panel bg

  ctx.fillStyle = 'rgba(0,0,0,0.7)';

  ctx.fillRect(bx, by, bw, bh);

  ctx.strokeStyle = '#fff';

  ctx.strokeRect(bx, by, bw, bh);

  // title

  ctx.fillStyle = '#fff';

  ctx.font      = '14px sans-serif';

  ctx.textAlign = 'left';

  ctx.fillText(`Bag (${bag.length}/${BAG\_CAPACITY})`, bx + 8, by + 24);

  // draw icons (up to 10 in a column)

  const iconSize = 48;

  bag.slice(0, 10).forEach((item, idx) => {

    const ix = bx + 8;

    const iy = by + 40 + idx \* (iconSize + 4);

    // pick the tiered sprite key, e.g. "glovet1"

    const key = `${item.type}t${item.tier}`;

    const img = images[key]?.complete ? images[key] : images[item.type];

    ctx.drawImage(img, ix, iy, iconSize, iconSize);

  });

}

const totalW = tiers.reduce((s, t) => s + t.w, 0);

function drawCharacter() {

  // pick sprite by current rank (1–8):

  const rank = player.rank.rank;

  const key  = `character\_r${rank}`;

  // fall back to default if missing:

  const img  = images[key] && images[key].complete

             ? images[key]

             : images.character;

  ctx.drawImage(img,

                player.x, player.y,

                player.size, player.size);

}

function formatCurrency(amount) {

  return amount.toLocaleString();

}

function rollTier() {

  let r = Math.random() \* totalW;

  for (const t of tiers) {

    if (r < t.w) return t.tier;

    r -= t.w;

  }

  return 1;

}

// ==== State ====

let lastTime=0;

let xp=0, points=0, gold=0, crystals=0, kills=0, stage=1, playerLevel=1;

let xpToLevel=0.0002;

const nextBossKills=5;

// ==== Player ====

const player = {

  x:     0,

  y:     0,

  size:  64,

  speed: 180,

  attack:12,

  maxHp: 100,

  hp:     100,

  def:     5,

  regen:   1,

  name:  'Chronicles',        // if you have a name field

  rank:  RANKS[0]             // ← set default to Tier 1

};

// ==== Monsters/Boss ====

const MAX\_MON=50;

let monsters=[], boss=null;

function spawnMonsters() {

  monsters = [];

  const cnt = Math.min(stage \* 4, MAX\_MON);

  for (let i = 0; i < cnt; i++) {

    const tier = rollTier();

    monsters.push({

      x: Math.random() \* (canvas.width - 48),

      y: Math.random() \* (canvas.height - 48),

      size: 48,

      speed: 100,

      maxHp: 30 + stage \* 20 + tier \* 5,

      hp: 30 + stage \* 20 + tier \* 5,

      attack: 8 + stage \* 10 + tier \* 2,

      def: tier + Math.floor(stage / 2),

      tier,

      type: types[Math.floor(Math.random() \* types.length)], // 👈 Add this line

      stats: { attack: tier, def: tier, hp: tier \* 5, regen: 0.1 \* tier }

    });

  }

    // ✅ Initialize each monster’s attack timer to 0

  monsterAttackTimers = new Array(monsters.length).fill(0);

}

function spawnBoss() {

  const tier = rollTier();

  boss = {

    x:       Math.random()\*(canvas.width - 64),

    y:       Math.random()\*(canvas.height - 64),

    size:    64,

    speed:   120,

    attack:  15 + stage \* 15 + tier \* 3,

    def:     tier + stage,                  // bosses get full‐stage def

    maxHp:   150 + stage \* 40 + tier \* 10,

    hp:      150 + stage \* 40 + tier \* 10,

    tier,

    type: types[Math.floor(Math.random() \* types.length)], // 👈 Add this line

    stats:   { attack: tier, def: tier, hp: tier\*5, regen: 0.1 \* tier }

  };

  kills = 0;

    // ✅ Initialize each monster’s attack timer to 0

  monsterAttackTimers = new Array(monsters.length).fill(0);

}

// ==== Level Up ====

function checkLevelUp(){

  if (xp >= xpToLevel){

    xp -= xpToLevel;

    playerLevel++;

    points += 10;

    xpToLevel = Math.floor(xpToLevel \* 1.5);

    // Persist immediately

    saveGame();

    addNotification(`Lv ${playerLevel}! +10 Pts`, player.x, player.y, '#00f');

  }

}

// ==== Upgrade UI ====

let showUpgrade=false, showPointUpgrade=false;

const statKeys = ['attack', 'hp', 'def', 'regen', 'atkSpeed'];

// 90% reduction

const statUpgrades = { attack:0, hp:0, def:0, regen:0, atkSpeed:0 };

const baseCosts = { attack:5, hp:8, def:6, regen:10, atkSpeed:12 };

const statCosts    = { ...baseCosts };

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

  // 1) Read mouse coords

  const mx = e.offsetX;

  const my = e.offsetY;

  // ⬇️ INSERT YOUR BAG TOGGLE HERE ⬇️

  // If you click the bag button, just open/close the bag

  // 0) Toggle bag panel

 // 0) Toggle bag open/close

if (collision(mx, my, bagBtn)) {

  showBag = !showBag;

  showUpgrade = showPointUpgrade = false;

  return;

}

// 1) All bag interactions

if (showBag) {

  const iconSize = 48, cols = 10, rows = 10, gap = 8;

  const panelW   = iconSize \* cols;

  const panelH   = iconSize \* rows + 24;

  const panelX   = bagBtn.x;

  const panelY   = bagBtn.y - panelH - gap;

  // 1a) Sell-all icon click

  const sellSize = 24,

        sellX    = panelX + panelW - sellSize - 8,

        sellY    = panelY + 4;

  if (collision(mx, my, { x: sellX, y: sellY, w: sellSize, h: sellSize })) {

    let totalG = 0;

    bag = bag.filter(it => {

      if (autoSellTiers[it.tier - 1]) {

        totalG += it.tier \* 5;

        return false;

      }

      return true;

    });

    gold += totalG;

    recordDrop({ text: `Auto-sold for ${totalG} G`, color: '#fc0' });

    showBag = false;

    return;

  }

  // 1b) Tier-checkbox toggle

  const headerY = panelY + 4, headerH = 16;

  if (

    my >= headerY && my <= headerY + headerH &&

    mx >= panelX && mx <= panelX + panelW

  ) {

    const clicked = Math.floor((mx - panelX) / (panelW / 8)) + 1;

    if (clicked >= 1 && clicked <= 8) {

      autoSellTiers[clicked - 1] = !autoSellTiers[clicked - 1];

      addNotification(

        `Auto-sell T${clicked}: ${autoSellTiers[clicked - 1] ? 'ON' : 'OFF'}`,

        player.x, player.y - 20, '#fff', 1.2

      );

    }

    return;

  }

  // 1c) Equip on grid-cell click

  const gridY = panelY + 24, gridH = iconSize \* rows;

  if (collision(mx, my, { x: panelX, y: gridY, w: panelW, h: gridH })) {

    const col = Math.floor((mx - panelX) / iconSize),

          row = Math.floor((my - gridY) / iconSize),

          idx = row \* cols + col;

    if (idx >= 0 && idx < bag.length) {

      const item = bag[idx];

      const eq   = equipment[item.type];

      // preserve upgrade level if same tier

      if (eq && eq.tier === item.tier) {

        item.level = eq.level;

        item.stats = { ...eq.stats };

      }

      // send old gear back to bag

      if (eq) {

        bag.push({

          type:      eq.type || item.type,

          tier:      eq.tier,

          level:     eq.level,

          baseStats: eq.baseStats,

          stats:     eq.stats

        });

      }

      // equip new

      equipment[item.type] = item;

      bag.splice(idx, 1);

      applyEquipmentStats();

      addNotification(

        `Equipped ${item.type} (T${item.tier})`,

        player.x, player.y - 20, '#0f0', 1.5

      );

    }

    showBag = false;

    return;

  }

  // 1d) click anywhere else in bag panel: consume

  return;

}

  // 2) Rank-Up button

if (collision(mx, my, rankBtn)) {

  const idx  = player.rank.rank - 1;

  const next = RANKS[idx + 1];

  if (next) {

    const cost   = next.cost;

    const have   = powerBalls;

    if (have >= cost) {

      // --- perform the rank up ---

      powerBalls    -= cost;

      player.rank    = next;

      player.attack += next.statBonus.atk;

      player.def    += next.statBonus.def;

      player.maxHp  += next.statBonus.hp;

      player.regen  += next.statBonus.regen;

      // floating popup

      addNotification(`Ranked Up to ${next.name}!`, player.x, player.y - 20, next.color, 2);

      // record in dropEvents

      recordDrop({ text: `Ranked Up: ${next.name}`, color: next.color });

      if (dropEvents.length > MAX\_DROPS) dropEvents.length = MAX\_DROPS;

      saveGame();

    } else {

      // --- not enough Power Balls ---

      const need = cost - have;

      const msg  = `Need ${formatNumber(need)} Power Ball${need > 1 ? 's' : ''}`;

      // floating popup

      addNotification(msg, player.x, player.y - 20, 'red', 1.5);

      // record in dropEvents

      recordDrop({ text: msg, color: 'red' });

      if (dropEvents.length > MAX\_DROPS) dropEvents.length = MAX\_DROPS;

    }

  } else {

    // already at max rank

    const msg = `Max Rank: ${player.rank.name}`;

    addNotification(msg, player.x, player.y - 20, player.rank.color, 1.5);

    recordDrop({ text: msg, color: player.rank.color });

    if (dropEvents.length > MAX\_DROPS) dropEvents.length = MAX\_DROPS;

  }

  return;

}

// 2) Name-Change button

if (collision(mx, my, nameBtn)) {

  const newName = prompt('Enter your new name:', player.name);

  if (newName && newName.trim()) {

    player.name = newName.trim();

    saveGame();

    addNotification(

      `Name changed to ${player.name}`,

      player.x, player.y - 20,

      'white', 2

    );

  }

  return;

}

  // 3) Bag-Panel Clicks

const bx = canvas.width - 200, by = 20, bw = 180, bh = 300;

if (collision(mx, my, { x: bx, y: by, w: bw, h: bh })) {

  const relY = my - (by + 40);

  const idx  = Math.floor(relY / 24);

  if (idx >= 0 && idx < bag.length && idx < 10) {

    const item = bag[idx];

    // Push old equipment \*with\* its type back into the bag:

    if (equipment[item.type]) {

      const old = equipment[item.type];

      bag.push({

        type:  item.type,

        tier:  old.tier,

        level: old.level,

        stats: old.stats

      });

    }

    // Equip the selected bag item:

    equipment[item.type] = item;

    bag.splice(idx, 1);

    applyEquipmentStats();

    addNotification(

      `Equipped ${item.type} T${item.tier}`,

      player.x, player.y - 20,

      '#0f0'

    );

  }

  return;

}

  // 4) Toggle Gold/Point Upgrade panels

  const ux = 10;

  const uy = canvas.height - 74;

  if (mx >= ux && mx < ux + 64 && my >= uy && my < uy + 64) {

    showUpgrade = !showUpgrade;

    showPointUpgrade = false;

    return;

  }

  if (mx >= ux + 74 && mx < ux + 138 && my >= uy && my < uy + 64) {

    showPointUpgrade = !showPointUpgrade;

    showUpgrade = false;

    return;

  }

  // 5) Gold→Stat upgrades

  if (showUpgrade) {

    handleUpgradeClick(mx, my);

    return;

  }

  // 6) Point→Stat upgrades

  if (showPointUpgrade) {

    handlePointUpgradeClick(mx, my);

    return;

  }

  // 7) Equipment upgrades

// 7) Equipment upgrades

types.forEach((t, i) => {

  const upx = 10 + 50;

  const upy = 10 + i \* 60 + 16;

  if (collision(mx, my, { x: upx, y: upy, w: 24, h: 24 })) {

    const eq = equipment[t];

    if (!eq) return;

    const costG = eq.level \* 10;

    const costC = eq.level \* 2;

    if (gold >= costG && crystals >= costC) {

      // — you can afford it —

      gold     -= costG;

      crystals -= costC;

      eq.level++;

      eq.stats.attack += eq.tier;

      eq.stats.def    += eq.tier;

      eq.stats.hp     += eq.tier \* 5;

      eq.stats.regen  += 0.1 \* eq.tier;

      eq.stats.atkSpeed = (eq.stats.atkSpeed || 0) + 0.005 \* eq.tier;

      addNotification(`+${t} L${eq.level}`, upx, upy, '#0f0');

      applyEquipmentStats();

      saveGame();

    } else {

      // — not enough, compute missing amounts —

      const needG = Math.max(0, costG - gold);

      const needC = Math.max(0, costC - crystals);

      const msg   = `Need ${formatNumber(needG)}G & ${formatNumber(needC)}C`;

      // 1) floating notification

      addNotification(msg, player.x, player.y - 20, 'red', 1.5);

      // 2) record in your dropEvents for the drop‐tracker

      recordDrop({ text: msg, color: 'red' });

      if (dropEvents.length > MAX\_DROPS) dropEvents.length = MAX\_DROPS;

    }

  }

});

});

function recordDrop(evt) {

  dropEvents.unshift(evt);

  if (dropEvents.length > MAX\_DROPS) dropEvents.length = MAX\_DROPS;

  updateDropList();

}

// ==== Stat‐for‐Gold Upgrade Click Handler ====

function handleUpgradeClick(mx, my) {

  const upgradeStats = ['attack', 'hp', 'def', 'regen', 'atkSpeed'];

  const panelHeight = 16 + upgradeStats.length \* 40 + 16;

  const by = canvas.height - 74 - panelHeight - 10;

  const bx = PANEL\_X;

  const w  = 32, h = 32;

  upgradeStats.forEach((k, i) => {

    const y       = by + 16 + i \* 40;

    const singleX = bx + 16 + 32 + 10;

    const allX    = bx + 16 + 32 \* 2 + 20;

    const cost    = statCosts[k];

    if (typeof cost !== 'number' || isNaN(cost)) return;

    if (collision(mx, my, { x: singleX, y, w, h })) {

      if (gold >= cost) {

        gold -= cost;

        if (!statUpgrades[k]) statUpgrades[k] = 0;

        statUpgrades[k] += (k === 'hp') ? 10 : (k === 'regen') ? 0.5 : (k === 'atkSpeed') ? 0.005 : 1;

        statCosts[k] += 50;

        addNotification(`+1 ${k}`, singleX, y, '#0f0');

      recordDrop({ text: `+1 ${k}`, color: '#0f0' });

        if (dropEvents.length > MAX\_DROPS) dropEvents.length = MAX\_DROPS;

        applyEquipmentStats(); saveGame();

      } else {

        const need = cost - gold;

        const msg  = `Need ${formatNumber(need)}G`;

        addNotification(msg, player.x, player.y - 20, 'red', 1.5);

        recordDrop({ text: msg, color: 'red' });

        if (dropEvents.length > MAX\_DROPS) dropEvents.length = MAX\_DROPS;

      }

    }

    if (collision(mx, my, { x: allX, y, w, h })) {

      let purchases = 0;

      while (gold >= statCosts[k]) {

        const c = statCosts[k];

        gold -= c;

        if (!statUpgrades[k]) statUpgrades[k] = 0;

        statUpgrades[k] += (k === 'hp') ? 10 : (k === 'regen') ? 0.5 : (k === 'atkSpeed') ? 0.005 : 1;

        statCosts[k] += 50;

        purchases++;

      }

      if (purchases > 0) {

        const msg = `+${purchases} ${k}`;

        addNotification(msg, allX, y, '#0f0');

        recordDrop({ text: msg, color: '#0f0' });

        if (dropEvents.length > MAX\_DROPS) dropEvents.length = MAX\_DROPS;

        applyEquipmentStats(); saveGame();

      } else {

        const need = statCosts[k] - gold;

        const msg  = `Need ${formatNumber(need)}G`;

        addNotification(msg, player.x, player.y - 20, 'red', 1.5);

        recordDrop({ text: msg, color: 'red' });

        if (dropEvents.length > MAX\_DROPS) dropEvents.length = MAX\_DROPS;

      }

    }

  });

}

// Add speed to statKeys if not already done

if (!statKeys.includes('speed')) statKeys.push('speed');

// Ensure speed icon is loaded (expected to be named speedattack.png)

if (!images.speed) {

  const img = new Image();

  img.src = 'assets/speedattack.png';

  images.speed = img;

}

// ==== Stat‐for‐Points Upgrade Click Handler ====

function handlePointUpgradeClick(mx, my) {

  const upgradeStats = ['attack', 'hp', 'def', 'regen', 'atkSpeed'];

  const panelHeight = 16 + upgradeStats.length \* 40 + 16;

  const by = canvas.height - 74 - panelHeight - 10;

  const bx = PANEL\_X;

  const w  = 32, h = 32;

  upgradeStats.forEach((k, i) => {

    const y       = by + 16 + i \* 40;

    const singleX = bx + 16 + 32 + 10;

    const allX    = bx + 16 + 32 \* 2 + 20;

    if (collision(mx, my, { x: singleX, y, w, h })) {

      if (points > 0) {

        points--;

        if (!statUpgrades[k]) statUpgrades[k] = 0;

        statUpgrades[k] += (k === 'hp') ? 10 : (k === 'regen') ? 0.5 : (k === 'atkSpeed') ? 0.0005 : 1;

        addNotification(`+1 ${k}`, singleX, y, '#09f');

        recordDrop({ text: `+1 ${k}`, color: '#09f' });

        if (dropEvents.length > MAX\_DROPS) dropEvents.length = MAX\_DROPS;

        applyEquipmentStats(); saveGame();

      } else {

        const msg = 'Need 1 Pt';

        addNotification(msg, player.x, player.y - 20, 'red', 1.5);

        recordDrop({ text: msg, color: 'red' });

        if (dropEvents.length > MAX\_DROPS) dropEvents.length = MAX\_DROPS;

      }

    }

    if (collision(mx, my, { x: allX, y, w, h })) {

      if (points > 0) {

        const times = points;

        points = 0;

        if (!statUpgrades[k]) statUpgrades[k] = 0;

        statUpgrades[k] += (k === 'hp') ? 10 \* times : (k === 'regen') ? 0.5 \* times : (k === 'atkSpeed') ? 0.0005 \* times : times;

        const msg = `+${times} ${k}`;

        addNotification(msg, allX, y, '#09f');

        recordDrop({ text: msg, color: '#09f' });

        if (dropEvents.length > MAX\_DROPS) dropEvents.length = MAX\_DROPS;

        applyEquipmentStats(); saveGame();

      } else {

        const msg = 'Need 1 Pt';

        addNotification(msg, player.x, player.y - 20, 'red', 1.5);

        recordDrop({ text: msg, color: 'red' });

        if (dropEvents.length > MAX\_DROPS) dropEvents.length = MAX\_DROPS;

      }

    }

  });

}

// — Helper to test cursor over a rectangle —

function collision(mx, my, rect) {

  return mx >= rect.x &&

         mx <= rect.x + rect.w &&

         my >= rect.y &&

         my <= rect.y + rect.h;

}

// ==== Hover Tooltip ====

canvas.addEventListener('mousemove', e => {

  mouseX = e.offsetX;

  mouseY = e.offsetY;

 // equipment tooltip logic…

  hoverEquip = types.find((t, i) =>

    collision(mouseX, mouseY, { x: 10, y: 10 + i \* 60, w: 48, h: 48 })

  ) || null;

  // rank/name button hover

  rankBtn.hover = collision(mouseX, mouseY, rankBtn);

  nameBtn.hover = collision(mouseX, mouseY, nameBtn);

});

// ==== Main Loop ====

function initGame(){

  loadGame();

  spawnMonsters();

  player.x = canvas.width/2 - player.size/2;

  player.y = canvas.height/2 - player.size/2;

  updateButtonPositions();    // <-- position the rank/name buttons

  lastTime = performance.now();

// window.setInterval(saveGame, 15000);  // ← disabled autosave

  requestAnimationFrame(gameLoop);

}

function gameLoop(ts){

  const dt=(ts-lastTime)/1000; lastTime=ts;

  update(dt); draw();

  notifications=notifications.filter(n=>{n.timer-=dt;return n.timer>0;});

  requestAnimationFrame(gameLoop);

}

function inMeleeRange(a, b) {

  const dx = b.x - a.x;

  const dy = b.y - a.y;

  const dist = Math.hypot(dx, dy);

  return dist <= (a.size + b.size) / 2 + 4;

}

// ==== Update ====

function update(dt) {

  if (monsters.length === 0 && !boss) {

    if (kills === 0) {

      spawnMonsters();

    } else {

      spawnBoss();

    }

  }

  [player, ...monsters, boss].filter(e => e).forEach(e => {

    const target = e === player ? (boss || monsters[0] || null) : player;

    if (!target) return;

    const dx = target.x - e.x;

    const dy = target.y - e.y;

    const d = Math.hypot(dx, dy);

    const minDist = (e.size + target.size) / 2 + 4;

    if (d > minDist) {

      e.x += dx / d \* e.speed \* dt;

      e.y += dy / d \* e.speed \* dt;

    }

    if (e === player) clampPlayer();

    else {

      e.x = Math.max(0, Math.min(canvas.width - e.size, e.x));

      e.y = Math.max(0, Math.min(canvas.height - e.size, e.y));

    }

  });

  monsters.forEach((m, i) => {

    if (inMeleeRange(player, m)) {

      if (playerAttackTimer <= 0 && player.atkSpeed > 0) {

        const dmg = Math.max(0, player.attack - m.def);

        m.hp -= dmg;

        m.flash = 0.1;

        addNotification(`-${Math.floor(dmg)}`, m.x, m.y, 'red');

       playerAttackTimer = 1 / player.atkSpeed;

      }

      if (monsterAttackTimers[i] <= 0) {

        const dmg = Math.max(0, m.attack - player.def);

        player.hp -= dmg;

        monsterAttackTimers[i] = ATTACK\_COOLDOWN;

      }

    }

if (m.hp <= 0) {

  // 1) XP / Gold / Crystal rewards

  const rewardXP       = 10000 \* stage;

  const rewardGold     = 10000 \* stage;

  const rewardCrystals = 1;

  xp       += rewardXP;

  gold     += rewardGold;

  crystals += rewardCrystals;

  kills++;

  recordDrop({ text: `+${formatNumber(rewardXP)} XP`, color: '#00f' });

  recordDrop({ text: `+${formatNumber(rewardGold)} G`, color: '#fc0' });

  recordDrop({ text: `+${formatNumber(rewardCrystals)} C`, color: '#0cf' });

  // 2) Chance for Power Ball (always 100% here; adjust if desired)

  if (Math.random() < 1) {

    powerBalls++;

    recordDrop({ text: '+1 Power Ball', color: 'cyan' });

  }

  // 3) Compute drop probability from your tiers[] weights

  const slot = m.type;

  const tier = m.tier;

  const wObj = tiers.find(o => o.tier === tier) || { w: 0 };

  const dropProb = wObj.w / totalW;

  // 4) Roll to see if gear actually drops

  if (Math.random() < dropProb) {

    recordDrop({ text: `Dropped ${slot} (T${tier})`, color: tierColor(tier) });

    // 5) Auto-sell vs. bag logic

    if (autoSellTiers[tier - 1]) {

      const sellVal = tier \* 5;

      gold += sellVal;

      recordDrop({ text: `Auto-sold T${tier} for ${sellVal} G`, color: '#fc0' });

    } else if (bag.length < BAG\_CAPACITY) {

      const base = generateEquipmentStats(tier);

// 1b) bag it with both copies

bag.push({

  type:      slot,

  tier:      tier,

  level:     1,

  baseStats: {...base},   // keep the original

  stats:     {...base}    // this will get mutated by upgrades

});

      recordDrop({ text: `Bagged ${slot} (T${tier})`, color: tierColor(tier) });

    } else {

      const sellVal = tier \* 5;

      gold += sellVal;

      recordDrop({ text: `Sold ${sellVal} G (bag full)`, color: '#fc0' });

    }

  } else {

    // 6) No gear dropped this time

    recordDrop({ text: 'No gear dropped', color: '#888' });

  }

  // 7) Cleanup

  if (dropEvents.length > MAX\_DROPS) dropEvents.length = MAX\_DROPS;

  saveGame();

  monsters.splice(i, 1);

  monsterAttackTimers.splice(i, 1);

}

  });

  if (boss && inMeleeRange(player, boss)) {

    if (playerAttackTimer <= 0) {

      const dmg = Math.max(0, player.attack - boss.def);

      boss.hp -= dmg;

      boss.flash = 0.1;

      addNotification(`-${Math.floor(dmg)}`, boss.x, boss.y, 'red');

      playerAttackTimer = ATTACK\_COOLDOWN;

    }

    if (bossAttackTimer <= 0) {

      const dmg = Math.max(0, boss.attack - player.def);

      player.hp -= dmg;

      bossAttackTimer = ATTACK\_COOLDOWN;

    }

   if (boss.hp <= 0) {

  // 1) XP / Gold / Crystal rewards

  const rewardXP       = 20000 \* stage;

  const rewardGold     = 20000 \* stage;

  const rewardCrystals = 5;

  xp       += rewardXP;

  gold     += rewardGold;

  crystals += rewardCrystals;

  recordDrop({ text: `+${formatNumber(rewardXP)} XP`, color: '#00f' });

  recordDrop({ text: `+${formatNumber(rewardGold)} G`, color: '#fc0' });

  recordDrop({ text: `+${formatNumber(rewardCrystals)} C`, color: '#0cf' });

  // 2) Chance for Power Ball

  if (Math.random() < 0.10) {

    powerBalls++;

    recordDrop({ text: '+1 Power Ball', color: 'cyan' });

  }

  // 3) Compute drop probability

  const slot = boss.type;

  const tier = boss.tier;

  const wObj = tiers.find(o => o.tier === tier) || { w: 0 };

  const dropProb = wObj.w / totalW;

  // 4) Attempt drop

  if (Math.random() < dropProb) {

    recordDrop({ text: `Dropped ${slot} (T${tier})`, color: tierColor(tier) });

    // 5) Auto-sell vs. bag

    if (autoSellTiers[tier - 1]) {

      const sellVal = tier \* 5;

      gold += sellVal;

      recordDrop({ text: `Auto-sold T${tier} for ${sellVal} G`, color: '#fc0' });

    }

    else if (bag.length < BAG\_CAPACITY) {

      // roll base once

      const base = generateEquipmentStats(tier);

      bag.push({

        type:      slot,

        tier:      tier,

        level:     1,

        baseStats: { ...base },

        stats:     { ...base }

      });

      recordDrop({ text: `Bagged ${slot} (T${tier})`, color: tierColor(tier) });

    }

    else {

      const sellVal = tier \* 5;

      gold += sellVal;

      recordDrop({ text: `Sold ${sellVal} G (bag full)`, color: '#fc0' });

    }

  }

  else {

    recordDrop({ text: 'No gear dropped', color: '#888' });

  }

  // 6) Cleanup & reset

  if (dropEvents.length > MAX\_DROPS) dropEvents.length = MAX\_DROPS;

  addNotification('Boss down', boss.x, boss.y + 40, '#0f0');

  stage++;

  kills  = 0;

  boss   = null;

  monsters = [];

}

  }

  if (player.hp <= 0) {

    player.hp = player.maxHp;

    stage = Math.max(1, stage - 1);

    addNotification('Stage Failed', player.x, player.y, 'red');

    monsters = [];

    monsterAttackTimers = [];

    boss = null;

    kills = 0;

    spawnMonsters();

    player.x = canvas.width / 2 - player.size / 2;

    player.y = canvas.height / 2 - player.size / 2;

  }

  player.hp = Math.min(player.maxHp, player.hp + player.regen \* dt);

  playerAttackTimer = Math.max(0, playerAttackTimer - dt);

  bossAttackTimer = Math.max(0, bossAttackTimer - dt);

  monsterAttackTimers = monsterAttackTimers.map(t => Math.max(0, t - dt));

  monsters.forEach(m => { if (m.flash > 0) m.flash -= dt; });

  if (boss && boss.flash > 0) boss.flash -= dt;

  checkLevelUp();

}

// — Draw any floating text effects (e.g. damage pop‐ups) —

function drawFloatingTexts() {

  ctx.font      = '20px "Press Start 2P"';

  ctx.textAlign = 'center';

  floatingTexts.forEach((ft, i) => {

    ctx.globalAlpha = ft.alpha;

    ctx.fillStyle   = ft.color;

    ctx.fillText(ft.text, ft.x, ft.y);

    ft.y     += ft.dy;

    ft.alpha -= 0.01;

    if (ft.alpha <= 0) floatingTexts.splice(i, 1);

  });

  ctx.globalAlpha = 1;

}

function addNotification(text, x, y, color='black', duration=1.5) {

  notifications.push({ text, x, y, color, timer: duration });

  floatingTexts.push({ text, x, y, color, alpha: 1, dy: -1 });

}

// — Draw on‐screen notifications queued by addNotification() —

function drawNotifications() {

  ctx.font      = '14px "Press Start 2P"';

  ctx.textAlign = 'center';

  notifications.forEach(n => {

    ctx.globalAlpha = Math.min(1, n.timer / 1.5);

    ctx.fillStyle   = n.color;

    ctx.fillText(n.text, n.x, n.y);

  });

  ctx.globalAlpha = 1;

}

// panel width & height

const PANEL\_W = 350;

const PANEL\_H = 200;

// toolbar Y

const TOOLBAR\_Y = canvas.height - 74;

// computed panel X/Y

const PANEL\_X = 10;

const PANEL\_Y = TOOLBAR\_Y - PANEL\_H - 8;  // 8px gap above buttons

// — Draw the gold‐for‐stats upgrade panel —

function drawUpgradePanel() {

  const upgradeStats = ['attack', 'hp', 'def', 'regen', 'atkSpeed'];

  const panelHeight = 16 + upgradeStats.length \* 40 + 16;

  const panelY = canvas.height - 74 - panelHeight - 10; // move above the buttons

  ctx.fillStyle = 'rgba(255,255,255,0.9)';

  ctx.fillRect(PANEL\_X, panelY, PANEL\_W, panelHeight);

  ctx.strokeStyle = '#000';

  ctx.strokeRect(PANEL\_X, panelY, PANEL\_W, panelHeight);

  upgradeStats.forEach((k, i) => {

    const y      = panelY + 16 + i \* 40;

    const icon   = k === 'regen' ? 'hpregen' : k;

    const cost   = statCosts[k];

    ctx.drawImage(images[icon],      PANEL\_X + 16,          y, 32, 32);

    ctx.drawImage(images.add,        PANEL\_X + 16 + 32 + 10, y, 32, 32);

    ctx.drawImage(images.addall,     PANEL\_X + 16 + 32\*2 + 20, y, 32, 32);

    ctx.drawImage(images.gold,       PANEL\_X + 16 + 32\*3 + 30, y + 8, 16, 16);

    ctx.fillStyle = '#000';

    ctx.font = '18px "Press Start 2P"';

    ctx.textBaseline = 'middle';

    ctx.fillText(formatNumber(cost), PANEL\_X + 16 + 32\*3 + 70, y + 16);

  });

}

// — Draw the point‐for‐stats upgrade panel —

function drawPointUpgradePanel() {

  const upgradeStats = ['attack', 'hp', 'def', 'regen', 'atkSpeed'];

  const panelHeight = 16 + upgradeStats.length \* 40 + 16;

  const panelY = canvas.height - 74 - panelHeight - 10; // move above the buttons

  ctx.fillStyle = 'rgba(255,255,255,0.9)';

  ctx.fillRect(PANEL\_X, panelY, PANEL\_W, panelHeight);

  ctx.strokeStyle = '#000';

  ctx.strokeRect(PANEL\_X, panelY, PANEL\_W, panelHeight);

  upgradeStats.forEach((k, i) => {

    const y      = panelY + 16 + i \* 40;

    const icon   = k === 'regen' ? 'hpregen' : k;

    ctx.drawImage(images[icon],      PANEL\_X + 16,          y, 32, 32);

    ctx.drawImage(images.add,        PANEL\_X + 16 + 32 + 10, y, 32, 32);

    ctx.drawImage(images.addall,     PANEL\_X + 16 + 32\*2 + 20, y, 32, 32);

    ctx.drawImage(images.pts,        PANEL\_X + 16 + 32\*3 + 30, y + 8, 16, 16);

    ctx.fillStyle = '#000';

    ctx.font = '18px "Press Start 2P"';

    ctx.textBaseline = 'middle';

    ctx.fillText(formatNumber(points), PANEL\_X + 16 + 32\*3 + 70, y + 16);

  });

}

// — Full Reset Function —

// —— Global trackers (near the top of your file) ——

// —— The reset helper ——

function resetGame() {

  // 1) Remove any saved data

  localStorage.removeItem('rpgSave');

  // 2) Reset progression & currencies

  xp          = 0;

  points      = 0;

  gold        = 0;

  crystals    = 0;

  powerBalls  = 0;

  kills       = 0;

  stage       = 1;

  playerLevel = 1;

  xpToLevel   = 2;

  // 3) Reset player to base stats & centered position

  Object.assign(player, {

    x:      canvas.width/2 - player.size/2,

    y:      canvas.height/2 - player.size/2,

    attack: 12,

    def:    5,

    maxHp:  100,

    hp:     100,

    regen:  1,

    atkSpeed: 1,     // reset attack‐speed to base

    speed:    180,   // reset movement speed to base

    name:   'Chronicles',

    rank:   RANKS[0]

  });

  // 4) Clear all equipment slots & bag

  types.forEach(t => {

    equipment[t] = null;

  });

  bag = [];

  // 5) Zero out purchased stats & restore upgrade costs,

  //    including atkSpeed if you let players upgrade it

  Object.assign(statUpgrades, { attack:0, hp:0, def:0, regen:0, atkSpeed:0 });

  Object.assign(statCosts,    baseCosts);

  // 6) Clear auto‐sell flags

  autoSellTiers = Array(8).fill(false);

  // 7) Clear UI trackers

  dropEvents.length    = 0;

  notifications.length = 0;

  floatingTexts.length = 0;

  showUpgrade          = false;

  showPointUpgrade     = false;

  showBag              = false;

  // 8) Rebuild the player’s derived stats (base + rank only)

  applyEquipmentStats();

  // 9) Respawn the world so you can start fresh

  boss = null;

  monsters = [];

  spawnMonsters();

  // 10) Persist this fresh state

  saveGame();

}

// Shortcut: press “T” to reset

window.addEventListener('keydown', e => {

  if (e.code === 'KeyT') {

    resetGame();

  }

});

function drawStatPanel() {

  const ix        = 10;   // left edge of the panel

  const iy        = 300;    // top edge of the panel

  const padding   = 10;    // panel padding

  const iconSize  = 24;    // icon width & height

  const textGap   = 8;     // space between icon and text

  const rowHeight = 32;    // vertical space per row

  const panelWidth  = 260; // width of the panel

  // your stats array:

const stats = [

  ['stage',     stage],

  ['level',     playerLevel],

  ['xp',        `${xp}/${xpToLevel}`],

  ['attack',    player.attack],

  ['hp',        `${Math.floor(player.hp)}/${player.maxHp}`],

  ['def',       player.def],

  ['regen',     player.regen.toFixed(1)],

  ['atkSpeed',  player.atkSpeed.toFixed(2)],

  ['gold',      gold],

  ['crystal',   crystals],

  ['pts',       points],

  ['powerball', powerBalls]

];

  // compute panel height

  const panelHeight = padding \* 2 + stats.length \* rowHeight;

  // background

  ctx.fillStyle = 'rgba(0,0,0,0.6)';

  ctx.fillRect(ix, iy, panelWidth, panelHeight);

  ctx.strokeStyle = '#fff';

  ctx.strokeRect(ix, iy, panelWidth, panelHeight);

  // text style

  ctx.font         = '16px "Press Start 2P"';

  ctx.fillStyle    = '#fff';

  ctx.textBaseline = 'middle';

  ctx.textAlign    = 'left';

  // draw each row

  stats.forEach((s, i) => {

    const xIcon = ix + padding;

    const yIcon = iy + padding + i \* rowHeight;

    const xText = xIcon + iconSize + textGap;

    const yText = yIcon + iconSize / 2;

    // pick icon (regen→hpregen)

    const key = s[0] === 'regen' ? 'hpregen' : s[0];

    const img = images[key];

    if (img && img.complete) {

      ctx.drawImage(img, xIcon, yIcon, iconSize, iconSize);

    }

    // draw the value to the right

    ctx.fillText(s[1], xText, yText);

  });

}

// — Draw the “Last Drops” panel —

// DEBUG: draw button outlines

ctx.strokeStyle = 'magenta';

ctx.lineWidth = 2;

ctx.strokeRect(rankBtn.x, rankBtn.y, rankBtn.w, rankBtn.h);

ctx.strokeRect(nameBtn.x, nameBtn.y, nameBtn.w, nameBtn.h);

function draw() {

 ctx.clearRect(0, 0, canvas.width, canvas.height);

if (images.background && images.background.complete) {

  ctx.drawImage(images.background, BG\_X, BG\_Y, BG\_W, BG\_H);

}

  // === Draw Monsters ===

 monsters.forEach(m => {

  drawHpBar(m);

  // ✅ Attack flash effect

  if (m.flash > 0) {

    ctx.globalAlpha = 1;

    ctx.drawImage(images.monster, m.x, m.y, m.size, m.size);

    ctx.fillStyle = 'rgba(255,255,255,0.6)';

    ctx.fillRect(m.x, m.y, m.size, m.size);

  } else {

    ctx.drawImage(images.monster, m.x, m.y, m.size, m.size);

  }

});

  // === Draw Boss ===

  if (boss) {

  drawHpBar(boss);

  if (boss.flash > 0) {

    ctx.globalAlpha = 1;

    ctx.drawImage(images.boss, boss.x, boss.y, boss.size, boss.size);

    ctx.fillStyle = 'rgba(255,255,255,0.6)';

    ctx.fillRect(boss.x, boss.y, boss.size, boss.size);

  } else {

    ctx.drawImage(images.boss, boss.x, boss.y, boss.size, boss.size);

  }

}

  // === Draw Player ===

  drawHpBar(player);

- ctx.drawImage(images.character, player.x, player.y, player.size, player.size);

+ drawCharacter();

   // === Display Rank & Name Above Player ===

// after drawHpBar(player) and drawing the character sprite:

ctx.font         = '12px "Press Start 2P"';

ctx.textBaseline = 'middle';

const rankText = `[ ${player.rank.name} ]`;   // now “Tier 1”, “Tier 2”, etc.

const nameText = ` ${player.name}`;

// Measure their widths

const rankW = ctx.measureText(rankText).width;

const nameW = ctx.measureText(nameText).width;

const totalW = rankW + nameW;

// Compute start X so the entire combo is centered

const startX = player.x + player.size/2 - totalW/2;

// Y position: a bit above the HP bar (HP bar is at player.y -10)

const yLine = player.y - 18;

// Draw the rank in its color

ctx.textAlign = 'left';

ctx.fillStyle = player.rank.color;

ctx.fillText(rankText, startX, yLine);

// Draw the name in black immediately after

ctx.fillStyle = 'black';

ctx.fillText(nameText, startX + rankW, yLine);

  // === Draw Rank-Up Button & Tooltip ===

  if (images.rankbutton && images.rankbutton.complete) {

    ctx.drawImage(images.rankbutton, rankBtn.x, rankBtn.y, rankBtn.w, rankBtn.h);

  }

  if (rankBtn.hover) {

    const next = RANKS[player.rank.rank];

    const lines = next

      ? [`Next: ${next.name}`, `Cost: ${next.cost}`]

      : ['Max Rank'];

    ctx.fillStyle = 'rgba(0,0,0,0.7)';

    ctx.fillRect(rankBtn.x, rankBtn.y + rankBtn.h + 4, 120, lines.length \* 12 + 8);

    ctx.fillStyle = 'white';

    ctx.font = '10px "Press Start 2P"';

    lines.forEach((l, i) => {

      ctx.fillText(l, rankBtn.x + 4, rankBtn.y + rankBtn.h + 16 + i \* 12);

    });

  }

const toolbarX = 10;

const toolbarY = canvas.height - 74;

const btnSize  = 64;

  // === Draw Change-Name Button & Tooltip ===

  if (images.namechange && images.namechange.complete) {

    ctx.drawImage(images.namechange, nameBtn.x, nameBtn.y, nameBtn.w, nameBtn.h);

  }

  if (nameBtn.hover) {

    const txt = 'Change Name';

    const w = ctx.measureText(txt).width + 8;

    ctx.fillStyle = 'rgba(0,0,0,0.7)';

    ctx.fillRect(nameBtn.x, nameBtn.y + nameBtn.h + 4, w, 16);

    ctx.fillStyle = 'white';

    ctx.font = '10px "Press Start 2P"';

    ctx.fillText(txt, nameBtn.x + 4, nameBtn.y + nameBtn.h + 14);

  }

    // === Draw Bag Button & Tooltip ===

  if (images.bag && images.bag.complete) {

    ctx.drawImage(images.bag, bagBtn.x, bagBtn.y, bagBtn.w, bagBtn.h);

  }

  if (bagBtn.hover) {

    ctx.fillStyle = 'rgba(0,0,0,0.7)';

    ctx.fillRect(bagBtn.x, bagBtn.y + bagBtn.h + 4, 80, 20);

    ctx.fillStyle = 'white';

    ctx.font = '12px "Press Start 2P"';

    ctx.fillText('Inventory', bagBtn.x + 4, bagBtn.y + bagBtn.h + 18);

  }

  // <— Insert panels here —>

  if (showUpgrade)      drawUpgradePanel();

  if (showPointUpgrade) drawPointUpgradePanel();

  // — Draw the bag grid when open —

// — Draw the bag panel and handle hover tooltips —

if (showBag) {

  const iconSize = 48, cols = 10, rows = 10, gap = 8;

  const panelW   = iconSize \* cols;

  const panelH   = iconSize \* rows + 24;  // +24px header

  const panelX   = bagBtn.x;

  const panelY   = bagBtn.y - panelH - gap;

  // — Panel background & border —

  ctx.fillStyle   = 'rgba(0,0,0,0.8)';

  ctx.fillRect(panelX, panelY, panelW, panelH);

  ctx.strokeStyle = '#fff'; ctx.lineWidth = 2;

  ctx.strokeRect(panelX, panelY, panelW, panelH);

  // — Header: 8 auto-sell checkboxes for T1–T8 —

  const boxSize = 16, headerY = panelY + 4;

  ctx.font = '12px sans-serif'; ctx.textAlign = 'left';

  for (let t = 1; t <= 8; t++) {

    const x = panelX + (t - 1) \* (panelW / 8) + 8,

          y = headerY;

    ctx.strokeRect(x, y, boxSize, boxSize);

    if (autoSellTiers[t-1]) {

      ctx.fillStyle = '#fff';

      ctx.fillRect(x+4, y+4, boxSize-8, boxSize-8);

    }

    ctx.fillStyle = '#fff';

    ctx.fillText(`T${t}`, x + boxSize + 4, y + boxSize - 2);

  }

  // — Sell-all icon (top-right) —

  const sellSize = 24,

        sellX    = panelX + panelW - sellSize - 8,

        sellY    = panelY + 4;

  if (images.sell && images.sell.complete) {

    ctx.drawImage(images.sell, sellX, sellY, sellSize, sellSize);

  }

  // — Draw 10×10 bag items —

  bag.forEach((item, idx) => {

    if (idx >= cols \* rows) return;

    const col = idx % cols,

          row = Math.floor(idx / cols),

          ix  = panelX + col \* iconSize,

          iy  = panelY + 24 + row \* iconSize;

    const key = `${item.type}t${item.tier}`;

    const img = images[key]?.complete ? images[key] : images[item.type];

    ctx.drawImage(img, ix, iy, iconSize, iconSize);

  });

  // — Tooltip on hover: compare Bag vs Equipped —

  const gridX = panelX, gridY = panelY + 24;

  if (

    mouseX >= gridX && mouseX < gridX + panelW &&

    mouseY >= gridY && mouseY < gridY + iconSize \* rows

  ) {

    const col = Math.floor((mouseX - gridX) / iconSize),

          row = Math.floor((mouseY - gridY) / iconSize),

          idx = row \* cols + col;

    if (idx < bag.length) {

      const item = bag[idx],

            eq   = equipment[item.type] || null,

            cap  = s => s[0].toUpperCase()+s.slice(1),

            bagS = item.stats,

            eqS  = eq ? eq.stats : null;

      const lines = [

        `${cap(item.type)} (T${item.tier}) Lv:${item.level}`,

        `— Bag Item —`,

        `Atk:  +${bagS.attack}`,

        `Def:  +${bagS.def}`,

        `HP:   +${bagS.hp}`,

        `Regen:+${bagS.regen.toFixed(1)}`,

        `Spd:  +${bagS.atkSpeed.toFixed(2)}`

      ];

      if (eqS) {

        lines.push(`— Equipped —`);

        lines.push(`Atk:  +${eqS.attack}`);

        lines.push(`Def:  +${eqS.def}`);

        lines.push(`HP:   +${eqS.hp}`);

        lines.push(`Regen:+${eqS.regen.toFixed(1)}`);

        lines.push(`Spd:  +${eqS.atkSpeed.toFixed(2)}`);

      }

      // measure & draw tooltip

      ctx.font = '14px sans-serif';

      let w=0; lines.forEach(l=>w=Math.max(w,ctx.measureText(l).width));

      const pad=6, lh=16, tw=w+pad\*2, th=lines.length\*lh+pad\*2;

      let tx=mouseX+12, ty=mouseY+12;

      if(tx+tw>canvas.width) tx=mouseX-tw-12;

      if(ty+th>canvas.height)ty=mouseY-th-12;

      ctx.fillStyle='rgba(0,0,0,0.8)'; ctx.fillRect(tx,ty,tw,th);

      ctx.strokeStyle='#fff'; ctx.strokeRect(tx,ty,tw,th);

      ctx.fillStyle='#fff';

      lines.forEach((l,i)=> ctx.fillText(l, tx+pad, ty+pad+(i+0.8)\*lh));

    }

  }

}

  drawStatPanel();

  // Finally, the floating texts & notifications

  drawFloatingTexts();

  drawNotifications();

// … after drawing the buttons …

  // === Notifications ===

  notifications.forEach(n => {

    ctx.globalAlpha = Math.min(1, n.timer / 1.5);

    ctx.fillStyle = n.color;

    ctx.font = '24px sans-serif';

    const w = ctx.measureText(n.text).width;

    ctx.fillText(n.text, n.x - w / 2, n.y);

    ctx.globalAlpha = 1;

  });

  // === Upgrade Buttons ===

  const ux = 10, uy = canvas.height - 74;

  ctx.drawImage(images.upgrade, ux, uy, 64, 64);

  ctx.drawImage(images.pointUpgrade, ux + 74, uy, 64, 64);

   // === Equipment UI & Tooltip ===

types.forEach((t, i) => {

  // 1) Compute slot position

  const x = 10;

  const y = 10 + i \* 60;

  // 2) Draw slot border

  ctx.strokeStyle = '#000';

  ctx.lineWidth   = 3;

  ctx.strokeRect(x, y, 48, 48);

  // 3) Choose icon: empty slot vs. tiered gear

  const eq = equipment[t];

  let iconKey = t; // default empty icon

  if (eq) {

    const tier = eq.tier || 1;

    iconKey = `${t}t${tier}`; // e.g. "glovet3"

  }

  const iconImg = images[iconKey]?.complete ? images[iconKey] : images[t];

  ctx.drawImage(iconImg, x, y, 48, 48);

  // 4) Draw tier/level text if equipped

  if (eq) {

    ctx.fillStyle    = '#fff';

    ctx.font         = '14px sans-serif';

    ctx.textAlign    = 'left';

    ctx.textBaseline = 'top';

    ctx.fillText(

      `T${eq.tier} Lv.${eq.level}`,

      x + 60,

      y + 12

    );

  }

  // 5) Draw the “+” upgrade button

  const btnX = x + 52;

  const btnY = y + 12;

  const btnW = 24;

  const btnH = 24;

  if (images.add && images.add.complete) {

    ctx.drawImage(images.add, btnX, btnY, btnW, btnH);

  }

  // 6) Draw tooltip on hover

  if (hoverEquip === t && eq && eq.stats) {

    const tooltipX = x + 120;

    const tooltipY = y;

    const tooltipW = 140;

    const tooltipH = 150;

    // background

    ctx.fillStyle = 'rgba(243, 11, 11, 0.7)';

    ctx.fillRect(tooltipX, tooltipY, tooltipW, tooltipH);

    ctx.strokeStyle = '#fff';

    ctx.strokeRect(tooltipX, tooltipY, tooltipW, tooltipH);

    // text lines

    ctx.fillStyle = '#fff';

    ctx.font      = '15px sans-serif';

    ctx.textAlign = 'left';

const base = eq.baseStats || { attack:0,def:0,hp:0,regen:0,atkSpeed:0 };

const cur  = eq.stats;

const bonus = {

  attack:   cur.attack   - base.attack,

  def:      cur.def      - base.def,

  hp:       cur.hp       - base.hp,

  regen:    (cur.regen   - base.regen).toFixed(2),

  atkSpeed: (cur.atkSpeed - base.atkSpeed).toFixed(2)

};

    const lines = [

  `Name:  ${t.charAt(0).toUpperCase() + t.slice(1)}`,

  `Tier:  ${eq.tier}`,

  `Level: ${eq.level}`,

  `Atk:   ${base.attack} + ${bonus.attack}`,

  `Def:   ${base.def} + ${bonus.def}`,

  `HP:    ${base.hp} + ${bonus.hp}`,

  `Regen: ${base.regen} + ${bonus.regen}`,

  `Spd:   ${base.atkSpeed} + ${bonus.atkSpeed}`

];

    lines.forEach((line, idx) => {

      ctx.fillText(line, tooltipX + 8, tooltipY + 16 + idx \* 16);

    });

  }

});

}

// ==== Helpers ====

function drawHpBar(e){ ctx.fillStyle='#555';ctx.fillRect(e.x,e.y-10,e.size,6); ctx.fillStyle='#f00';ctx.fillRect(e.x,e.y-10,e.size\*(e.hp/e.maxHp),6); }

function rectOverlap(a,b){ return !(a.x+a.size<b.x||a.x>b.x+b.size||a.y+a.size<b.y||a.y>b.y+b.size); }

function clampPlayer() {

  player.x = Math.max(BG\_X, Math.min(BG\_X + BG\_W - player.size, player.x));

  player.y = Math.max(BG\_Y, Math.min(BG\_Y + BG\_H - player.size, player.y));

}

function applyEquipmentStats() {

  // 1) Base stats

  let atk      = 12;

  let def      = 5;

  let maxHp    = 100;

  let regen    = 1;

  let atkSpeed = 1;     // attacks per second

  const moveSpeed = 180; // constant, fixed movement speed

  // 2) Add rank bonuses

  const rb = player.rank.statBonus || {};

  atk      += rb.atk    || 0;

  def      += rb.def    || 0;

  maxHp    += rb.hp     || 0;

  regen    += rb.regen  || 0;

  atkSpeed += rb.atkSpeed || 0; // if future ranks include atkSpeed

  // 3) Add upgrade bonuses

  atk      += statUpgrades.attack;

  def      += statUpgrades.def;

  maxHp    += statUpgrades.hp;

  regen    += statUpgrades.regen;

  atkSpeed += statUpgrades.atkSpeed || 0;

  // 4) Add equipment bonuses

  types.forEach(t => {

    const eq = equipment[t];

    if (eq && eq.stats) {

      atk      += eq.stats.attack || 0;

      def      += eq.stats.def    || 0;

      maxHp    += eq.stats.hp     || 0;

      regen    += eq.stats.regen  || 0;

      atkSpeed += eq.stats.atkSpeed || 0;

    }

  });

  // 5) Apply to player

  player.attack   = atk;

  player.def      = def;

  player.maxHp    = maxHp;

  player.regen    = regen;

  player.atkSpeed = atkSpeed;

  player.speed    = moveSpeed; // always fixed

  // 6) Clamp HP to new max

  player.hp = Math.min(player.hp, player.maxHp);

}

function tierColor(tier) {

  return ['#ccc','#6cf','#5f5','#cc3','#fc3','#f84','#f39','#f0f'][tier-1] || '#999';

}

function updateDropList() {

  const list = document.getElementById('dropList');

  if (!list) return;

  list.innerHTML = '';

  dropEvents.slice(0, MAX\_DROPS).forEach(evt => {

    const li = document.createElement('li');

    li.style.color = evt.color;

    li.textContent = evt.text;

    list.appendChild(li);

  });

}

window.addEventListener('load', () => {

  updateDropList();

});

updateDropList();

// ==== Start ====

initGame();