<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Lumberjack Tycoon</title>

<script src="https://cdn.tailwindcss.com"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/tone/14.7.77/Tone.js"></script>

<link href="https://fonts.googleapis.com/css2?family=Press+Start+2P&display=swap" rel="stylesheet">

<style>

:root {

--primary-bg: #34495e;

--secondary-bg: #2c3e50;

--accent-color: #e67e22;

--text-color: #ecf0f1;

--success-color: #2ecc71;

--error-color: #e74c3c;

--info-color: #3498db;

--bonus-color: #f1c40f;

--trait-point-color: #3498db;

--ability-point-color: #9b59b6;

--boost-color: #f1c40f;

--epic-color: #9b59b6;

--danger-color: #c0392b;

--font: 'Press Start 2P', cursive;

}

html, body {

overscroll-behavior-y: contain;

}

body {

font-family: var(--font);

background-color: var(--primary-bg);

color: var(--text-color);

text-shadow: 2px 2px #000;

}

.game-container {

width: 100%;

max-width: 1200px;

margin: auto;

padding: 1rem;

}

.game-world {

position: relative;

height: 350px;

background: linear-gradient(to bottom, #3498db 0%, #87ceeb 70%, #27ae60 70%, #2ecc71 100%);

border-radius: 1rem;

border: 4px solid var(--text-color);

margin-bottom: 1rem;

overflow: hidden;

}

.character {

position: absolute;

bottom: 10px;

font-size: clamp(28px, 5vw, 40px);

transition: left 0.5s ease, bottom 0.5s ease, opacity 0.3s ease;

z-index: 10;

}

.building {

position: absolute;

bottom: 10px;

font-size: clamp(40px, 7vw, 60px);

z-index: 5;

transition: opacity 0.5s ease;

}

.game-object {

position: absolute;

font-size: clamp(35px, 6vw, 50px);

cursor: pointer;

transition: transform 0.1s ease, opacity 0.5s ease, left 0.5s ease, bottom 0.5s ease;

-webkit-tap-highlight-color: transparent;

}

.game-object.spawning {

animation: spawn 0.5s ease-out;

}

.game-object.hidden-despawning {

opacity: 0;

transform: scale(0.5);

pointer-events: none;

}

.game-object:hover:not(.hidden-despawning) {

transform: scale(1.1);

}

.floating-text {

position: absolute;

font-size: clamp(1rem, 3vw, 1.2rem);

animation: floatUp 1.5s ease-out forwards;

pointer-events: none;

text-shadow: 2px 2px #000;

}

.floating-text.crit {

color: var(--bonus-color);

font-weight: bold;

font-size: clamp(1.2rem, 4vw, 1.5rem);

animation: floatUp 2s ease-out forwards;

}

.node-health-bar-container {

position: absolute;

top: -15px;

left: 50%;

transform: translateX(-50%);

width: 60px;

height: 8px;

background-color: rgba(0,0,0,0.5);

border-radius: 4px;

border: 1px solid #fff;

}

.node-health-bar {

height: 100%;

background-color: var(--success-color);

border-radius: 3px;

transition: width 0.2s ease;

}

@keyframes floatUp {

from { transform: translateY(0); opacity: 1; }

to { transform: translateY(-80px); opacity: 0; }

}

@keyframes spawn {

from { transform: scale(0); opacity: 0; }

to { transform: scale(1); opacity: 1; }

}

.tab-button {

background: var(--secondary-bg);

padding: 0.5rem 1rem;

font-size: clamp(0.6rem, 2vw, 0.9rem);

border-radius: 0.5rem 0.5rem 0 0;

cursor: pointer;

border: 2px solid var(--text-color);

border-bottom: none;

margin-bottom: -2px;

}

.tab-button.active {

background: var(--accent-color);

}

.panel {

background: rgba(0,0,0,0.3);

padding: 1rem;

border-radius: 0 0.5rem 0.5rem 0.5rem;

border: 2px solid var(--text-color);

display: none;

}

.panel.active {

display: block;

}

.resource-grid {

display: grid;

grid-template-columns: repeat(auto-fit, minmax(140px, 1fr));

gap: 0.5rem;

background: rgba(0,0,0,0.2);

padding: 0.75rem;

border-radius: 0.5rem;

}

.resource-item {

text-align: center;

}

.resource-item .label {

font-size: 0.7rem;

opacity: 0.8;

}

.resource-item .value {

font-size: clamp(0.9rem, 3vw, 1.1rem);

}

.clickable { cursor: pointer; }

.action-button, .ability-button, .craft-button, .quest-button, .use-button, .upgrade-button {

background: var(--secondary-bg);

padding: 0.75rem;

font-size: clamp(0.8rem, 3vw, 1rem);

border-radius: 0.5rem;

border: 2px solid var(--text-color);

cursor: pointer;

transition: background-color 0.2s, transform 0.1s;

width: 100%;

text-align: left;

}

.action-button:hover:not(:disabled), .ability-button:hover:not(:disabled), .craft-button:hover:not(:disabled), .quest-button:hover:not(:disabled), .use-button:hover:not(:disabled), .upgrade-button:hover:not(:disabled) {

background: var(--accent-color);

transform: translateY(-2px);

}

.action-button:disabled, .ability-button:disabled, .craft-button:disabled, .quest-button:disabled, .use-button:disabled, .upgrade-button:disabled {

background: #7f8c8d;

cursor: not-allowed;

opacity: 0.5;

}

.xp-bar {

width: 100%;

background-color: var(--secondary-bg);

border-radius: 0.5rem;

overflow: hidden;

border: 2px solid var(--text-color);

}

.xp-bar-fill {

height: 20px;

background-color: var(--trait-point-color);

width: 0%;

transition: width 0.5s ease;

}

.crafting-grid, .building-grid, .trait-grid {

display: grid;

grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));

gap: 1rem;

}

.ability-button span, .craft-button span, .action-button span, .quest-button span, .use-button span, .upgrade-button span {

display: block;

}

.ability-button .cost, .craft-button .cost, .action-button .cost, .quest-button .cost, .upgrade-button .cost {

font-size: 0.8em;

color: #bdc3c7;

}

.family-member, .quest-item, .trait-item {

background: rgba(0,0,0,0.2);

border-radius: 0.5rem;

padding: 1rem;

}

.quest-item { border-left: 5px solid var(--epic-color); }

.trait-item { border-left: 5px solid var(--trait-point-color); }

.inventory-category {

margin-bottom: 2rem;

}

.inventory-grid {

display: grid;

grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));

gap: 1rem;

}

.inventory-item {

background: rgba(0,0,0,0.2);

border-radius: 0.5rem;

padding: 1rem;

border-left: 5px solid var(--info-color);

}

.inventory-item.epic {

border-left-color: var(--epic-color);

}

.quest-item.completed {

border-left-color: var(--success-color);

opacity: 0.6;

}

.achievement {

background: rgba(0,0,0,0.2);

padding: 0.5rem;

border-radius: 0.5rem;

opacity: 0.5;

transition: opacity 0.3s;

}

.achievement.completed {

opacity: 1;

border: 2px solid var(--success-color);

}

.message-box {

position: fixed;

top: 20px;

left: 50%;

transform: translateX(-50%);

padding: 1rem 2rem;

border-radius: 0.5rem;

border: 2px solid white;

z-index: 1000;

display: none;

text-align: center;

}

.sound-toggle {

font-size: 1.5rem;

cursor: pointer;

}

#status-display, #waifu-request-display {

text-align: center;

height: 1.5em;

}

.status-boost { color: var(--boost-color); }

.status-danger { color: var(--danger-color); }

#waifu-request-display { color: var(--epic-color); font-weight: bold; }

.hidden { display: none !important; }

.trait-item .upgrade-btn {

background-color: var(--trait-point-color);

color: var(--text-color);

border: none;

border-radius: 50%;

width: 30px;

height: 30px;

font-size: 1.2rem;

line-height: 1;

cursor: pointer;

}

.trait-item .upgrade-btn:disabled {

background-color: #7f8c8d;

cursor: not-allowed;

}

.ability-tree-container {

display: flex;

justify-content: space-around;

gap: 2rem;

}

.ability-branch {

flex: 1;

border: 2px solid var(--secondary-bg);

border-radius: 0.5rem;

padding: 1rem;

}

.agent-bill-modal-container {

position: fixed;

top: 0;

left: 0;

width: 100%;

height: 100%;

background-color: rgba(0, 0, 0, 0.8);

display: flex;

justify-content: center;

align-items: center;

z-index: 2000;

padding: 1rem;

}

.agent-bill-modal-content {

background-color: var(--secondary-bg);

border: 4px solid var(--danger-color);

border-radius: 1rem;

padding: 2rem;

width: 100%;

max-width: 600px;

text-shadow: 2px 2px #000;

}

#npc-dialogue-modal .agent-bill-modal-content {

border-color: var(--info-color);

}

#combat-modal .agent-bill-modal-content {

border-color: var(--danger-color);

}

#map-grid {

grid-template-columns: repeat(auto-fit, minmax(180px, 1fr));

}

.weapon-select-btn {

font-size: 1.5rem;

padding: 0.5rem;

border: 2px solid transparent;

}

.weapon-select-btn.active {

border-color: var(--bonus-color);

background-color: rgba(255,255,255,0.2);

}

</style>

</head>

<body class="bg-gray-800">

<div class="game-container">

<!-- Game World -->

<div id="gameWorld" class="game-world">

<div id="lumberjack" class="character" style="left: 30%;">🪓</div>

<div id="miner" class="character" style="left: 70%;">⛏️</div>

<div id="house-building" class="building" style="left: 10%;">🏠</div>

<div id="mine-building" class="building hidden" style="left: 30%;">🏭</div>

<div id="library-building" class="building hidden" style="left: 45%;">📚</div>

<div id="sawmill-building" class="building hidden" style="left: 60%;">🪵</div>

<div id="zoo-building" class="building hidden" style="left: 75%;">🦓</div>

<div id="factory-building" class="building hidden" style="left: 85%;">🎁</div>

</div>

<!-- Player Stats -->

<div class="mb-2">

<div class="flex justify-between items-center mb-1 text-sm">

<span id="player-level-text">LVL: 1</span>

<span id="toggle-sound" class="sound-toggle">🔇</span>

<span id="trait-points-text" class="text-blue-400">TRAIT PTS: 0</span>

<span id="ability-points-text" class="text-purple-400">ABILITY PTS: 0</span>

</div>

<div class="xp-bar"><div id="xp-bar-fill" class="xp-bar-fill"></div></div>

<div id="xp-text" class="text-center text-xs mt-1">0 / 100 XP</div>

</div>

<!-- Status Display -->

<div id="status-display" class="mb-1 font-bold"></div>

<div id="waifu-request-display" class="mb-2"></div>

<!-- Resource Grid -->

<div id="resource-grid" class="resource-grid mb-4"></div>

<!-- Panels -->

<div>

<button class="tab-button active" data-panel="upgrades">Upgrades</button>

<button class="tab-button" data-panel="character">Character</button>

<button class="tab-button" data-panel="abilities">Abilities</button>

<button class="tab-button" data-panel="buildings">Buildings</button>

<button class="tab-button" data-panel="crafting">Crafting</button>

<button class="tab-button" data-panel="inventory">Inventory</button>

<button class="tab-button" data-panel="quests">Quests</button>

<button class="tab-button" data-panel="family">Family</button>

<button class="tab-button" data-panel="achievements">Achievements</button>

<button class="tab-button" data-panel="map">Map</button>

<button class="tab-button" data-panel="npcs">NPCs</button>

<button class="tab-button" data-panel="encounters">Encounters</button>

</div>

<div id="upgrades-panel" class="panel active">

<h2 class="text-2xl mb-4">Upgrades & Actions</h2>

<div class="grid grid-cols-1 md:grid-cols-2 gap-4">

<button id="sell-all-btn" class="action-button">SELL ALL RESOURCES</button>

<button id="upgrade-axe-btn" class="action-button">Upgrade Axe</button>

<button id="upgrade-pickaxe-btn" class="action-button">Upgrade Pickaxe</button>

</div>

</div>

<div id="character-panel" class="panel">

<h2 class="text-2xl mb-4">Character Traits</h2>

<p class="text-sm mb-4">Spend Trait Points earned from leveling up to improve your core stats.</p>

<div id="trait-grid" class="trait-grid"></div>

</div>

<div id="abilities-panel" class="panel">

<h2 class="text-2xl mb-4">Ability Tree</h2>

<div id="ability-tree-start" class="mb-4"></div>

<div id="ability-tree-branches" class="ability-tree-container"></div>

</div>

<div id="buildings-panel" class="panel">

<h2 class="text-2xl mb-4">Your Property</h2>

<div id="building-grid" class="building-grid"></div>

</div>

<div id="crafting-panel" class="panel">

<h2 class="text-2xl mb-4">Item Crafting</h2>

<p class="text-sm mb-4">Craft consumables here. Your permanent items are in the Inventory tab.</p>

<div id="crafting-grid" class="crafting-grid"></div>

</div>

<div id="inventory-panel" class="panel">

<div id="inventory-content"></div>

</div>

<div id="quests-panel" class="panel">

<h2 class="text-2xl mb-4">Town Quests</h2>

<div id="quests-grid" class="space-y-4"></div>

</div>

<div id="family-panel" class="panel">

<h2 class="text-2xl mb-4">Your Family</h2>

<div id="family-grid" class="space-y-4"></div>

<div id="dream-journal" class="mt-4">

<h3 class="text-xl mb-2">Dream Journal</h3>

<button id="dream-btn" class="action-button">✨ Record Last Night's Dream</button>

<p id="dream-output" class="mt-2 text-sm p-2 bg-gray-900 rounded hidden"></p>

</div>

</div>

<div id="achievements-panel" class="panel">

<h2 class="text-2xl mb-4">Achievements</h2>

<div id="achievements-grid" class="grid grid-cols-2 md:grid-cols-4 gap-2 text-center"></div>

</div>

<div id="map-panel" class="panel">

<h2 class="text-2xl mb-4">World Map</h2>

<div id="map-grid" class="grid grid-cols-3 gap-4 text-center"></div>

<div id="current-location-display" class="mt-4 text-xl text-center">Current Location: <span>Town Square</span></div>

</div>

<div id="npcs-panel" class="panel">

<h2 class="text-2xl mb-4">Non-Player Characters</h2>

<div id="npc-list" class="space-y-4"></div>

<div id="npc-dialogue-modal" class="agent-bill-modal-container hidden">

<div class="agent-bill-modal-content">

<h2 id="dialogue-npc-name" class="text-3xl mb-4"></h2>

<div id="dialogue-tabs" class="mb-2"></div>

<div id="dialogue-text" class="mb-4 p-4 bg-gray-900 rounded border border-gray-600 text-lg"></div>

<div id="dialogue-options" class="flex flex-col gap-2"></div>

<div id="market-grid" class="hidden inventory-grid"></div>

<button id="close-dialogue-btn" class="action-button mt-4 bg-gray-700 hover:bg-gray-600">Close</button>

</div>

</div>

</div>

<div id="encounters-panel" class="panel">

<h2 class="text-2xl mb-4">Active Threats</h2>

<div id="active-enemies-list" class="space-y-4">

<!-- Enemies list will be populated here -->

</div>

<p class="text-center text-gray-400" id="no-active-enemies-message">No immediate threats. Explore the map!</p>

</div>

</div>

<!-- Message Box -->

<div id="message-box" class="message-box"></div>

<!-- Agent Bill Modal -->

<div id="agent-bill-modal" class="agent-bill-modal-container hidden">

<div class="agent-bill-modal-content">

<h2 class="text-3xl mb-4 text-red-500">A Shady Offer...</h2>

<p class="mb-2"><span class="font-bold text-gray-400">From:</span> Shady Agent Bill</p>

<p id="agent-bill-prompt" class="mb-4 p-4 bg-gray-900 rounded border border-gray-600 text-lg"></p>

<div class="flex justify-between gap-4">

<button id="agent-bill-accept" class="action-button flex-1 bg-green-600 hover:bg-green-500">Accept Deal</button>

<button id="agent-bill-decline" class="action-button flex-1 bg-red-600 hover:bg-red-500">Decline</button>

</div>

</div>

</div>

<!-- Combat Modal -->

<div id="combat-modal" class="agent-bill-modal-container hidden">

<div class="agent-bill-modal-content">

<h2 id="combat-enemy-name" class="text-3xl mb-4 text-red-500"></h2>

<div class="flex justify-around items-center mb-4">

<div class="text-center">

<p class="text-xl">Your Health:</p>

<div class="xp-bar w-48 mx-auto"><div id="player-combat-health-bar" class="xp-bar-fill bg-green-500" style="width: 100%;"></div></div>

<p id="player-combat-health-text" class="text-lg">100/100</p>

</div>

<div class="text-center">

<p class="text-xl">Enemy Health:</p>

<div class="xp-bar w-48 mx-auto"><div id="enemy-combat-health-bar" class="xp-bar-fill bg-red-500" style="width: 100%;"></div></div>

<p id="enemy-combat-health-text" class="text-lg">50/50</p>

</div>

</div>

<p id="combat-log" class="mb-4 p-4 bg-gray-900 rounded border border-gray-600 text-sm h-32 overflow-y-auto"></p>

<div id="combat-actions" class="flex justify-between gap-4">

<div id="weapon-selection" class="flex gap-2">

<button id="select-axe" class="weapon-select-btn rounded">🪓</button>

<button id="select-pickaxe" class="weapon-select-btn rounded">⛏️</button>

</div>

<button id="combat-attack-btn" class="action-button flex-1 bg-red-600 hover:bg-red-500">Attack!</button>

<button id="combat-flee-btn" class="action-button flex-1 bg-gray-600 hover:bg-gray-500">Flee!</button>

</div>

</div>

</div>

<script type="module">

// DOM Elements

let gameWorld, messageBox, soundToggleBtn, statusDisplay, lumberjackEl, minerEl, waifuRequestDisplay, resourceGridEl;

let houseEl, mineEl, sawmillEl, factoryEl, libraryEl, zooEl;

let activeEnemiesListEl, noActiveEnemiesMessageEl;

// Sound

let sfx;

let soundEnabled = false;

let musicStarted = false;

// Game State

let state = {

resources: { wood: 0, stone: 0, gold: 0, planks: 0, superStone: 0, yogurt: 0, flinstoneVitamins: 0, fishsticks: 0, mangos: 0 },

inventory: {

cheddarPopcorn: 0,

superWood: 0,

legendaryOre: 0,

chocolateBox: 0,

flowerBouquet: 0,

ancientRelic: 0,

shadowScale: 0,

bentoBox: 0,

loveLetter: 0,

diamondRing: 0,

perfume: 0,

teddyBear: 0,

},

tools: { axe: 1, pickaxe: 1, baseDamage: 1 },

buildings: {

house: 1,

mine: 0,

sawmill: 0,

factory: 0,

library: 0,

zoo: 0,

},

collection: { goldenAxeLevel: 0, dankCurryLevel: 0 },

stats: {

totalWoodChopped: 0,

totalStoneMined: 0,

totalGoldEarned: 0,

totalPlanksCrafted: 0,

elderEntKills: 0,

stoneTitanKills: 0,

ancientGolemKills: 0,

shadowSerpentKills: 0,

},

player: {

level: 1,

xp: 0,

traitPoints: 0,

abilityPoints: 0,

traits: {

strength: 1,

agility: 1,

intelligence: 1,

rizz: 1,

luck: 1,

anime: 1,

},

currentLocation: 'townSquare',

combat: {

health: 100,

currentHealth: 100,

attack: 10,

defense: 5,

}

},

abilities: {

honorableWorker: 0,

lumberjackPath: 0,

woodExpertise: 0,

forestsBlessing: 0,

splinteringStrikes: 0,

chainsawMastery: 0,

minerPath: 0,

stoneExpertise: 0,

earthsBounty: 0,

shatteringBlows: 0,

jackhammerMastery: 0,

},

npcs: {

oldManJenkins: { relationship: 0, quests: { oldManJenkinsWood: { status: 'available' } } },

merchantMolly: { relationship: 0},

townCrier: { relationship: 0, quest: null }

},

waifu: {

affection: 0,

unlocked: false,

lastInteractionTime: 0,

},

babies: 0,

questStatus: {

beaver: 'unavailable',

elias: 'unavailable',

mayor: 'unavailable',

oldManJenkinsWood: 'unavailable',

},

waifuRequest: {

active: false,

itemKey: null,

endTime: 0,

},

agentBillMission: {

active: false,

completed: false,

prompt: null,

requirements: {},

reward: {}

},

townCrierQuest: {

active: false,

completed: false,

prompt: null,

requirements: {},

reward: {}

},

combat: {

active: false,

currentEnemy: null,

playerTurn: true,

log: [],

fleeAttempts: 0,

selectedWeapon: 'axe',

turnCount: 0,

},

activeEnemies: [],

lastDreamTime: 0,

// Buffs

speedBoostEndTime: 0,

slowdownEndTime: 0,

damageBoostEndTime: 0,

goldBoostEndTime: 0,

bentoBoostEndTime: 0,

dropBoostEndTime: 0,

lastClickTime: 0,

lastGameUpdate: Date.now(),

waifuMinigameActive: false,

waifuCollected: 0,

discountNextPurchase: 0,

};

let resourceNodes = [];

// Game Configuration

const CONFIG = {

baseUpgradeCost: 50,

prices: { wood: 2, stone: 3, planks: 8 },

baseBabyCost: 20,

xpPerAction: 1,

xpPerGold: 0.1,

xpForLevel: (level) => 100 \* Math.pow(1.5, level - 1),

speedBoostDuration: 15000,

slowdownDuration: 10000,

globalClickCooldown: 500,

rareDropChance: 0.02,

buildings: {

house: { name: "House", emoji: "🏠", baseCost: { gold: 100 }, upgradeCostMultiplier: 1.8, baseProduction: 0.2, productionResource: 'gold' },

mine: { name: "Auto Mine", emoji: "🏭", baseCost: { wood: 100, stone: 50 }, upgradeCostMultiplier: 2.0, baseProduction: 0.25, productionResource: 'stone' },

library: { name: "Library", emoji: "📚", baseCost: { wood: 500, planks: 50 }, upgradeCostMultiplier: 2.5, baseProduction: 0.1, productionResource: 'xp' },

sawmill: { name: "Sawmill", emoji: "🪵", baseCost: { wood: 200, stone: 100 }, upgradeCostMultiplier: 2.2, baseProduction: 0.1, productionResource: 'planks', consumes: { resource: 'wood', amount: 2 } },

zoo: { name: "Zoo", emoji: "🦓", baseCost: { wood: 1000, gold: 500 }, upgradeCostMultiplier: 3.0, baseProduction: 0.01, productionResource: 'cheddarPopcorn' },

factory: { name: "Gift Factory", emoji: "🎁", baseCost: { gold: 1000, superStone: 5 }, upgradeCostMultiplier: 3.0, baseProduction: 0.02, productionResource: 'waifuGift', consumes: { resource: 'superStone', amount: 1 } },

},

waifuGifts: { loveLetter: "💌", diamondRing: "💍", perfume: "💨", teddyBear: "🧸", chocolateBox: "🍫", flowerBouquet: "💐" },

epicItems: {

bentoBox: { name: "Bento Box", emoji: "🍱", desc: "Use for a massive 5x damage boost for 30 seconds." },

cheddarPopcorn: { name: "Cheddar Popcorn", emoji: "🍿", desc: "A salty, cheesy snack. Seems important." },

superWood: { name: "Super Wood", emoji: "🪵+", desc: "Extremely rare wood from the Elder Ent. Can be sold for a lot.", goldValue: 200 },

legendaryOre: { name: "Legendary Ore", emoji: "💎", desc: "A sparkling ore from the Stone Titan. Very valuable.", goldValue: 500 },

},

inventory: {

ancientRelic: { name: "Ancient Relic", emoji: "🏺", desc: "A mystical artifact from the Ancient Golem. Worth a fortune.", goldValue: 2000 },

shadowScale: { name: "Shadow Scale", emoji: "🐉", desc: "A dark, shimmering scale from the Shadow Serpent. Highly sought after.", goldValue: 5000 },

},

resourceTiers: {

tree: [

{ emoji: '🌳', health: 3, yield: 1, count: 3, respawnTime: 5000 },

{ emoji: '🌲', health: 6, yield: 4, count: 2, respawnTime: 10000 },

{ emoji: '🌴', health: 9, yield: 8, count: 1, respawnTime: 15000 }

],

rock: [

{ emoji: '🪨', health: 4, yield: 1, count: 3, respawnTime: 6000 },

{ emoji: '⛰️', health: 8, yield: 4, count: 2, respawnTime: 12000 },

{ emoji: '🗿', health: 12, yield: 8, count: 1, respawnTime: 18000 }

]

},

map: {

townSquare: {

name: "Town Square",

emoji: "🏛️",

description: "The bustling heart of the village.",

connections: ["denseForest", "rockyMountains", "riverside"],

encounterChance: 0.15, // Increased chance for more enemies

potentialEncounters: ['greedyRaccoon', 'yawningSnail', 'brainrotChild'],

npcs: ['oldManJenkins', 'merchantMolly', 'townCrier'],

resourceTypes: ['tree', 'rock'] // Ensures resources spawn at start

},

denseForest: {

name: "Dense Forest",

emoji: "🌲",

description: "Thick woods, good for woodcutting.",

connections: ["townSquare", "hiddenGrove"],

encounterChance: 0.3,

potentialEncounters: ['angryBeaver', 'hauntedLog', 'christmasTree'],

resourceType: 'tree'

},

rockyMountains: {

name: "Rocky Mountains",

emoji: "🏔️",

description: "Great for mining stone and rare minerals.",

connections: ["townSquare", "deepCave", "ancientRuins"],

encounterChance: 0.3,

potentialEncounters: ['funkyFungus', 'stegosaurus'],

resourceType: 'rock'

},

riverside: {

name: "Riverside",

emoji: "🏞️",

description: "A calm river, perhaps with some fish.",

connections: ["townSquare"],

encounterChance: 0.2,

potentialEncounters: ['nastySquid', 'friendlyDolphinJeff']

},

hiddenGrove: {

name: "Hidden Grove",

emoji: "🌳+",

description: "A mystical place, rumored to hold secrets. Home to the Elder Ent.",

connections: ["denseForest"],

encounterChance: 0.4,

potentialEncounters: ['yogurtGolem'],

bossEncounter: 'elderEnt'

},

deepCave: {

name: "Deep Cave",

emoji: "🦇",

description: "A dark, ominous cave. Tread carefully. The Stone Titan guards its depths.",

connections: ["rockyMountains", "shadowCavern"],

encounterChance: 0.5,

potentialEncounters: [],

bossEncounter: 'stoneTitan'

},

ancientRuins: {

name: "Ancient Ruins",

emoji: "🏛️",

description: "The crumbling remnants of an ancient civilization. Beware the colossal golem!",

connections: ["rockyMountains"],

encounterChance: 0.6,

potentialEncounters: ['hauntedLog', 'stegosaurus'],

bossEncounter: 'ancientGolem'

},

shadowCavern: {

name: "Shadow Cavern",

emoji: "💀",

description: "A cavern steeped in unnatural darkness. The Shadow Serpent lurks within its deepest chambers.",

connections: ["deepCave"],

encounterChance: 0.7,

potentialEncounters: ['nastySquid', 'yogurtGolem'],

bossEncounter: 'shadowSerpent'

},

},

npcs: {

oldManJenkins: {

name: "Old Man Jenkins",

emoji: "👴",

dialogue: [

{ type: "initial", text: "Ah, a young lumberjack! How's the woodcutting these days?", options: [{ text: "It's good!", response: "That's the spirit! (+5 relationship)", effect: { relationship: 5 } }, { text: "It's tough work.", response: "Aye, it always is. Keep at it. (-2 relationship)", effect: { relationship: -2} }] },

{ type: "quest\_available", text: "My back isn't what it used to be. Could you fetch me 10 wood?", options: [{ text: "Sure thing!", response: "Much obliged!", effect: { startQuest: 'oldManJenkinsWood' } }] },

{ type: "quest\_complete", text: "You're a lifesaver! Here's a little something.", options: [{ text: "Thanks!", response: "", effect: { completeQuest: 'oldManJenkinsWood' } }] },

{ type: "low\_relationship", text: "Hmph. Don't waste my time, whippersnapper.", options: []},

{ type: "high\_relationship", text: "You're a good kid, you know that? The best. Keep up the good work.", options: [{ text: "Thanks, Old Man!", response: ""}]}

],

quests: {

oldManJenkinsWood: {

name: "Jenkins' Firewood",

desc: "Old Man Jenkins needs 10 wood for his fireplace.",

req: { wood: 10 },

reward: { gold: 50, xp: 100 }

}

}

},

merchantMolly: {

name: "Merchant Molly",

emoji: "👩‍💼",

shop: [

{ item: 'flinstoneVitamins', price: 150 },

{ item: 'fishsticks', price: 120 },

{ item: 'mangos', price: 100 },

{ item: 'superStone', price: 1000 },

],

dialogue: [

{ type: "initial", text: "Welcome! Looking for anything special today?", options: [{ text: "Just Browse.", response: "Come back when you're ready to spend!", effect: { relationship: -1}}, { text: "Got any rare goods?", response: "Perhaps, for the right price... (+3 relationship)", effect: { relationship: 3 } }] },

{ type: "high\_relationship", text: "A valued customer like you deserves a discount!", options: [{ text: "Show me the deals!", response: "", effect: { discountNextPurchase: 0.1}}]}

],

persona: "You are Merchant Molly, a cheerful and slightly greedy shopkeeper in a quirky lumberjack town. You are always trying to make a sale. Your dialogue should be friendly but always hint at making a profit."

},

townCrier: {

name: "Town Crier",

emoji: "📣",

dialogue: [],

persona: "You are the Town Crier. You have a booming voice and a flair for the dramatic. You announce new quests and happenings in the town with great importance, even if they are mundane."

}

},

enemies: {

// Bonus enemies

goldenStar: { type: 'bonus', emoji: '⭐', spawnInterval: 35000, lifespan: 10000, handler: handleGoldenStarClick },

friendlyDolphinJeff: { type: 'bonus', emoji: '🐬', spawnInterval: 90000, lifespan: 8000, handler: handleDolphinClick},

smellyHamburgerGene: { type: 'bonus', emoji: '🍔', spawnInterval: 180000, lifespan: 15000, handler: handleHamburgerClick },

// Regular Combat Enemies

yawningSnail: { type: 'enemy', emoji: '🐌', spawnInterval: 30000, lifespan: 15000, combat: { health: 20, attack: 5, defense: 5, xpReward: 15, goldReward: 5, dodgeChance: 0.01, critChance: 0, specialMove: 'slow' }, name: "Yawning Snail" },

brainrotChild: { type: 'enemy', emoji: '📱', spawnInterval: 50000, lifespan: 10000, combat: { health: 30, attack: 8, defense: 2, xpReward: 25, goldReward: 10, dodgeChance: 0.2, critChance: 0.05, specialMove: 'stun' }, name: "Brainrot Child" },

greedyRaccoon: { type: 'enemy', emoji: '🦝', spawnInterval: 45000, lifespan: 10000, combat: { health: 30, attack: 7, defense: 3, xpReward: 30, goldReward: 15, dodgeChance: 0.15, critChance: 0.08, specialMove: 'stealGold' }, name: "Greedy Raccoon" },

christmasTree: { type: 'enemy', emoji: '🎄', spawnInterval: 40000, lifespan: 10000, combat: { health: 40, attack: 9, defense: 4, xpReward: 40, goldReward: 20, dodgeChance: 0.05, critChance: 0.05, specialMove: 'slow' }, name: "Christmas Tree" },

funkyFungus: { type: 'enemy', emoji: '🍄', spawnInterval: 40000, lifespan: 8000, combat: { health: 45, attack: 10, defense: 5, xpReward: 45, goldReward: 22, dodgeChance: 0.05, critChance: 0.05, specialMove: 'poison' }, name: "Funky Fungus" },

hauntedLog: { type: 'enemy', emoji:'🪵👻', spawnInterval: 50000, lifespan: 12000, combat: { health: 50, attack: 11, defense: 6, xpReward: 50, goldReward: 25, dodgeChance: 0.07, critChance: 0.07, specialMove: 'fear' }, name: "Haunted Log" },

angryBeaver: { type: 'enemy', emoji: '🦫', spawnInterval: 60000, lifespan: 10000, combat: { health: 55, attack: 12, defense: 7, xpReward: 55, goldReward: 28, dodgeChance: 0.12, critChance: 0.06, specialMove: 'gnaw' }, name: "Angry Beaver" },

nastySquid: { type: 'enemy', emoji: '🦑', spawnInterval: 30000, lifespan: 12000, combat: { health: 60, attack: 13, defense: 8, xpReward: 60, goldReward: 30, dodgeChance: 0.10, critChance: 0.05, specialMove: 'stun' }, name: "Nasty Squid" },

stegosaurus: { type: 'enemy', emoji: '🦕', spawnInterval: 60000, lifespan: 15000, combat: { health: 70, attack: 15, defense: 9, xpReward: 70, goldReward: 35, dodgeChance: 0.08, critChance: 0.10, specialMove: 'roar' }, name: "Stegosaurus" },

yogurtGolem: { type: 'enemy', emoji: '🍦', spawnInterval: 120000, lifespan: 15000, combat: { health: 80, attack: 18, defense: 10, xpReward: 80, goldReward: 40, dodgeChance: 0.03, critChance: 0.08, specialMove: 'sticky' }, name: "Yogurt Golem" },

// Bosses

elderEnt: { type: 'boss', emoji: '🌳😠', combat: { health: 200, attack: 20, defense: 10, xpReward: 500, goldReward: 200, specialDrop: 'superWood', dodgeChance: 0.10, critChance: 0.15, specialMove: 'rootBind' }, name: "Elder Ent", desc: "An ancient, powerful tree spirit."},

stoneTitan: { type: 'boss', emoji: '🗿🔥', combat: { health: 300, attack: 25, defense: 15, xpReward: 750, goldReward: 300, specialDrop: 'legendaryOre', dodgeChance: 0.05, critChance: 0.20, specialMove: 'quake' }, name: "Stone Titan", desc: "A massive golem carved from mountain rock." },

ancientGolem: {

type: 'boss', emoji: '🤖', combat: {

health: 1000,

attack: 100,

defense: 50,

xpReward: 2500,

goldReward: 1000,

dodgeChance: 0.08,

specialDrop: 'ancientRelic',

critChance: 0.18,

specialMove: 'tremor'

},

name: "Ancient Golem", desc: "A colossal guardian forged from primordial earth, protecting forgotten treasures."

},

shadowSerpent: {

type: 'boss', emoji: '🐍', combat: {

health: 1200,

attack: 120,

defense: 60,

xpReward: 3500,

goldReward: 1500,

specialDrop: 'shadowScale',

dodgeChance: 0.20,

critChance: 0.25,

specialMove: 'darkness'

},

name: "Shadow Serpent", desc: "A mythical serpent born of pure shadow, capable of devouring light and hope."

},

},

abilities: {

honorableWorker: { name: "Honorable Worker", desc: "+1 to Axe & Pickaxe base damage.", cost: {gold: 100, wood: 100, stone: 100, ap: 1}, max: 1 },

lumberjackPath: { name: "Path of the Lumberjack", desc: "Commit to the way of the forest.", cost: {gold: 1000, ap: 0}, max: 1, requires: 'honorableWorker' },

woodExpertise: { name: "Wood Expertise", desc: "Gain +5% wood from all sources per point.", cost: {gold: 2000, ap: 2}, max: 5, requires: 'lumberjackPath' },

forestsBlessing: { name: "Forest's Blessing", desc: "Trees have a 10% chance to drop extra wood.", cost: {gold: 5000, ap: 5}, max: 1, requires: 'woodExpertise' },

splinteringStrikes: { name: "Splintering Strikes", desc: "Critical hits on trees deal +50% damage.", cost: {gold: 10000, ap: 5}, max: 1, requires: 'forestsBlessing' },

chainsawMastery: { name: "Chainsaw Mastery", desc: "Axe speed is dramatically increased.", cost: { gold: 25000, ap: 10}, max: 1, requires: 'splinteringStrikes' },

minerPath: { name: "Path of the Miner", desc: "Commit to the way of the mountain.", cost: {gold: 1000, ap: 0}, max: 1, requires: 'honorableWorker' },

stoneExpertise: { name: "Stone Expertise", desc: "Gain +5% stone from all sources per point.", cost: {gold: 2000, ap: 2}, max: 5, requires: 'minerPath' },

earthsBounty: { name: "Earth's Bounty", desc: "Rocks have a 10% chance to drop extra stone.", cost: {gold: 5000, ap: 5}, max: 1, requires: 'stoneExpertise' },

shatteringBlows: { name: "Shattering Blows", desc: "Critical hits on rocks deal +50% damage.", cost: {gold: 10000, ap: 5}, max: 1, requires: 'earthsBounty' },

jackhammerMastery: { name: "Jackhammer Mastery", desc: "Pickaxe speed is dramatically increased.", cost: {gold: 25000, ap: 10}, max: 1, requires: 'shatteringBlows' },

},

traits: {

strength: { name: "Strength", emoji: "💪", desc: "Increases base click damage by 1 per point." },

agility: { name: "Agility", emoji: "🏃", desc: "Reduces global click cooldown by 1.5% per point." },

intelligence: { name: "Intelligence", emoji: "🧠", desc: "Increases all XP gains by 5% per point." },

rizz: { name: "Rizz", emoji: "😎", desc: "Increases Gold from selling by 2% per point." },

luck: { name: "Luck", emoji: "🍀", desc: "Increases rare item drop chance by 5% per point." },

anime: { name: "Anime", emoji: "✨", desc: "Increases critical hit chance by 1% per point." }

},

crafting: {

gigaShake: { name: "Giga-Gains Shake", emoji: "🥤", desc: "Temporarily double click damage for 30s.", cost: { flinstoneVitamins: 1, fishsticks: 1}},

vitaminCBlast: { name: "Vitamin C-Blast", emoji: "🍊", desc: "Instantly gain 25% of XP needed for next level.", cost: { flinstoneVitamins: 1, mangos: 1}},

tropicalSalad: { name: "Tropical Fish Salad", emoji: "🥗", desc: "Doubles gold from selling for 60s.", cost: { fishsticks: 1, mangos: 1 }},

superVitamin: { name: "Super Vitamin", emoji: "💊", desc: "Permanently increases base click damage by 1.", cost: { flinstoneVitamins: 5 } },

fishBait: { name: "Fish Bait", emoji: "🎣", desc: "Doubles rare drop chance for 2 minutes.", cost: {fishsticks: 3} },

mangoSmoothie: { name: "Mango Smoothie", emoji: "🥭", desc: "Instantly gain 100 Gold.", cost: { mangos: 2}}

},

collection: {

goldenAxe: { name: "Golden Axe", emoji: "🪓✨", desc: "Increases chop damage by 5 per level.", baseCost: { stone: 100, gold: 500 }, upgradeCostMultiplier: 2.5},

dankCurry: { name: "Dank Curry", emoji: "🍛", desc: "Increases speed boost duration by 5% per level.", baseCost: { yogurt: 3, mangos: 10 }, upgradeCostMultiplier: 2}

},

quests: {

beaver: { name: "Beaver Dave's Dam", level: 8, desc: "Beaver Dave needs 50 wood to repair his dam.", req: { wood: 50 }, reward: { xp: 500, gold: 200 } },

elias: { name: "Elias's Barn", level: 14, desc: "Help Amish Farmer Elias rebuild his barn with 50 wood and 30 stone.", req: { wood: 50, stone: 30 }, reward: { xp: 700, gold: 300 } },

mayor: { name: "Mayor's 'Fee'", level: 20, desc: "The corrupt Mayor demands a 'fee' of 100 wood, 50 stone, and 100 gold.", req: { wood: 100, stone: 50, gold: 100 }, reward: { xp: 1000, traitPoints: 1}}

},

achievements: {

firstWood: { name: "First Wood", desc: "Chop your first piece of wood.", goal: () => state.stats.totalWoodChopped >= 1 },

firstStone: { name: "First Stone", desc: "Mine your first piece of stone.", goal: () => state.stats.totalStoneMined >= 1 },

plankMaster: { name: "Plank Master", desc: "Craft 50 planks.", goal: () => state.stats.totalPlanksCrafted >= 50 },

richTycoon: { name: "Rich Tycoon", desc: "Earn 1000 gold.", goal: () => state.stats.totalGoldEarned >= 1000 },

waifu: { name: "True Love", desc: "Unlock the Beautiful Waifu.", goal: () => state.waifu.unlocked },

firstBaby: { name: "Bundle of Joy", desc: "Acquire your first baby.", goal: () => state.babies >= 1 },

fiveBabies: { name: "Full House", desc: "Raise a family of 5 babies.", goal: () => state.babies >= 5 },

entSlayer: { name: "Ent Slayer", desc: "Defeat the Elder Ent.", goal: () => state.stats.elderEntKills >= 1 || false },

titanConqueror: { name: "Titan Conqueror", desc: "Defeat the Stone Titan.", goal: () => state.stats.stoneTitanKills >= 1 || false },

golemHunter: { name: "Golem Hunter", desc: "Defeat the Ancient Golem.", goal: () => state.stats.ancientGolemKills >= 1 || false },

serpentSlayer: { name: "Serpent Slayer", desc: "Defeat the Shadow Serpent.", goal: () => state.stats.shadowSerpentKills >= 1 || false }

}

};

// Initialize achievement tracking

state.stats.ancientGolemKills = state.stats.ancientGolemKills || 0;

state.stats.shadowSerpentKills = state.stats.shadowSerpentKills || 0;

Object.keys(CONFIG.achievements).forEach(key => CONFIG.achievements[key].completed = false);

document.addEventListener('DOMContentLoaded', () => {

try {

assignElements();

initializeSounds();

setupUI();

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

// Using a timeout to ensure the DOM is fully ready for manipulation

setTimeout(() => {

createWorldObjects();

spawnNPCsInWorld();

setInterval(update, 100);

for (const key in CONFIG.enemies) {

scheduleNextSpawn(key);

}

}, 100);

});

} catch (error) {

console.error("A critical error occurred during initialization:", error);

document.body.innerHTML = '<div class="text-red-500 p-8">A critical error occurred. Please refresh the page. Check the console for details.</div>';

}

});

function assignElements() {

gameWorld = document.getElementById('gameWorld');

messageBox = document.getElementById('message-box');

soundToggleBtn = document.getElementById('toggle-sound');

statusDisplay = document.getElementById('status-display');

waifuRequestDisplay = document.getElementById('waifu-request-display');

resourceGridEl = document.getElementById('resource-grid');

lumberjackEl = document.getElementById('lumberjack');

minerEl = document.getElementById('miner');

houseEl = document.getElementById('house-building');

mineEl = document.getElementById('mine-building');

sawmillEl = document.getElementById('sawmill-building');

factoryEl = document.getElementById('factory-building');

libraryEl = document.getElementById('library-building');

zooEl = document.getElementById('zoo-building');

// Combat & NPC Elements

document.getElementById('combat-attack-btn').onclick = playerAttack;

document.getElementById('combat-flee-btn').onclick = playerFlee;

document.getElementById('close-dialogue-btn').onclick = closeDialogueModal;

document.getElementById('dream-btn').onclick = haveADream;

document.getElementById('select-axe').onclick = () => selectWeapon('axe');

document.getElementById('select-pickaxe').onclick = () => selectWeapon('pickaxe');

// Encounters Panel Elements

activeEnemiesListEl = document.getElementById('active-enemies-list');

noActiveEnemiesMessageEl = document.getElementById('no-active-enemies-message');

}

function initializeSounds() {

const sfxVolume = new Tone.Volume(-18).toDestination();

sfx = {

chop: () => new Tone.MembraneSynth().connect(sfxVolume).triggerAttackRelease("C2", "8n"),

mine: () => new Tone.MembraneSynth().connect(sfxVolume).triggerAttackRelease("G1", "8n"),

coin: () => new Tone.MetalSynth({frequency: 400, envelope: {attack: 0.001, decay: 0.1, release: 0.1}}).toDestination().triggerAttackRelease("C5", "32n"),

build: () => new Tone.NoiseSynth({noise: {type: "white"}, envelope: {attack: 0.005, decay: 0.3, sustain: 0}}).toDestination().triggerAttackRelease("2n"),

error: () => new Tone.Synth().toDestination().triggerAttackRelease("C2", "8n"),

powerup: () => new Tone.Synth({oscillator: {type: "sawtooth"}, envelope: { attack: 0.01, decay: 0.1, sustain: 0.2, release: 0.1}}).toDestination().triggerAttackRelease("C4", "8n")

};

const masterReverb = new Tone.Reverb({ decay: 4, wet: 0.5 }).toDestination();

const musicBus = new Tone.Compressor(-20, 10).connect(masterReverb);

const padSynth = new Tone.PolySynth(Tone.FMSynth, { volume: -22, envelope: { attack: 2, decay: 0.1, sustain: 0.9, release: 5}, harmonicity: 1.5, modulationIndex: 1.2 }).connect(musicBus);

const chords = [ { time: '0:0', notes: ['C3', 'E4', 'G4'] }, { time: '1:0', notes: ['A3', 'C4', 'E4'] }, { time: '2:0', notes: ['F3', 'A3', 'C4'] }, { time: '3:0', notes: ['G3', 'B3', 'D4'] } ];

const arpSynth = new Tone.MonoSynth({ volume: -20, oscillator: { type: 'triangle8' }, envelope: { attack: 0.01, decay: 0.2, sustain: 0.1, release: 0.3} }).connect(musicBus);

const arpPattern = new Tone.Pattern((time, noteMidi) => {

arpSynth.triggerAttackRelease(Tone.Frequency(noteMidi, 'midi'), '16n', time);

}, [60, 64, 67], 'up');

arpPattern.interval = '16n';

arpPattern.start(0);

const chordPart = new Tone.Part((time, value) => {

padSynth.triggerAttackRelease(value.notes, '1m', time);

arpPattern.values = value.notes.map(note => Tone.Frequency(note).toMidi());

}, chords).start(0);

chordPart.loop = true;

chordPart.loopEnd = '4m';

const melodyData = [

{ notes: ['C5', 'E5', null, 'G5', null, 'E5', null, 'C5'], subdivision: '8n', volume: -18 },

{ notes: [null, 'A4', 'C5', null, 'A4', null, 'E4', null], subdivision: '8n', volume: -20 },

{ notes: ['F4', null, 'F4', null, 'A4', null, 'C5', null], subdivision: '4n', volume: -19},

{ notes: [null, null, null, 'G4', 'B4', 'D5', 'B4', 'G4'], subdivision: '8n', volume: -21 }

];

melodyData.forEach(data => {

const melodySynth = new Tone.Synth({ volume: data.volume, oscillator: { type: 'sine' }, envelope: { attack: 0.1, decay: 0.3, sustain: 0.4, release: 1.2 } }).connect(musicBus);

const sequence = new Tone.Sequence((time, note) => {

if (note !== null) melodySynth.triggerAttackRelease(note, '8n', time);

}, data.notes, data.subdivision).start(0);

sequence.loop = true;

});

Tone.Transport.bpm.value = 90;

Tone.Transport.pause();

}

function createWorldObjects() {

gameWorld.querySelectorAll('.game-object:not(.character)').forEach(n => n.remove());

resourceNodes = [];

const currentLocationData = CONFIG.map[state.player.currentLocation];

// Support both `resourceType` (string) and `resourceTypes` (array) for locations

const typesToSpawn = currentLocationData.resourceTypes || (currentLocationData.resourceType ? [currentLocationData.resourceType] : []);

typesToSpawn.forEach(resourceType => {

if (CONFIG.resourceTiers[resourceType]) {

CONFIG.resourceTiers[resourceType].forEach(tier => {

// In town square, spawn fewer of each type to not clutter it.

const count = (state.player.currentLocation === 'townSquare' && tier.count > 1) ? 1 : tier.count;

for (let i = 0; i < count; i++) {

createNode(resourceType, tier);

}

});

}

});

state.activeEnemies.forEach(e => clearTimeout(e.despawnTimer));

state.activeEnemies = [];

updateActiveEnemiesListUI();

}

function playSound(soundFunc) {

if (soundEnabled) {

if (Tone.context.state !== 'running') Tone.context.resume();

soundFunc();

}

}

function toggleMusic() {

soundEnabled = !soundEnabled;

soundToggleBtn.textContent = soundEnabled ? '🔊' : '🔇';

if (soundEnabled) {

if (!musicStarted) {

Tone.start();

musicStarted = true;

}

Tone.Transport.start();

} else {

Tone.Transport.pause();

}

}

function addXp(amount) {

const xpBonus = 1 + (state.player.traits.intelligence \* 0.05);

state.player.xp += amount \* xpBonus;

let xpNeeded = CONFIG.xpForLevel(state.player.level);

while (state.player.xp >= xpNeeded) {

state.player.level++;

state.player.xp -= xpNeeded;

state.player.traitPoints++;

if (state.player.level % 3 === 0) {

state.player.abilityPoints++;

}

showMessage(`Leveled up to ${state.player.level}!`, 'bonus');

if (state.player.level % 5 === 0 && state.player.level >= 5 && !state.waifu.unlocked && !state.waifuMinigameActive) {

startWaifuMinigame();

}

if (state.player.level === 30 && !state.agentBillMission.completed) {

startAgentBillMission();

}

updateQuestsUI();

xpNeeded = CONFIG.xpForLevel(state.player.level);

}

}

function handleHit(node, damage, characterEl) {

if (node.element.classList.contains('hidden-despawning')) return;

if (state.combat.active) {

showMessage("You are busy fighting!", "error");

return;

}

const now = Date.now();

const agilityBonus = 1 - (state.player.traits.agility \* 0.015);

let cooldown = CONFIG.globalClickCooldown \* agilityBonus;

if (node.type === 'tree' && state.abilities.chainsawMastery > 0) cooldown \*= 0.25;

if (node.type === 'rock' && state.abilities.jackhammerMastery > 0) cooldown \*= 0.25;

if (state.speedBoostEndTime > now) cooldown /= 4;

if (state.slowdownEndTime > now) cooldown \*= 2;

if (now - state.lastClickTime < cooldown) return;

state.lastClickTime = now;

moveCharacterToNode(characterEl, node.element);

let currentDamage = damage;

if (state.damageBoostEndTime > now) currentDamage \*= 2;

if (state.bentoBoostEndTime > now) currentDamage \*= 5;

const critChance = state.player.traits.anime \* 0.01;

if (Math.random() < critChance) {

let critMultiplier = 1.5 + (state.player.traits.anime \* 0.1);

if (node.type === 'tree' && state.abilities.splinteringStrikes > 0) critMultiplier += 0.5;

if (node.type === 'rock' && state.abilities.shatteringBlows > 0) critMultiplier += 0.5;

currentDamage \*= critMultiplier;

showFloatingText(`CRIT! ${Math.floor(currentDamage)}`, node.element, true);

} else {

showFloatingText(`-${Math.floor(currentDamage)}`, node.element);

}

node.health -= currentDamage;

playSound(node.type === 'tree' ? sfx.chop : sfx.mine);

const luckBonus = 1 + (state.player.traits.luck \* 0.05);

let dropChance = CONFIG.rareDropChance \* luckBonus;

if (state.dropBoostEndTime > now) dropChance \*= 2;

if (Math.random() < dropChance) {

const drops = ['flinstoneVitamins', 'fishsticks', 'mangos'];

const droppedItem = drops[Math.floor(Math.random() \* drops.length)];

state.resources[droppedItem]++;

const dropEmojis = { flinstoneVitamins: '💊', fishsticks: '🐟', mangos: '🥭' };

showMessage(`Lucky! You found a ${dropEmojis[droppedItem]}!`, 'bonus');

}

if (node.health <= 0) {

node.element.classList.add('hidden-despawning');

let resourceGained = node.tier.yield;

if (node.type === 'tree') {

resourceGained \*= (1 + state.abilities.woodExpertise \* 0.05);

if (state.abilities.forestsBlessing > 0 && Math.random() < 0.1) resourceGained \*= 2;

state.resources.wood += resourceGained;

state.stats.totalWoodChopped += resourceGained;

} else {

resourceGained \*= (1 + state.abilities.stoneExpertise \* 0.05);

if (state.abilities.earthsBounty > 0 && Math.random() < 0.1) resourceGained \*= 2;

state.resources.stone += resourceGained;

state.stats.totalStoneMined += resourceGained;

}

addXp(CONFIG.xpPerAction \* node.tier.health);

checkAchievements();

respawnNode(node);

}

updateNodeHealthBar(node);

}

// --- Enemy Handlers ---

function scheduleNextSpawn(enemyKey) {

const enemyData = CONFIG.enemies[enemyKey];

const baseInterval = enemyData.spawnInterval;

if (!baseInterval) return;

const randomMultiplier = Math.random() \* (1.1 - 0.75) + 0.75;

const nextInterval = baseInterval \* randomMultiplier;

setTimeout(() => {

const currentLocation = CONFIG.map[state.player.currentLocation];

const canSpawnInLocation = enemyData.type === 'bonus' ||

(currentLocation.potentialEncounters && currentLocation.potentialEncounters.includes(enemyKey)) ||

(currentLocation.bossEncounter && currentLocation.bossEncounter === enemyKey);

if (canSpawnInLocation) {

spawnEnemy(enemyKey);

}

scheduleNextSpawn(enemyKey);

}, nextInterval);

}

function spawnEnemy(enemyKey) {

if (state.combat.active) return;

const enemyData = CONFIG.enemies[enemyKey];

const uniqueId = enemyKey + '\_' + Date.now() + '\_' + Math.floor(Math.random() \* 1000);

const enemy = {

id: uniqueId,

key: enemyKey,

element: document.createElement('div'),

animationId: null,

};

enemy.element.classList.add('game-object');

enemy.element.textContent = enemyData.emoji;

enemy.element.id = `enemy-spawn-${uniqueId}`;

if (enemyData.type === 'bonus') {

enemy.element.addEventListener('click', () => {

enemyData.handler(enemy);

});

} else {

enemy.element.addEventListener('click', () => {

showMessage(`You can fight the ${enemyData.name || enemyKey.replace(/([A-Z])/g, ' $1').trim()} from the Encounters tab!`, 'info');

});

}

const pos = getSafeSpawnPosition({ width: 50, height: 50 });

enemy.element.style.left = `${pos.left}px`;

enemy.element.style.bottom = `${pos.bottom}px`;

enemy.element.classList.add('spawning');

setTimeout(() => enemy.element.classList.remove('spawning'), 500);

gameWorld.appendChild(enemy.element);

state.activeEnemies.push(enemy);

const messageType = enemyData.type === 'bonus' ? 'bonus' : 'info';

showMessage(`A ${enemyData.name || enemyKey.replace(/([A-Z])/g, ' $1').trim()} has appeared!`, messageType);

if (enemyKey === 'friendlyDolphinJeff') {

enemy.startTime = Date.now();

enemy.animationId = requestAnimationFrame(() => animateDolphin(enemy));

}

enemy.despawnTimer = setTimeout(() => {

const index = state.activeEnemies.findIndex(e => e.id === uniqueId);

if (index > -1) {

if(enemy.animationId) cancelAnimationFrame(enemy.animationId);

showMessage(`The ${enemyData.name || enemy.key.replace(/([A-Z])/g, ' $1').trim()} disappeared.`, 'info');

enemy.element.remove();

state.activeEnemies.splice(index, 1);

updateActiveEnemiesListUI();

}

}, enemyData.lifespan);

updateActiveEnemiesListUI();

}

function animateDolphin(enemy) {

const worldRect = gameWorld.getBoundingClientRect();

const elapsedTime = Date.now() - enemy.startTime;

const progress = elapsedTime / CONFIG.enemies.friendlyDolphinJeff.lifespan;

if (progress > 1 || !enemy.element.parentElement) {

if(enemy.animationId) cancelAnimationFrame(enemy.animationId);

return;

}

const x = progress \* worldRect.width;

const y = (worldRect.height / 3) + Math.sin(progress \* Math.PI \* 4) \* (worldRect.height / 4);

enemy.element.style.left = `${x}px`;

enemy.element.style.top = `${y}px`;

enemy.animationId = requestAnimationFrame(() => animateDolphin(enemy));

}

function removeEnemy(enemy) {

clearTimeout(enemy.despawnTimer);

if(enemy.animationId) cancelAnimationFrame(enemy.animationId);

enemy.element.remove();

const index = state.activeEnemies.findIndex(e => e.id === enemy.id);

if (index > -1) state.activeEnemies.splice(index, 1);

updateActiveEnemiesListUI();

}

function handleGoldenStarClick(enemy) {

removeEnemy(enemy);

playSound(sfx.coin);

let boostDuration = CONFIG.speedBoostDuration \* (1 + state.collection.dankCurryLevel \* 0.05);

state.speedBoostEndTime = Date.now() + boostDuration;

state.resources.yogurt++;

showMessage("SPEED FRENZY and a Yogurt! 🍦", "bonus");

}

function handleDolphinClick(enemy) {

removeEnemy(enemy);

playSound(sfx.powerup);

state.resources.superStone++;

showMessage("Jeff the Dolphin left you a Super Stone! 💎", "bonus");

}

function handleHamburgerClick(enemy) {

removeEnemy(enemy);

playSound(sfx.build);

let woodCollected = 0;

let stoneCollected = 0;

resourceNodes.forEach(node => {

if (node.element.classList.contains('hidden-despawning')) return;

if (node.type === 'tree') woodCollected += node.tier.yield;

else stoneCollected += node.tier.yield;

node.element.classList.add('hidden-despawning');

respawnNode(node);

});

state.resources.wood += woodCollected;

state.resources.stone += stoneCollected;

showMessage(`Gene's stick bomb cleared the screen! +${woodCollected}W, +${stoneCollected}S`, 'bonus');

}

// --- Gemini API Functions ---

async function callGemini(prompt) {

const payload = {

contents: [{ role: "user", parts: [{ text: prompt }] }],

};

const apiKey = ""; // Leave empty, handled by environment

const apiUrl = `https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent?key=${apiKey}`;

try {

const response = await fetch(apiUrl, {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify(payload)

});

if (!response.ok) {

throw new Error(`API request failed with status ${response.status}`);

}

const result = await response.json();

if (result.candidates && result.candidates.length > 0 &&

result.candidates[0].content && result.candidates[0].content.parts &&

result.candidates[0].content.parts.length > 0) {

return result.candidates[0].content.parts[0].text;

} else {

console.error("Invalid API response structure", result);

return null;

}

} catch (error) {

console.error("Error calling Gemini API:", error);

return null;

}

}

async function getGeminiPoweredMission(prompt, schema) {

const payload = {

contents: [{ role: "user", parts: [{ text: prompt }] }],

generationConfig: {

responseMimeType: "application/json",

responseSchema: schema

}

};

const apiKey = ""; // Leave empty

const apiUrl = `https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent?key=${apiKey}`;

try {

const response = await fetch(apiUrl, {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify(payload)

});

if (!response.ok) throw new Error(`API request failed with status ${response.status}`);

const result = await response.json();

if (result.candidates && result.candidates[0].content && result.candidates[0].content.parts[0]) {

return JSON.parse(result.candidates[0].content.parts[0].text);

} else {

throw new Error("Invalid API response structure");

}

} catch (error) {

console.error("Error fetching mission from Gemini API:", error);

return null;

}

}

// --- Agent Bill Scenario ---

async function startAgentBillMission() {

const modal = document.getElementById('agent-bill-modal');

const promptEl = document.getElementById('agent-bill-prompt');

const acceptBtn = document.getElementById('agent-bill-accept');

const declineBtn = document.getElementById('agent-bill-decline');

promptEl.textContent = "Agent Bill is contacting his sources...";

modal.classList.remove('hidden');

const prompt = 'You are a shady government agent named "Bill" in a lumberjack-themed idle game. The player is level 30. Create a short, morally ambiguous mission. The mission should be slightly absurd and fit the game\'s quirky tone. Specify what the player needs to collect and the reward. Format the response as a single JSON object with three keys: "missionText" (a string for the player), "requirements" (a JSON object mapping resource names like \'wood\', \'stone\', \'planks\', \'superStone\', \'gold\' to integer amounts), and "reward" (a JSON object mapping reward types like \'gold\', \'xp\', or \'buff\' to values). For the \'buff\' reward, the value should be a string like \'damageBoost\'. Do not include any text outside of the JSON object.';

const schema = {

type: "OBJECT",

properties: {

"missionText": { "type": "STRING" },

"requirements": { "type": "OBJECT" },

"reward": { "type": "OBJECT" }

},

required: ["missionText", "requirements", "reward"]

};

let mission = await getGeminiPoweredMission(prompt, schema);

if (!mission) {

// Fallback mission

mission = {

missionText: "My comms are down. We'll have to do this the old-fashioned way. Bring me 250 wood and 250 stone. I'll make it worth your while.",

requirements: { wood: 250, stone: 250 },

reward: { gold: 500, xp: 1000 }

};

}

state.agentBillMission.prompt = mission.missionText;

state.agentBillMission.requirements = mission.requirements;

state.agentBillMission.reward = mission.reward;

promptEl.textContent = mission.missionText;

acceptBtn.onclick = () => {

state.agentBillMission.active = true;

showMessage("Deal accepted. The mission is in your Quests tab.", "bonus");

modal.classList.add('hidden');

updateQuestsUI();

};

declineBtn.onclick = () => {

state.agentBillMission.completed = true; // Mark as completed so it doesn't trigger again

showMessage("Agent Bill has vanished... for now.", "info");

modal.classList.add('hidden');

};

}

function completeAgentBillMission() {

const mission = state.agentBillMission;

if (!mission.active) return;

for (const res in mission.requirements) {

if ((state.resources[res] || 0) < mission.requirements[res]) {

showMessage('You don\'t have the resources for Agent Bill!', 'error');

return;

}

}

for (const res in mission.requirements) {

state.resources[res] -= mission.requirements[res];

}

let rewardText = "Agent Bill is pleased. Reward: ";

const rewards = [];

for (const rew in mission.reward) {

if (rew === 'gold') {

state.resources.gold += mission.reward[rew];

rewards.push(`${mission.reward[rew]} Gold`);

} else if (rew === 'xp') {

addXp(mission.reward[rew]);

rewards.push(`${mission.reward[rew]} XP`);

} else if (rew === 'buff' && mission.reward[rew] === 'damageBoost') {

state.damageBoostEndTime = Date.now() + 60000;

rewards.push("60s Damage Boost!");

}

}

showMessage(rewardText + rewards.join(', '), 'success');

playSound(sfx.powerup);

mission.active = false;

mission.completed = true;

updateQuestsUI();

}

// --- UI Setup and Rendering ---

function setupUI() {

document.getElementById('sell-all-btn').onclick = sellAll;

document.getElementById('upgrade-axe-btn').onclick = () => upgradeTool('axe');

document.getElementById('upgrade-pickaxe-btn').onclick = () => upgradeTool('pickaxe');

soundToggleBtn.onclick = toggleMusic;

document.querySelectorAll('.tab-button').forEach(btn => {

btn.addEventListener('click', (event) => {

const panelId = event.currentTarget.dataset.panel;

document.querySelectorAll('.panel').forEach(p => p.classList.remove('active'));

document.querySelectorAll('.tab-button').forEach(b => b.classList.remove('active'));

document.getElementById(`${panelId}-panel`).classList.add('active');

event.currentTarget.classList.add('active');

if (panelId === 'map') updateMapUI();

if (panelId === 'npcs') updateNPCsUI();

if (panelId === 'encounters') updateActiveEnemiesListUI();

// Update NPC visibility whenever a tab is changed

updateNPCVisibility();

});

});

const resourceNames = { wood: 'Wood', stone: 'Stone', gold: 'Gold', planks: 'Planks', superStone: 'Super Stone', yogurt: 'Yogurt', flinstoneVitamins: 'Vitamins', fishsticks: 'Fishsticks', mangos: 'Mangos' };

const resourceEmojis = { wood: '🪵', stone: '🪨', gold: '💰', planks: '🧱', superStone: '💎', yogurt: '🍦', flinstoneVitamins: '💊', fishsticks: '🐟', mangos: '🥭' };

resourceGridEl.innerHTML = Object.keys(resourceNames).map(key => `<div id="res-${key}" class="resource-item"><div class="label">${resourceNames[key]}</div><div class="value">${resourceEmojis[key]} 0</div></div>`).join('');

updateAbilityUI();

updateTraitsUI();

updateCraftingUI();

updateBuildingUI();

updateQuestsUI();

updateInventoryPanelUI();

updateFamilyUI();

updateNPCsUI();

updateMapUI();

const achievementsGrid = document.getElementById('achievements-grid');

achievementsGrid.innerHTML = Object.keys(CONFIG.achievements).map(key => `<div id="ach-${key}" class="achievement"><strong>${CONFIG.achievements[key].name}</strong><p class="text-xs">${CONFIG.achievements[key].desc}</p></div>`).join('');

}

function updateCraftingUI() {

const craftingContainer = document.getElementById('crafting-grid');

craftingContainer.innerHTML = Object.keys(CONFIG.crafting).map(key => {

const recipe = CONFIG.crafting[key];

const costString = Object.entries(recipe.cost).map(([res, count]) => `${count} ${res.replace(/([A-Z])/g, ' $1').trim()}`).join(', ');

return `

<div class="crafting-recipe inventory-item">

<h3 class="text-lg font-bold">${recipe.emoji} ${recipe.name}</h3>

<p class="text-sm mb-2">${recipe.desc}</p>

<button id="craft-${key}" class="craft-button w-full">

<span>Craft</span>

<span class="cost">${costString}</span>

</button>

</div>`;

}).join('');

Object.keys(CONFIG.crafting).forEach(key => document.getElementById(`craft-${key}`).onclick = () => craftItem(key));

}

function updateFamilyUI() {

const familyContainer = document.getElementById('family-grid');

let waifuHTML = '';

if (state.waifu.unlocked) {

const rizzBonus = 1 + (state.player.traits.rizz \* 0.01);

waifuHTML = `<div class="family-member"><h3 class="text-lg font-bold">💖 Beautiful Waifu</h3><p>Your loving partner. Affection: ${state.waifu.affection.toFixed(0)}/100</p><p>Sometimes she'll ask you to craft things for her!</p></div>`;

} else {

waifuHTML = '<div class="family-member"><h3 class="text-lg font-bold">? Mystery Partner</h3><p>Reach level 5 to meet them.</p></div>';

}

let babyHTML = '';

if (state.waifu.unlocked) {

const babyCost = CONFIG.baseBabyCost + (state.babies \* 15);

babyHTML = `

<div class="family-member">

<h3 class="text-lg font-bold">Babies: ${state.babies}</h3>

<p>Each baby provides a small, passive production boost.</p>

<button id="trade-baby-btn" class="action-button mt-2">

<span>Trade for a Baby</span>

<span class="cost">Cost: ${babyCost}G</span>

</button>

</div>`;

}

familyContainer.innerHTML = waifuHTML + babyHTML;

if (state.waifu.unlocked) {

const babyBtn = document.getElementById('trade-baby-btn');

babyBtn.onclick = handleTradeBaby;

babyBtn.disabled = state.resources.gold < (CONFIG.baseBabyCost + (state.babies \* 15));

}

}

function updateBuildingUI() {

const container = document.getElementById('building-grid');

container.innerHTML = Object.keys(CONFIG.buildings).map(key => {

const building = CONFIG.buildings[key];

const level = state.buildings[key];

let title = `<h3 class="text-lg font-bold">${building.emoji} ${building.name}</h3>`;

if (level > 0) {

title = `<h3 class="text-lg font-bold">${building.emoji} ${building.name} (Lvl ${level})</h3>`;

}

let cost = {};

let buttonText = 'Build';

if (level === 0) {

cost = building.baseCost;

} else {

buttonText = `Upgrade to Lvl ${level + 1}`;

for (const res in building.baseCost) {

cost[res] = Math.floor(building.baseCost[res] \* Math.pow(building.upgradeCostMultiplier, level));

}

}

const costString = Object.entries(cost).map(([r, c]) => `${c} ${r.replace(/([A-Z])/g, ' $1').trim()}`).join(', ');

return `

<div class="building-item">

${title}

<p class="text-sm mb-2">Generates ${(building.baseProduction \* level).toFixed(2)} ${building.productionResource.replace(/([A-Z])/g, ' $1').trim()}/sec.</p>

<button id="build-upgrade-${key}" class="action-button mt-2">

<span>${buttonText}</span>

<span class="cost">Cost: ${costString}</span>

</button>

</div>`;

}).join('');

Object.keys(CONFIG.buildings).forEach(key => {

document.getElementById(`build-upgrade-${key}`).onclick = () => buildOrUpgradeBuilding(key);

});

}

function updateInventoryPanelUI() {

const container = document.getElementById('inventory-content');

container.innerHTML = `

<div class="inventory-category">

<h2 class="text-2xl mb-4">Epic Gear</h2>

<div id="inventory-epic-gear" class="inventory-grid"></div>

</div>

<div class="inventory-category">

<h2 class="text-2xl mb-4">Special Items</h2>

<div id="inventory-special" class="inventory-grid"></div>

</div>

<div class="inventory-category">

<h2 class="text-2xl mb-4">Consumables</h2>

<div id="inventory-consumables" class="inventory-grid"></div>

</div>

`;

const epicContainer = document.getElementById('inventory-epic-gear');

const specialContainer = document.getElementById('inventory-special');

const consumablesContainer = document.getElementById('inventory-consumables');

epicContainer.innerHTML = Object.keys(CONFIG.collection).map(key => {

const item = CONFIG.collection[key];

const level = state.collection[`${key}Level`];

let title = `<h3 class="text-lg font-bold">${item.emoji} ${item.name}</h3>`;

if (level > 0) title = `<h3 class="text-lg font-bold">${item.emoji} ${item.name} (Lvl ${level})</h3>`;

let cost = {};

let buttonText = 'Craft';

if (level === 0) {

cost = item.baseCost;

} else {

buttonText = `Upgrade to Lvl ${level + 1}`;

for (const res in item.baseCost) {

cost[res] = Math.floor(item.baseCost[res] \* Math.pow(item.upgradeCostMultiplier, level));

}

}

const costString = Object.entries(cost).map(([r, c]) => `${c} ${r.replace(/([A-Z])/g, ' $1').trim()}`).join(', ');

return `

<div class="inventory-item epic">

${title}

<p class="text-sm mb-2">${item.desc}</p>

<button id="upgrade-collection-${key}" class="upgrade-button w-full">

<span>${buttonText}</span>

<span class="cost">${costString}</span>

</button>

</div>`;

}).join('');

Object.keys(CONFIG.collection).forEach(key => document.getElementById(`upgrade-collection-${key}`).onclick = () => upgradeEpicItem(key));

for (const key in state.inventory) {

if (state.inventory[key] <= 0) continue;

const itemData = CONFIG.crafting[key] || CONFIG.waifuGifts[key] || CONFIG.epicItems[key] || CONFIG.inventory[key];

if (!itemData) continue;

const itemEmoji = itemData.emoji || (CONFIG.waifuGifts[key] ? CONFIG.waifuGifts[key] : '?');

const itemName = itemData.name || key.replace(/([A-Z])/g, ' $1').trim();

const itemDesc = itemData.desc || "An item you possess.";

const el = document.createElement('div');

el.className = 'inventory-item';

el.innerHTML = `

<h3 class="text-lg font-bold">${itemEmoji} ${itemName} (x${state.inventory[key]})</h3>

<p class="text-sm mb-2">${itemDesc}</p>

<button id="use-item-${key}" class="use-button w-full">Use</button>

`;

const useButton = el.querySelector('button');

if (CONFIG.epicItems[key] && !CONFIG.crafting[key]) {

useButton.disabled = true;

useButton.textContent = "Cannot Use";

if (itemData.goldValue) {

useButton.textContent = "Sell for Gold";

useButton.disabled = false;

useButton.onclick = () => sellBossDrop(key);

}

specialContainer.appendChild(el);

} else if (CONFIG.inventory[key] && !CONFIG.crafting[key]) {

useButton.disabled = false;

useButton.textContent = "Sell for Gold";

useButton.onclick = () => sellBossDrop(key);

specialContainer.appendChild(el);

} else if (CONFIG.waifuGifts[key]) {

if (state.waifuRequest.active && state.waifuRequest.itemKey === key) {

useButton.textContent = "Give to Waifu";

} else {

useButton.disabled = true;

useButton.textContent = "Cannot Use";

}

specialContainer.appendChild(el);

} else if (CONFIG.crafting[key]) {

consumablesContainer.appendChild(el);

useButton.onclick = () => useItem(key);

}

}

}

function sellBossDrop(itemKey) {

if (!state.inventory[itemKey] || state.inventory[itemKey] <= 0) return;

const itemData = CONFIG.epicItems[itemKey] || CONFIG.inventory[itemKey];

if (!itemData || !itemData.goldValue) return;

const goldValue = itemData.goldValue;

if (goldValue > 0) {

state.resources.gold += goldValue;

state.inventory[itemKey]--;

showMessage(`Sold ${itemData.name} for ${goldValue} Gold!`, 'success');

playSound(sfx.coin);

updateInventoryPanelUI();

}

}

// --- Main Game Loop ---

function update() {

const now = Date.now();

const deltaTime = (now - state.lastGameUpdate) / 1000;

const babyBonus = 1 + (state.babies \* 0.05);

const rizzBonus = 1 + (state.player.traits.rizz \* 0.01);

for (const key in state.buildings) {

const level = state.buildings[key];

if (level > 0) {

const building = CONFIG.buildings[key];

const production = level \* building.baseProduction \* babyBonus \* deltaTime;

if (building.consumes) {

const cost = building.consumes.amount \* production;

if (state.resources[building.consumes.resource] >= cost) {

state.resources[building.consumes.resource] -= cost;

if (building.productionResource === 'waifuGift') {

if (Math.random() < production) {

const giftKeys = Object.keys(CONFIG.waifuGifts);

const randomGift = giftKeys[Math.floor(Math.random() \* giftKeys.length)];

if (!state.inventory[randomGift]) state.inventory[randomGift] = 0;

state.inventory[randomGift]++;

showMessage(`The factory produced a ${randomGift.replace(/([A-Z])/g, ' $1').trim()}!`, 'bonus');

}

} else {

state.resources[building.productionResource] += production;

if(building.productionResource === 'planks') state.stats.totalPlanksCrafted += production;

}

}

} else {

if (building.productionResource === 'xp') {

addXp(production);

} else if (building.productionResource === 'cheddarPopcorn') {

if (Math.random() < production) {

state.inventory.cheddarPopcorn++;

showMessage('The Zoo produced some Cheddar Popcorn!', 'bonus');

}

} else {

state.resources[building.productionResource] += production;

}

}

}

}

// Waifu passive affection gain

if (state.waifu.unlocked) {

state.waifu.affection = Math.min(100, state.waifu.affection + (0.01 \* deltaTime \* rizzBonus));

}

if (state.waifu.unlocked && !state.waifuRequest.active && Math.random() < 0.0005) {

startWaifuRequest();

}

if (state.waifuRequest.active) {

const timeLeft = Math.ceil((state.waifuRequest.endTime - now) / 1000);

if (timeLeft <= 0) {

waifuRequestDisplay.textContent = "Your waifu's request expired...";

setTimeout(() => {

if (waifuRequestDisplay.textContent === "Your waifu's request expired...") {

waifuRequestDisplay.textContent = "";

}

}, 2000);

state.waifuRequest.active = false;

state.waifu.affection = Math.max(0, state.waifu.affection - 10);

showMessage("Your Waifu is disappointed. (-10 Affection)", "error");

} else {

const itemEmoji = CONFIG.waifuGifts[state.waifuRequest.itemKey];

waifuRequestDisplay.textContent = `Waifu wants a ${itemEmoji}! (${timeLeft}s left)`;

}

}

const resourceEmojis = { wood: '🪵', stone: '🪨', gold: '💰', planks: '🧱', superStone: '💎', yogurt: '🍦', flinstoneVitamins: '💊', fishsticks: '🐟', mangos: '🥭' };

for (const key in resourceEmojis) {

const el = document.getElementById(`res-${key}`);

if (el) {

el.querySelector('.value').textContent = `${resourceEmojis[key]} ${Math.floor(state.resources[key])}`;

}

}

const xpNeeded = CONFIG.xpForLevel(state.player.level);

document.getElementById('player-level-text').textContent = `LVL: ${state.player.level}`;

document.getElementById('trait-points-text').textContent = `TRAIT PTS: ${state.player.traitPoints}`;

document.getElementById('ability-points-text').textContent = `ABILITY PTS: ${state.player.abilityPoints}`;

document.getElementById('xp-text').textContent = `${Math.floor(state.player.xp)} / ${Math.floor(xpNeeded)} XP`;

document.getElementById('xp-bar-fill').style.width = `${(state.player.xp / xpNeeded) \* 100}%`;

updateDynamicButtons();

let statuses = [];

if (state.bentoBoostEndTime > now) statuses.push(`<span class="status-boost">BENTO POWER! x5 DMG! (${Math.ceil((state.bentoBoostEndTime - now) / 1000)}s)</span>`);

if (state.abilities.chainsawMastery > 0) statuses.push(`<span class="status-boost">CHAINSAW!</span>`);

if (state.abilities.jackhammerMastery > 0) statuses.push(`<span class="status-boost">JACKHAMMER!</span>`);

if (state.speedBoostEndTime > now) statuses.push(`<span class="status-boost">SPEED FRENZY! (${Math.ceil((state.speedBoostEndTime - now) / 1000)}s)</span>`);

if (state.damageBoostEndTime > now) statuses.push(`<span class="status-boost">GIGA GAINS! (${Math.ceil((state.damageBoostEndTime - now) / 1000)}s)</span>`);

if (state.goldBoostEndTime > now) statuses.push(`<span class="status-boost">GOLD RUSH! (${Math.ceil((state.goldBoostEndTime - now) / 1000)}s)</span>`);

if (state.dropBoostEndTime > now) statuses.push('<span class="status-boost">LUCKY! (' + Math.ceil((state.dropBoostEndTime - now) / 1000) + 's)</span>');

if (state.slowdownEndTime > now) statuses.push(`<span class="status-danger">SLOWED! (${Math.ceil((state.slowdownEndTime - now) / 1000)}s)</span>`);

if (state.combat.active) statuses.push(`<span class="status-danger">COMBAT!</span>`);

statusDisplay.innerHTML = statuses.join(' | ');

state.lastGameUpdate = now;

}

function updateDynamicButtons() {

const axeCost = CONFIG.baseUpgradeCost \* Math.pow(2, state.tools.axe - 1);

const pickaxeCost = CONFIG.baseUpgradeCost \* Math.pow(2, state.tools.pickaxe - 1);

document.getElementById('upgrade-axe-btn').innerHTML = `<span>Upgrade Axe (Lvl ${state.tools.axe + 1})</span><span class="cost">Cost: ${axeCost}G</span>`;

document.getElementById('upgrade-axe-btn').disabled = state.resources.gold < axeCost;

document.getElementById('upgrade-pickaxe-btn').innerHTML = `<span>Upgrade Pickaxe (Lvl ${state.tools.pickaxe + 1})</span><span class="cost">Cost: ${pickaxeCost}G</span>`;

document.getElementById('upgrade-pickaxe-btn').disabled = state.resources.gold < pickaxeCost;

updateBuildingUI();

updateQuestsUI();

updateInventoryPanelUI();

updateFamilyUI();

updateAbilityUI();

updateTraitsUI();

for (const key in CONFIG.achievements) {

const ach = CONFIG.achievements[key];

const achEl = document.getElementById(`ach-${key}`);

if (ach.completed && achEl) {

achEl.classList.add('completed');

}

}

for (const key in CONFIG.crafting) {

const recipe = CONFIG.crafting[key];

const btn = document.getElementById(`craft-${key}`);

if (btn) {

let canAfford = true;

for (const res in recipe.cost) {

if (state.resources[res] < recipe.cost[res]) {

canAfford = false;

break;

}

}

btn.disabled = !canAfford;

}

}

// Update player combat stats

state.player.combat.attack = state.tools.baseDamage + state.player.traits.strength + state.tools.axe;

state.player.combat.defense = Math.floor(state.player.traits.strength \* 0.5);

}

function showMessage(text, type = 'info') {

messageBox.textContent = text;

messageBox.style.color = 'var(--text-color)';

if (type === 'success') {

messageBox.style.backgroundColor = 'var(--success-color)';

} else if (type === 'error') {

messageBox.style.backgroundColor = 'var(--error-color)';

} else if (type === 'bonus') {

messageBox.style.backgroundColor = 'var(--bonus-color)';

messageBox.style.color = '#FFFFFF';

messageBox.style.textShadow = '2px 2px #000';

} else {

messageBox.style.backgroundColor = 'var(--info-color)';

}

messageBox.style.display = 'block';

setTimeout(() => {

messageBox.style.display = 'none';

}, 3000);

}

function startWaifuMinigame() {

if (state.waifu.unlocked || state.waifuMinigameActive) return;

state.waifuMinigameActive = true;

state.waifuCollected = 0;

const totalItems = 20;

const required = Math.ceil(totalItems \* 0.75);

for (let i = 0; i < totalItems; i++) {

const item = document.createElement('div');

item.classList.add('game-object');

item.textContent = '💖';

const pos = getSafeSpawnPosition({ width: 50, height: 50 });

item.style.left = `${pos.left}px`;

item.style.bottom = `${pos.bottom}px`;

item.style.fontSize = '2rem';

item.addEventListener('click', () => {

if (!state.waifuMinigameActive) return;

state.waifuCollected++;

item.remove();

});

gameWorld.appendChild(item);

}

showMessage(`Collect ${required} hearts to impress your waifu!`, "bonus");

setTimeout(() => {

state.waifuMinigameActive = false;

document.querySelectorAll('#gameWorld .game-object').forEach(el => {

if (el.textContent === '💖') el.remove();

});

if (state.waifuCollected >= required) {

state.waifu.unlocked = true;

state.waifu.affection = 25;

showMessage("You unlocked the Beautiful Waifu!", "bonus");

playSound(sfx.coin);

checkAchievements();

} else {

showMessage("Waifu vanished... Try again at the next level!", "error");

}

updateFamilyUI();

}, 20000);

}

function startWaifuRequest() {

const giftKeys = Object.keys(CONFIG.waifuGifts);

const itemKey = giftKeys[Math.floor(Math.random() \* giftKeys.length)];

state.waifuRequest.active = true;

state.waifuRequest.itemKey = itemKey;

state.waifuRequest.endTime = Date.now() + 120000; // 2 minutes

const itemEmoji = CONFIG.waifuGifts[itemKey];

showMessage(`Your Waifu wants a gift! She wants a ${itemKey.replace(/([A-Z])/g, ' $1').trim()} ${itemEmoji}.`, 'bonus');

}

function completeWaifuRequest(itemKey) {

if (!state.waifuRequest.active || itemKey !== state.waifuRequest.itemKey) return;

if (!state.inventory[itemKey] || state.inventory[itemKey] < 1) return;

state.inventory[itemKey]--;

const rizzBonus = 1 + (state.player.traits.rizz \* 0.01);

state.waifu.affection = Math.min(100, state.waifu.affection + (15 \* rizzBonus));

showMessage(`You gave the gift to your Waifu! She is ecstatic! (+${Math.floor(15 \* rizzBonus)} Affection)`, 'success');

if (!state.inventory.bentoBox) state.inventory.bentoBox = 0;

state.inventory.bentoBox++;

showMessage('She gave you a Bento Box as thanks! Use it for a massive damage boost!', 'bonus');

state.waifuRequest.active = false;

state.waifuRequest.itemKey = null;

waifuRequestDisplay.textContent = "";

updateFamilyUI();

}

function handleTradeBaby() {

const cost = CONFIG.baseBabyCost + (state.babies \* 15);

if (state.resources.gold >= cost) {

state.resources.gold -= cost;

state.babies++;

showMessage("Congratulations! You got a baby!", "success");

playSound(sfx.coin);

checkAchievements();

} else {

showMessage("Not enough gold!", "error");

playSound(sfx.error);

}

updateFamilyUI();

}

function craftItem(key) {

const recipe = CONFIG.crafting[key];

if (!recipe) return;

for (const res in recipe.cost) {

if (state.resources[res] < recipe.cost[res]) {

showMessage('Not enough materials!', 'error');

playSound(sfx.error);

return;

}

}

for (const res in recipe.cost) {

state.resources[res] -= recipe.cost[res];

}

if (!state.inventory[key]) state.inventory[key] = 0;

state.inventory[key]++;

showMessage(`Crafted ${recipe.name}!`, 'success');

playSound(sfx.build);

updateInventoryPanelUI();

}

function useItem(key) {

if (!state.inventory[key] || state.inventory[key] <= 0) return;

if (CONFIG.epicItems[key]) return;

state.inventory[key]--;

if (key === 'bentoBox') {

state.bentoBoostEndTime = Date.now() + 30000;

playSound(sfx.powerup);

showMessage('Bento Power! 5x damage for 30 seconds!', 'bonus');

return;

}

if (CONFIG.waifuGifts[key]) {

completeWaifuRequest(key);

return;

}

const item = CONFIG.crafting[key];

playSound(sfx.powerup);

showMessage(`Used ${item.name}!`, 'success');

switch(key) {

case 'gigaShake':

state.damageBoostEndTime = Date.now() + 30000;

break;

case 'vitaminCBlast':

addXp(CONFIG.xpForLevel(state.player.level) \* 0.25);

break;

case 'tropicalSalad':

state.goldBoostEndTime = Date.now() + 60000;

break;

case 'superVitamin':

state.tools.baseDamage++;

showMessage("Your base damage permanently increased!", "bonus");

break;

case 'fishBait':

state.dropBoostEndTime = Date.now() + 120000;

break;

case 'mangoSmoothie':

state.resources.gold += 100;

break;

}

updateInventoryPanelUI();

}

function updateQuestsUI() {

const container = document.getElementById('quests-grid');

container.innerHTML = '';

if (state.agentBillMission.active) {

const mission = state.agentBillMission;

const el = document.createElement('div');

el.className = 'quest-item';

el.style.borderLeftColor = 'var(--danger-color)';

const reqString = Object.entries(mission.requirements)

.map(([res, count]) => `${count} ${res.replace(/([A-Z])/g, ' $1').trim()}`).join(', ');

el.innerHTML = `

<h3 class="text-lg font-bold">Shady Offer</h3>

<p class="text-sm mb-2">${mission.prompt}</p>

<button id="agent-bill-complete-btn" class="quest-button w-full">

<span>Complete 'Task'</span>

<span class="cost">Requires: ${reqString}</span>

</button>

`;

container.appendChild(el);

const completeBtn = document.getElementById('agent-bill-complete-btn');

completeBtn.onclick = completeAgentBillMission;

let canComplete = true;

for (const res in mission.requirements) {

if ((state.resources[res] || 0) < mission.requirements[res]) {

canComplete = false;

break;

}

}

completeBtn.disabled = !canComplete;

}

if (state.townCrierQuest.active) {

const mission = state.townCrierQuest;

const el = document.createElement('div');

el.className = 'quest-item';

el.style.borderLeftColor = 'var(--info-color)';

const reqString = Object.entries(mission.requirements)

.map(([res, count]) => `${count} ${res.replace(/([A-Z])/g, ' $1').trim()}`).join(', ');

el.innerHTML = `

<h3 class="text-lg font-bold">Town Announcement</h3>

<p class="text-sm mb-2">${mission.prompt}</p>

<button id="town-crier-complete-btn" class="quest-button w-full">

<span>Complete Quest</span>

<span class="cost">Requires: ${reqString}</span>

</button>

`;

container.appendChild(el);

const completeBtn = document.getElementById('town-crier-complete-btn');

completeBtn.onclick = completeTownCrierQuest;

let canComplete = true;

for (const res in mission.requirements) {

if ((state.resources[res] || 0) < mission.requirements[res]) {

canComplete = false;

break;

}

}

completeBtn.disabled = !canComplete;

}

// Regular quests

for (const key in CONFIG.quests) {

const quest = CONFIG.quests[key];

if (state.player.level >= quest.level && state.questStatus[key] === 'unavailable') {

state.questStatus[key] = 'available';

}

if (state.questStatus[key] !== 'unavailable') {

const el = document.createElement('div');

el.className = 'quest-item';

const isCompleted = state.questStatus[key] === 'completed';

if(isCompleted) el.classList.add('completed');

const costString = Object.entries(quest.req).map(([res, count]) => `${count} ${res.charAt(0).toUpperCase() + res.slice(1)}`).join(', ');

const rewardString = Object.entries(quest.reward).map(([res, count]) => {

if (res === 'traitPoints') return `${count} Trait Points`;

return `${count} ${res.replace(/([A-Z])/g, ' $1').trim()}`;

}).join(', ');

el.innerHTML = `<h3 class="text-lg font-bold">${quest.name}</h3><p class="text-sm mb-2">${quest.desc}</p><p class="text-xs mb-2">Reward: ${rewardString}</p>`;

if (!isCompleted) {

const btn = document.createElement('button');

btn.id = `quest-btn-${key}`;

btn.className = 'quest-button w-full';

btn.innerHTML = `<span>Complete Quest</span><span class="cost">Requires: ${costString}</span>`;

btn.onclick = () => completeQuest(key);

el.appendChild(btn);

} else {

el.innerHTML += '<div class="text-green-400 font-bold">COMPLETED</div>';

}

container.appendChild(el);

}

}

// NPC quests

for (const npcKey in CONFIG.npcs) {

const npcData = CONFIG.npcs[npcKey];

if (!npcData.quests) continue;

for (const questKey in npcData.quests) {

if (state.questStatus[questKey] === 'available' || state.questStatus[questKey] === 'completed') {

const el = document.createElement('div');

el.className = 'quest-item';

const isCompleted = state.questStatus[questKey] === 'completed';

if(isCompleted) el.classList.add('completed');

const quest = npcData.quests[questKey];

const costString = Object.entries(quest.req).map(([res, count]) => `${count} ${res.charAt(0).toUpperCase() + res.slice(1)}`).join(', ');

const rewardString = Object.entries(quest.reward).map(([res, count]) => `${count} ${res.replace(/([A-Z])/g, ' $1').trim()}`).join(', ');

el.innerHTML = `<h3 class="text-lg font-bold">NPC: ${npcData.name}'s Request: ${quest.name}</h3><p class="text-sm mb-2">${quest.desc}</p><p class="text-xs mb-2">Reward: ${rewardString}</p>`;

if (!isCompleted) {

const btn = document.createElement('button');

btn.id = `quest-btn-${questKey}`;

btn.className = 'quest-button w-full';

btn.innerHTML = `<span>Complete Quest</span><span class="cost">Requires: ${costString}</span>`;

btn.onclick = () => initiateDialogue(npcKey);

el.appendChild(btn);

} else {

el.innerHTML += '<div class="text-green-400 font-bold">COMPLETED</div>';

}

container.appendChild(el);

}

}

}

for (const key in CONFIG.quests) {

if (state.questStatus[key] === 'available') {

const quest = CONFIG.quests[key];

const btn = document.getElementById(`quest-btn-${key}`);

if(btn) {

let canAfford = true;

for (const res in quest.req) {

if (state.resources[res] < quest.req[res]) {

canAfford = false;

break;

}

}

btn.disabled = !canAfford;

}

}

}

for (const npcKey in CONFIG.npcs) {

if (!CONFIG.npcs[npcKey].quests) continue;

for (const questKey in CONFIG.npcs[npcKey].quests) {

if (state.questStatus[questKey] === 'available') {

const quest = CONFIG.npcs[npcKey].quests[questKey];

const btn = document.getElementById(`quest-btn-${questKey}`);

if(btn) {

let canAfford = true;

for (const res in quest.req) {

if (state.resources[res] < quest.req[res]) {

canAfford = false;

break;

}

}

btn.disabled = !canAfford;

}

}

}

}

updateNPCVisibility();

}

function completeQuest(key) {

const quest = CONFIG.quests[key];

if (!quest || state.questStatus[key] !== 'available') return;

for (const res in quest.req) {

if (state.resources[res] < quest.req[res]) {

showMessage(`Not enough resources for ${quest.name}!`, 'error');

return;

}

}

for (const res in quest.req) {

state.resources[res] -= quest.req[res];

}

for (const rew in quest.reward) {

if (rew === 'xp') addXp(quest.reward[rew]);

else if (rew === 'traitPoints') state.player.traitPoints += quest.reward[rew];

else state.resources[rew] += quest.reward[rew];

}

state.questStatus[key] = 'completed';

showMessage(`Quest Complete: ${quest.name}!`, 'bonus');

playSound(sfx.powerup);

updateQuestsUI();

}

function updateTraitsUI() {

const container = document.getElementById('trait-grid');

container.innerHTML = Object.keys(CONFIG.traits).map(key => {

const trait = CONFIG.traits[key];

const level = state.player.traits[key];

return `

<div class="trait-item">

<div class="flex justify-between items-center">

<div>

<h3 class="text-lg font-bold">${trait.emoji} ${trait.name} [${level}]</h3>

<p class="text-xs">${trait.desc}</p>

</div>

<button class="upgrade-btn" data-trait="${key}" ${state.player.traitPoints > 0 ? '' : 'disabled'}>+</button>

</div>

</div>`;

}).join('');

container.querySelectorAll('.upgrade-btn').forEach(btn => {

btn.onclick = () => spendTraitPoint(btn.dataset.trait);

});

}

function spendTraitPoint(traitKey) {

if (state.player.traitPoints > 0) {

state.player.traitPoints--;

state.player.traits[traitKey]++;

playSound(sfx.powerup);

updateTraitsUI();

updateDynamicButtons();

} else {

playSound(sfx.error);

}

}

// --- Utility Functions ---

function moveCharacterToNode(characterEl, nodeElement) {

const worldRect = gameWorld.getBoundingClientRect();

const nodeRect = nodeElement.getBoundingClientRect();

// Calculate position relative to the game world container

const targetLeft = nodeRect.left - worldRect.left;

const targetBottom = worldRect.bottom - nodeRect.bottom;

characterEl.style.left = `${targetLeft - 20}px`; // Position character slightly to the left of the node

characterEl.style.bottom = `${targetBottom}px`;

}

function updateNodeHealthBar(node) {

const percentage = (node.health / node.maxHealth) \* 100;

node.healthBar.style.width = `${percentage}%`;

}

function respawnNode(node) {

setTimeout(() => {

const pos = getSafeSpawnPosition({ width: 50, height: 50 });

node.element.style.left = `${pos.left}px`;

node.element.style.bottom = `${pos.bottom}px`;

node.element.classList.remove('hidden-despawning');

node.element.classList.add('spawning');

node.health = node.maxHealth;

updateNodeHealthBar(node);

setTimeout(() => node.element.classList.remove('spawning'), 500);

}, node.tier.respawnTime);

}

function sellAll() {

let goldEarned = 0;

let rizzBonus = 1 + (state.player.traits.rizz \* 0.02);

let multiplier = (state.goldBoostEndTime > Date.now() ? 2 : 1) \* rizzBonus;

goldEarned += state.resources.wood \* CONFIG.prices.wood;

goldEarned += state.resources.stone \* CONFIG.prices.stone;

goldEarned += state.resources.planks \* CONFIG.prices.planks;

if (state.discountNextPurchase > 0) {

goldEarned \*= (1 + state.discountNextPurchase);

state.discountNextPurchase = 0;

showMessage("Molly's discount applied!", "info");

}

goldEarned = Math.floor(goldEarned \* multiplier);

if (goldEarned > 0) {

state.resources.gold += goldEarned;

state.stats.totalGoldEarned += goldEarned;

state.resources.wood = 0;

state.resources.stone = 0;

state.resources.planks = 0;

addXp(goldEarned \* CONFIG.xpPerGold);

playSound(sfx.coin);

showMessage(`Sold resources for ${goldEarned} Gold!`, 'success');

checkAchievements();

} else {

showMessage('No resources to sell.', 'error');

playSound(sfx.error);

}

}

function upgradeTool(tool) {

const cost = CONFIG.baseUpgradeCost \* Math.pow(2, state.tools[tool] - 1);

if(state.resources.gold >= cost) {

state.resources.gold -= cost;

state.tools[tool]++;

showMessage(`${tool.charAt(0).toUpperCase() + tool.slice(1)} upgraded to Lvl ${state.tools[tool]}!`, 'success');

playSound(sfx.build);

updateDynamicButtons();

} else {

showMessage(`Not enough gold. Need ${cost}G.`, 'error');

playSound(sfx.error);

}

}

function buildOrUpgradeBuilding(key) {

const building = CONFIG.buildings[key];

const level = state.buildings[key];

let cost = {};

if (level === 0) {

cost = building.baseCost;

} else {

for (const res in building.baseCost) {

cost[res] = Math.floor(building.baseCost[res] \* Math.pow(building.upgradeCostMultiplier, level));

}

}

for (const res in cost) {

if (state.resources[res] < cost[res]) {

showMessage(`Not enough ${res}!`, 'error');

playSound(sfx.error);

return;

}

}

for (const res in cost) {

state.resources[res] -= cost[res];

}

state.buildings[key]++;

const action = level === 0 ? 'Built' : 'Upgraded';

showMessage(`${action} ${building.name} to Level ${state.buildings[key]}!`, 'success');

playSound(sfx.build);

const buildingEl = document.getElementById(`${key}-building`);

if (buildingEl) buildingEl.classList.remove('hidden');

updateBuildingUI();

}

function upgradeEpicItem(key) {

const item = CONFIG.collection[key];

const level = state.collection[`${key}Level`];

let currentCost = {};

if (level === 0) {

currentCost = item.baseCost;

} else {

for (const res in item.baseCost) {

currentCost[res] = Math.floor(item.baseCost[res] \* Math.pow(item.upgradeCostMultiplier, level));

}

}

for (const res in currentCost) {

if (state.resources[res] < currentCost[res]) {

showMessage(`Not enough ${res}!`, 'error');

playSound(sfx.error);

return;

}

}

for (const res in currentCost) {

state.resources[res] -= currentCost[res];

}

state.collection[`${key}Level`]++;

const action = level === 0 ? 'Crafted' : 'Upgraded';

showMessage(`${action} the ${item.name} to Level ${state.collection[`${key}Level`]}!`, 'success');

playSound(sfx.build);

updateInventoryPanelUI();

}

function showFloatingText(text, element, isCrit = false) {

const textEl = document.createElement('div');

textEl.classList.add('floating-text');

if (isCrit) textEl.classList.add('crit');

textEl.textContent = text;

const rect = element.getBoundingClientRect();

const worldRect = gameWorld.getBoundingClientRect();

textEl.style.left = `${rect.left - worldRect.left + rect.width / 2 - 20}px`;

textEl.style.top = `${rect.top - worldRect.top - 20}px`;

gameWorld.appendChild(textEl);

setTimeout(() => textEl.remove(), isCrit ? 2000 : 1500);

}

function getSafeSpawnPosition(nodeSize) {

const worldRect = gameWorld.getBoundingClientRect();

const groundLevel = worldRect.height \* 0.3;

let attempts = 0;

let pos = { x: 0, y: 0 };

let isSafe = false;

const playerCharEls = document.querySelectorAll('.character:not(.npc-character)');

const playerCharRects = Array.from(playerCharEls).map(el => el.getBoundingClientRect());

while (!isSafe && attempts < 50) {

isSafe = true;

pos.x = Math.random() \* (worldRect.width - nodeSize.width);

pos.y = Math.random() \* (worldRect.height - groundLevel - nodeSize.height) + groundLevel;

const nodeRect = {

left: pos.x,

right: pos.x + nodeSize.width,

top: pos.y,

bottom: pos.y + nodeSize.height

};

const allObjects = [

...resourceNodes.map(n => n.element),

...state.activeEnemies.map(e => e.element),

...Array.from(document.querySelectorAll('.building, .npc-character'))

];

for (const playerRect of playerCharRects) {

if (rectsOverlap(nodeRect, playerRect)) {

isSafe = false;

break;

}

}

if (!isSafe) {

attempts++;

continue;

}

for (const otherEl of allObjects) {

if (rectsOverlap(nodeRect, otherEl.getBoundingClientRect())) {

isSafe = false;

break;

}

}

attempts++;

}

return { left: pos.x, bottom: worldRect.height - (pos.y + nodeSize.height) };

}

function rectsOverlap(r1, r2) {

return !(r2.left > r1.right || r2.right < r1.left || r2.top > r1.bottom || r2.bottom < r1.top);

}

function checkAchievements() {

for (const key in CONFIG.achievements) {

const ach = CONFIG.achievements[key];

if (!ach.completed && ach.goal()) {

ach.completed = true;

showMessage(`Achievement: ${ach.name}!`, 'bonus');

}

}

}

function createNode(type, tier) {

const node = {

type: type,

tier: tier,

clickHandler: type === 'tree' ? handleChop : handleMine,

element: document.createElement('div'),

maxHealth: tier.health,

health: tier.health,

};

node.element.classList.add('game-object');

node.element.textContent = tier.emoji;

const healthBarContainer = document.createElement('div');

healthBarContainer.className = 'node-health-bar-container';

const healthBar = document.createElement('div');

healthBar.className = 'node-health-bar';

healthBarContainer.appendChild(healthBar);

node.element.appendChild(healthBarContainer);

node.healthBar = healthBar;

node.element.addEventListener('click', () => node.clickHandler(node));

const pos = getSafeSpawnPosition({ width: 50, height: 50 });

node.element.style.left = `${pos.left}px`;

node.element.style.bottom = `${pos.bottom}px`;

node.element.classList.add('spawning');

setTimeout(() => node.element.classList.remove('spawning'), 500);

gameWorld.appendChild(node.element);

resourceNodes.push(node);

return node;

}

function updateAbilityUI() {

const startContainer = document.getElementById('ability-tree-start');

const branchesContainer = document.getElementById('ability-tree-branches');

startContainer.innerHTML = '';

branchesContainer.innerHTML = '';

const renderAbility = (key) => {

const ability = CONFIG.abilities[key];

const currentLevel = state.abilities[key] || 0;

let cost = {};

if (key === 'honorableWorker' || key === 'lumberjackPath' || key === 'minerPath') {

cost = ability.cost;

} else {

cost.gold = Math.floor(ability.cost.gold \* Math.pow(1.8, currentLevel));

cost.ap = ability.cost.ap;

}

let canBuy = currentLevel < ability.max;

if (state.player.abilityPoints < (cost.ap || 0)) canBuy = false;

for(const res in cost) {

if (res !== 'ap' && state.resources[res] < cost[res]) canBuy = false;

}

if (ability.requires) {

const req = CONFIG.abilities[ability.requires];

if (!state.abilities[ability.requires] || state.abilities[ability.requires] < req.max) {

canBuy = false;

}

}

if (key === 'lumberjackPath' && state.abilities.minerPath > 0) return '';

if (key === 'minerPath' && state.abilities.lumberjackPath > 0) return '';

if ((key.startsWith('wood') || key.startsWith('forest') || key.startsWith('chain')) && state.abilities.minerPath > 0) return '';

if ((key.startsWith('stone') || key.startsWith('earth') || key.startsWith('jack')) && state.abilities.lumberjackPath > 0) return '';

const costString = Object.entries(cost).map(([k, v]) => `${v} ${k}`).join(', ');

return `<button class="ability-button" data-ability-key="${key}" ${canBuy ? '' : 'disabled'}><span>${ability.name} (${currentLevel}/${ability.max})</span><span class="cost">${ability.desc}</span><span class="cost">Cost: ${costString}</span></button>`;

};

startContainer.innerHTML = renderAbility('honorableWorker');

if (state.abilities.honorableWorker >= 1) {

if (state.abilities.lumberjackPath === 0 && state.abilities.minerPath === 0) {

branchesContainer.innerHTML = `<div class="ability-branch text-center">${renderAbility('lumberjackPath')}</div><div class="ability-branch text-center">${renderAbility('minerPath')}</div>`;

} else if (state.abilities.lumberjackPath > 0) {

branchesContainer.innerHTML = `<div class="ability-branch space-y-2"><h3 class="text-lg text-center text-green-400">Lumberjack</h3>${renderAbility('woodExpertise')}${renderAbility('forestsBlessing')}${renderAbility('splinteringStrikes')}${renderAbility('chainsawMastery')}</div>`;

} else if (state.abilities.minerPath > 0) {

branchesContainer.innerHTML = `<div class="ability-branch space-y-2"><h3 class="text-lg text-center text-gray-400">Miner</h3>${renderAbility('stoneExpertise')}${renderAbility('earthsBounty')}${renderAbility('shatteringBlows')}${renderAbility('jackhammerMastery')}</div>`;

}

}

document.querySelectorAll('.ability-button').forEach(btn => btn.onclick = () => buyAbility(btn.dataset.abilityKey));

}

function handleChop(node) {

if (node.element.classList.contains('hidden-despawning')) return;

let damage = state.tools.baseDamage + state.player.traits.strength + (state.abilities.honorableWorker > 0 ? 1 : 0) + (state.collection.goldenAxeLevel \* 5);

handleHit(node, damage, lumberjackEl);

}

function handleMine(node) {

if (node.element.classList.contains('hidden-despawning')) return;

let damage = state.tools.baseDamage + state.player.traits.strength + (state.abilities.honorableWorker > 0 ? 1 : 0);

handleHit(node, damage, minerEl);

}

function buyAbility(key) {

const ability = CONFIG.abilities[key];

const currentLevel = state.abilities[key] || 0;

if (currentLevel >= ability.max) return;

let cost = {};

if (key === 'honorableWorker' || key === 'lumberjackPath' || key === 'minerPath') {

cost = ability.cost;

} else {

cost.gold = Math.floor(ability.cost.gold \* Math.pow(1.8, currentLevel));

cost.ap = ability.cost.ap;

}

if (state.player.abilityPoints < (cost.ap || 0)) {

showMessage('Not enough Ability Points!', 'error');

return;

}

for(const res in cost) {

if (res !== 'ap' && state.resources[res] < cost[res]) {

showMessage(`Not enough ${res}!`, 'error');

return;

}

}

if (ability.requires) {

const req = CONFIG.abilities[ability.requires];

if (!state.abilities[ability.requires] || state.abilities[ability.requires] < req.max) {

showMessage('Requires max level of previous ability!', 'error');

return;

}

}

state.player.abilityPoints -= (cost.ap || 0);

for(const res in cost) {

if (res !== 'ap') state.resources[res] -= cost[res];

}

state.abilities[key]++;

playSound(sfx.build);

updateAbilityUI();

}

// --- Map System Functions ---

function updateMapUI() {

const mapGrid = document.getElementById('map-grid');

const currentLocationDisplay = document.getElementById('current-location-display').querySelector('span');

mapGrid.innerHTML = '';

currentLocationDisplay.textContent = CONFIG.map[state.player.currentLocation].name;

const currentLocationData = CONFIG.map[state.player.currentLocation];

const currentLocDiv = document.createElement('div');

currentLocDiv.className = 'p-4 rounded bg-blue-700 border-2 border-white';

currentLocDiv.innerHTML = `<p class="text-xl font-bold">${currentLocationData.emoji} ${currentLocationData.name}</p><p class="text-sm">${currentLocationData.description}</p>`;

mapGrid.appendChild(currentLocDiv);

currentLocationData.connections.forEach(connKey => {

const connData = CONFIG.map[connKey];

const connDiv = document.createElement('button');

connDiv.className = 'action-button p-4 rounded bg-green-700 border-2 border-white clickable';

connDiv.innerHTML = `<p class="text-xl font-bold">${connData.emoji} Go to ${connData.name}</p><p class="text-xs">${connData.description}</p>`;

connDiv.onclick = () => travelToLocation(connKey);

mapGrid.appendChild(connDiv);

});

}

function travelToLocation(locationKey) {

if (state.combat.active) {

showMessage("Cannot travel during combat!", "error");

return;

}

gameWorld.querySelectorAll('.game-object').forEach(n => n.remove());

gameWorld.querySelectorAll('.npc-character').forEach(n => n.remove());

resourceNodes = [];

state.activeEnemies.forEach(e => clearTimeout(e.despawnTimer));

state.activeEnemies = [];

updateActiveEnemiesListUI();

state.player.currentLocation = locationKey;

showMessage(`You arrived at ${CONFIG.map[locationKey].name}!`, 'info');

updateMapUI();

createWorldObjects();

spawnNPCsInWorld();

const currentLocationData = CONFIG.map[locationKey];

if (Math.random() < currentLocationData.encounterChance) {

const availableEnemies = currentLocationData.potentialEncounters;

if (availableEnemies.length > 0) {

const randomEnemyKey = availableEnemies[Math.floor(Math.random() \* availableEnemies.length)];

if (CONFIG.enemies[randomEnemyKey].type === 'enemy' || CONFIG.enemies[randomEnemyKey].type === 'boss') {

spawnEnemy(randomEnemyKey);

}

}

} else if (currentLocationData.bossEncounter && Math.random() < 0.05) {

spawnEnemy(currentLocationData.bossEncounter);

}

}

function spawnNPCsInWorld() {

gameWorld.querySelectorAll('.npc-character').forEach(n => n.remove());

const currentLocationData = CONFIG.map[state.player.currentLocation];

if (currentLocationData.npcs) {

currentLocationData.npcs.forEach(npcKey => {

const npcData = CONFIG.npcs[npcKey];

const npcEl = document.createElement('div');

npcEl.id = `npc-${npcKey}`; // Assign an ID for visibility toggling

npcEl.classList.add('character', 'npc-character', 'hidden'); // Start as hidden

npcEl.textContent = npcData.emoji;

const pos = getSafeSpawnPosition({ width: 50, height: 50 });

npcEl.style.left = `${pos.left}px`;

npcEl.style.bottom = `${pos.bottom}px`;

npcEl.onclick = () => initiateDialogue(npcKey);

gameWorld.appendChild(npcEl);

});

}

updateNPCVisibility(); // Set initial visibility

}

// --- NPC and Dialogue System Functions ---

function updateNPCVisibility() {

const isNpcTabActive = document.getElementById('npcs-panel').classList.contains('active');

const shouldBeVisible = {};

// Condition 1: If NPC tab is active, all NPCs at the current location should be visible.

if (isNpcTabActive) {

const npcsAtLocation = CONFIG.map[state.player.currentLocation].npcs || [];

npcsAtLocation.forEach(npcKey => {

shouldBeVisible[npcKey] = true;

});

}

// Condition 2: If an NPC has an available quest, they should be visible.

for (const npcKey in CONFIG.npcs) {

const npcData = CONFIG.npcs[npcKey];

if (npcData.quests) {

for (const questKey in npcData.quests) {

if (state.questStatus[questKey] === 'available') {

shouldBeVisible[npcKey] = true;

}

}

}

}

// Apply visibility to all NPCs

for (const npcKey in CONFIG.npcs) {

const npcEl = document.getElementById(`npc-${npcKey}`);

if (npcEl) {

const isAtLocation = (CONFIG.map[state.player.currentLocation].npcs || []).includes(npcKey);

if (shouldBeVisible[npcKey] && isAtLocation) {

npcEl.classList.remove('hidden');

} else {

npcEl.classList.add('hidden');

}

}

}

}

function updateNPCsUI() {

const npcList = document.getElementById('npc-list');

npcList.innerHTML = '';

const npcsAtLocation = CONFIG.map[state.player.currentLocation].npcs || [];

if (npcsAtLocation.length === 0) {

npcList.innerHTML = '<p class="text-center text-gray-400">No NPCs in this area.</p>';

return;

}

npcsAtLocation.forEach(npcKey => {

const npcData = CONFIG.npcs[npcKey];

const relationship = state.npcs[npcKey].relationship;

let relStatus = "Neutral";

if (relationship > 50) relStatus = "Friendly";

else if (relationship > 20) relStatus = "Good";

else if (relationship < -50) relStatus = "Hostile";

else if (relationship < -20) relStatus = "Unfriendly";

const npcDiv = document.createElement('div');

npcDiv.className = 'family-member';

let buttonText = `Talk to ${npcData.name}`;

if (npcKey === 'townCrier') {

buttonText = '✨ Hear Announcement';

}

npcDiv.innerHTML = `

<h3 class="text-lg font-bold">${npcData.emoji} ${npcData.name}</h3>

<p class="text-sm">Relationship: ${relStatus} (${relationship})</p>

<button class="action-button mt-2" onclick="initiateDialogue('${npcKey}')">${buttonText}</button>

`;

npcList.appendChild(npcDiv);

});

}

async function initiateDialogue(npcKey) {

const npcData = CONFIG.npcs[npcKey];

const npcState = state.npcs[npcKey];

const dialogueModal = document.getElementById('npc-dialogue-modal');

const dialogueNpcName = document.getElementById('dialogue-npc-name');

const dialogueText = document.getElementById('dialogue-text');

const dialogueOptions = document.getElementById('dialogue-options');

dialogueNpcName.textContent = npcData.emoji + " " + npcData.name;

dialogueOptions.innerHTML = '';

dialogueText.textContent = '...';

dialogueModal.classList.remove('hidden');

// Special handler for Town Crier to generate quests

if (npcKey === 'townCrier') {

if (state.townCrierQuest.active) {

dialogueText.textContent = "I've already given you a task! Check your Quests tab.";

dialogueOptions.innerHTML = '';

} else {

dialogueText.textContent = "The Town Crier is clearing his throat...";

await generateTownCrierQuest();

dialogueText.textContent = state.townCrierQuest.prompt;

const acceptBtn = document.createElement('button');

acceptBtn.className = 'action-button';

acceptBtn.textContent = 'Accept Quest';

acceptBtn.onclick = () => {

showMessage("Quest accepted! Check your Quests tab.", "bonus");

closeDialogueModal();

};

dialogueOptions.appendChild(acceptBtn);

}

return;

}

// Standard quest checks (like for Old Man Jenkins)

if (npcData.quests) {

for (const questKey in npcData.quests) {

const quest = npcData.quests[questKey];

if (state.questStatus[questKey] === 'available' && npcState.quests[questKey].status !== 'completed') {

let canCompleteNow = true;

for (const res in quest.req) {

if (state.resources[res] < quest.req[res]) {

canCompleteNow = false;

break;

}

}

if (canCompleteNow) {

const chosenDialogue = npcData.dialogue.find(d => d.type === 'quest\_complete');

if (chosenDialogue) {

dialogueText.textContent = chosenDialogue.text;

chosenDialogue.options.forEach(option => {

const optionBtn = document.createElement('button');

optionBtn.className = 'action-button';

optionBtn.textContent = option.text;

optionBtn.onclick = () => handleDialogueOption(npcKey, option);

dialogueOptions.appendChild(optionBtn);

});

return;

}

}

}

if (state.questStatus[questKey] === 'available' && npcState.quests[questKey].status === 'available') {

const chosenDialogue = npcData.dialogue.find(d => d.type === 'quest\_available');

if (chosenDialogue) {

dialogueText.textContent = chosenDialogue.text;

chosenDialogue.options.forEach(option => {

const optionBtn = document.createElement('button');

optionBtn.className = 'action-button';

optionBtn.textContent = option.text;

optionBtn.onclick = () => handleDialogueOption(npcKey, option);

dialogueOptions.appendChild(optionBtn);

});

return;

}

}

}

}

// Gemini-powered dynamic dialogue for other NPCs

if (npcData.persona) {

const playerState = `My level is ${state.player.level}. I have ${state.resources.wood} wood, ${state.resources.stone} stone, and ${state.resources.gold} gold. My relationship with you is ${npcState.relationship}.`;

const prompt = `${npcData.persona} The player is talking to you. Based on this persona and the player's current state, generate a short, engaging line of dialogue. Player state: ${playerState}`;

const generatedText = await callGemini(prompt);

dialogueText.textContent = generatedText || "They seem lost in thought.";

} else {

// Fallback to initial scripted dialogue if no persona

const chosenDialogue = npcData.dialogue.find(d => d.type === 'initial');

if (chosenDialogue) {

dialogueText.textContent = chosenDialogue.text;

chosenDialogue.options.forEach(option => {

const optionBtn = document.createElement('button');

optionBtn.className = 'action-button';

optionBtn.textContent = option.text;

optionBtn.onclick = () => handleDialogueOption(npcKey, option);

dialogueOptions.appendChild(optionBtn);

});

} else {

dialogueText.textContent = "They have nothing to say.";

}

}

}

function handleDialogueOption(npcKey, option) {

const npcState = state.npcs[npcKey];

const dialogueText = document.getElementById('dialogue-text');

if (option.effect.relationship) {

npcState.relationship += option.effect.relationship;

npcState.relationship = Math.max(-100, Math.min(100, npcState.relationship));

}

showMessage(option.response, 'info');

if (option.effect.startQuest) {

state.questStatus[option.effect.startQuest] = 'available';

if (npcState.quests && npcState.quests[option.effect.startQuest]) {

npcState.quests[option.effect.startQuest].status = 'available';

}

dialogueText.textContent = option.response;

updateQuestsUI();

}

if (option.effect.completeQuest) {

const questKey = option.effect.completeQuest;

const quest = CONFIG.npcs[npcKey].quests[questKey];

if (quest) {

let hasRequirements = true;

for (const res in quest.req) {

if (state.resources[res] < quest.req[res]) {

hasRequirements = false;

showMessage(`You don't have enough ${res} for this!`, 'error');

break;

}

}

if (hasRequirements) {

for (const res in quest.req) {

state.resources[res] -= quest.req[res];

}

for (const rew in quest.reward) {

if (rew === 'xp') addXp(quest.reward[rew]);

else if (rew === 'traitPoints') state.player.traitPoints += quest.reward[rew];

else state.resources[rew] += quest.reward[rew];

}

state.questStatus[questKey] = 'completed';

npcState.quests[questKey].status = 'completed';

dialogueText.textContent = option.response + " You received rewards!";

showMessage(`Quest Complete: ${quest.name}!`, 'bonus');

playSound(sfx.powerup);

updateQuestsUI();

setTimeout(closeDialogueModal, 1500);

return;

}

}

}

if (option.effect.discountNextPurchase) {

state.discountNextPurchase = option.effect.discountNextPurchase;

showMessage("Next sale will be discounted!", "bonus");

}

if (option.response) {

dialogueText.textContent = option.response;

setTimeout(() => closeDialogueModal(), 1500);

} else {

closeDialogueModal();

}

updateNPCsUI();

}

function closeDialogueModal() {

document.getElementById('npc-dialogue-modal').classList.add('hidden');

}

// --- Combat System Functions ---

function updateActiveEnemiesListUI() {

const activeEnemiesListEl = document.getElementById('active-enemies-list');

const noActiveEnemiesMessageEl = document.getElementById('no-active-enemies-message');

if (!activeEnemiesListEl || !noActiveEnemiesMessageEl) {

return; // Exit if the elements don't exist

}

activeEnemiesListEl.innerHTML = ''; // Clear only the list

const combatThreats = state.activeEnemies.filter(enemy =>

CONFIG.enemies[enemy.key] && (CONFIG.enemies[enemy.key].type === 'enemy' || CONFIG.enemies[enemy.key].type === 'boss')

);

if (combatThreats.length === 0) {

noActiveEnemiesMessageEl.classList.remove('hidden');

} else {

noActiveEnemiesMessageEl.classList.add('hidden');

combatThreats.forEach(enemyInstance => {

const enemyData = CONFIG.enemies[enemyInstance.key];

const enemyDiv = document.createElement('div');

enemyDiv.className = 'quest-item';

enemyDiv.innerHTML = `

<h3 class="text-lg font-bold">${enemyData.emoji} ${enemyData.name || enemyInstance.key.replace(/([A-Z])/g, ' $1').trim()}</h3>

<p class="text-sm mb-2">Threat Level: ${state.player.level}x Scaled</p>

<button class="action-button mt-2" data-enemy-id="${enemyInstance.id}" onclick="initiateCombatFromList(this.dataset.enemyId)">Fight!</button>

`;

activeEnemiesListEl.appendChild(enemyDiv);

});

}

}

function initiateCombatFromList(enemyInstanceId) {

if (state.combat.active) {

showMessage("You are already in combat!", "error");

return;

}

const enemyInstance = state.activeEnemies.find(e => e.id === enemyInstanceId);

if (!enemyInstance) {

showMessage("Enemy not found.", "error");

updateActiveEnemiesListUI();

return;

}

if (enemyInstance.despawnTimer) clearTimeout(enemyInstance.despawnTimer);

if (enemyInstance.element && enemyInstance.element.parentElement) {

enemyInstance.element.remove();

}

const index = state.activeEnemies.findIndex(e => e.id === enemyInstanceId);

if (index > -1) {

state.activeEnemies.splice(index, 1);

}

updateActiveEnemiesListUI();

startCombat(enemyInstance.key);

}

function startCombat(enemyKey) {

const enemyBaseData = CONFIG.enemies[enemyKey];

if (!enemyBaseData || !enemyBaseData.combat) {

showMessage(`Error: Combat data not found for ${enemyKey}.`, 'error');

return;

}

const playerLevel = state.player.level;

const scaledHealth = Math.floor(enemyBaseData.combat.health \* playerLevel);

const scaledAttack = Math.floor(enemyBaseData.combat.attack \* playerLevel);

const scaledDefense = Math.floor(enemyBaseData.combat.defense \* playerLevel);

const scaledXpReward = Math.floor(enemyBaseData.combat.xpReward \* playerLevel);

const scaledGoldReward = Math.floor(enemyBaseData.combat.goldReward \* playerLevel);

state.combat.active = true;

state.combat.turnCount = 0;

state.combat.currentEnemy = {

key: enemyKey,

name: enemyBaseData.name || enemyKey.replace(/([A-Z])/g, ' $1').trim(),

emoji: enemyBaseData.emoji,

maxHealth: scaledHealth,

currentHealth: scaledHealth,

attack: scaledAttack,

defense: scaledDefense,

xpReward: scaledXpReward,

goldReward: scaledGoldReward,

specialDrop: enemyBaseData.combat.specialDrop,

dodgeChance: enemyBaseData.combat.dodgeChance,

critChance: enemyBaseData.combat.critChance,

specialMove: enemyBaseData.combat.specialMove,

};

state.combat.log = [];

state.combat.fleeAttempts = 0;

state.combat.playerTurn = true;

document.getElementById('combat-enemy-name').textContent = ` ${state.combat.currentEnemy.emoji} ${state.combat.currentEnemy.name}`;

updateCombatUI();

document.getElementById('combat-modal').classList.remove('hidden');

addCombatLog(`A wild ${state.combat.currentEnemy.name} appeared!`);

}

function updateCombatUI() {

const playerHealthBar = document.getElementById('player-combat-health-bar');

const playerHealthText = document.getElementById('player-combat-health-text');

const enemyHealthBar = document.getElementById('enemy-combat-health-bar');

const enemyHealthText = document.getElementById('enemy-combat-health-text');

const combatLogEl = document.getElementById('combat-log');

playerHealthBar.style.width = `${(state.player.combat.currentHealth / state.player.combat.health) \* 100}%`;

playerHealthText.textContent = `${state.player.combat.currentHealth}/${state.player.combat.health}`;

enemyHealthBar.style.width = `${(state.combat.currentEnemy.currentHealth / state.combat.currentEnemy.maxHealth) \* 100}%`;

enemyHealthText.textContent = `${state.combat.currentEnemy.currentHealth}/${state.combat.currentEnemy.maxHealth}`;

combatLogEl.innerHTML = state.combat.log.map(entry => `<div>${entry}</div>`).join('');

combatLogEl.scrollTop = combatLogEl.scrollHeight;

}

function addCombatLog(message) {

state.combat.log.push(message);

if (state.combat.log.length > 10) {

state.combat.log.shift();

}

updateCombatUI();

}

function selectWeapon(weapon) {

state.combat.selectedWeapon = weapon;

document.getElementById('select-axe').classList.toggle('active', weapon === 'axe');

document.getElementById('select-pickaxe').classList.toggle('active', weapon === 'pickaxe');

}

function playerAttack() {

if (!state.combat.active || !state.combat.playerTurn) return;

if (Math.random() < state.combat.currentEnemy.dodgeChance) {

addCombatLog(`${state.combat.currentEnemy.name} dodged your attack!`);

playSound(sfx.error);

state.combat.playerTurn = false;

setTimeout(enemyAttack, 1000);

return;

}

let playerDamage = state.player.combat.attack;

if(state.combat.selectedWeapon === 'axe') {

playerDamage += state.tools.axe;

} else {

playerDamage += state.tools.pickaxe;

}

const playerCritChance = state.player.traits.anime \* 0.01;

let isCrit = false;

if (Math.random() < playerCritChance) {

playerDamage \*= (1.5 + (state.player.traits.anime \* 0.1));

isCrit = true;

}

if (state.damageBoostEndTime > Date.now()) playerDamage \*= 2;

if (state.bentoBoostEndTime > Date.now()) playerDamage \*= 5;

playerDamage = Math.max(1, Math.floor(playerDamage - state.combat.currentEnemy.defense));

state.combat.currentEnemy.currentHealth -= playerDamage;

addCombatLog(`You hit ${state.combat.currentEnemy.name} for ${playerDamage} damage!${isCrit ? ' (CRIT!)' : ''}`);

playSound(sfx.chop);

updateCombatUI();

if (state.combat.currentEnemy.currentHealth <= 0) {

endCombat(true);

} else {

state.combat.playerTurn = false;

setTimeout(enemyAttack, 1000);

}

}

function enemyAttack() {

if (!state.combat.active || state.combat.playerTurn) return;

const playerDodgeChance = state.player.traits.agility \* 0.005;

if (Math.random() < playerDodgeChance) {

addCombatLog(`You dodged ${state.combat.currentEnemy.name}'s attack!`);

playSound(sfx.powerup);

state.combat.playerTurn = true;

updateCombatUI();

return;

}

let enemyDamage = state.combat.currentEnemy.attack;

let isCrit = false;

if (Math.random() < state.combat.currentEnemy.critChance) {

enemyDamage \*= 1.5;

isCrit = true;

}

enemyDamage = Math.max(1, Math.floor(enemyDamage - state.player.combat.defense));

state.player.combat.currentHealth -= enemyDamage;

addCombatLog(`${state.combat.currentEnemy.name} hits you for ${enemyDamage} damage!${isCrit ? ' (CRIT!)' : ''}`);

playSound(sfx.error);

if (state.combat.currentEnemy.specialMove && Math.random() < 0.25) {

handleEnemySpecialMove(state.combat.currentEnemy.specialMove);

}

state.combat.turnCount++;

if (state.combat.turnCount % 2 === 0) {

const healthRegen = 5;

state.player.combat.currentHealth = Math.min(state.player.combat.health, state.player.combat.currentHealth + healthRegen);

addCombatLog(`You regenerate ${healthRegen} health.`);

}

updateCombatUI();

if (state.player.combat.currentHealth <= 0) {

endCombat(false);

} else {

state.combat.playerTurn = true;

}

}

function playerFlee() {

if (!state.combat.active) return;

state.combat.fleeAttempts++;

const playerDodgeChance = state.player.traits.agility \* 0.01;

const fleeSuccessChance = 0.5 + playerDodgeChance;

if (Math.random() < fleeSuccessChance) {

addCombatLog("You successfully fled!");

endCombat(false, true);

} else {

addCombatLog("You failed to flee!");

state.combat.playerTurn = false;

setTimeout(enemyAttack, 1000);

}

}

function endCombat(playerWon, fled = false) {

state.combat.active = false;

document.getElementById('combat-modal').classList.add('hidden');

state.player.combat.currentHealth = state.player.combat.health;

if (fled) {

showMessage("You escaped the battle!", "info");

} else if (playerWon) {

const enemy = state.combat.currentEnemy;

addXp(enemy.xpReward);

state.resources.gold += enemy.goldReward;

showMessage(`You defeated the ${enemy.name}! Gained ${enemy.xpReward} XP and ${enemy.goldReward} Gold!`, 'success');

playSound(sfx.coin);

if (enemy.key === 'elderEnt') state.stats.elderEntKills++;

if (enemy.key === 'stoneTitan') state.stats.stoneTitanKills++;

if (enemy.key === 'ancientGolem') state.stats.ancientGolemKills++;

if (enemy.key === 'shadowSerpent') state.stats.shadowSerpentKills++;

if (enemy.specialDrop) {

if (!state.inventory[enemy.specialDrop]) state.inventory[enemy.specialDrop] = 0;

state.inventory[enemy.specialDrop]++;

showMessage(`You found a ${CONFIG.inventory[enemy.specialDrop].emoji || CONFIG.epicItems[enemy.specialDrop].name}!`, 'bonus');

}

} else {

showMessage("You were defeated! You lost some resources.", "error");

state.resources.gold = Math.max(0, state.resources.gold - 50);

state.player.xp = Math.max(0, state.player.xp - (CONFIG.xpForLevel(state.player.level) \* 0.1));

state.player.currentLocation = 'townSquare';

updateMapUI();

}

state.combat.currentEnemy = null;

updateDynamicButtons();

checkAchievements();

}

function handleEnemySpecialMove(moveType) {

switch (moveType) {

case 'slow':

state.slowdownEndTime = Date.now() + 5000;

addCombatLog(`${state.combat.currentEnemy.name} used SLOW! Your actions are slower!`);

break;

case 'stun':

state.combat.playerTurn = false;

addCombatLog(`${state.combat.currentEnemy.name} used STUN! You are stunned!`);

setTimeout(() => {

state.combat.playerTurn = true;

addCombatLog('You recovered from stun.');

updateCombatUI();

}, 2000);

break;

case 'roar':

state.player.combat.defense = Math.max(0, state.player.combat.defense - 2);

addCombatLog(`${state.combat.currentEnemy.name} roared! Your defense decreased!`);

break;

case 'stealGold':

const stolenGold = Math.min(state.resources.gold, 10);

state.resources.gold -= stolenGold;

addCombatLog(`${state.combat.currentEnemy.name} stole ${stolenGold} gold!`);

break;

case 'fear':

state.player.combat.attack = Math.max(1, state.player.combat.attack - 3);

addCombatLog(`${state.combat.currentEnemy.name} caused fear! Your attack decreased!`);

break;

case 'gnaw':

const woodLost = Math.min(state.resources.wood, 5);

state.resources.wood -= woodLost;

addCombatLog(`${state.combat.currentEnemy.name} gnawed away ${woodLost} wood!`);

break;

case 'poison':

addCombatLog(`${state.combat.currentEnemy.name} poisoned you!`);

break;

case 'sticky':

state.player.combat.attack = Math.max(1, state.player.combat.attack - 5);

addCombatLog(`${state.combat.currentEnemy.name} made you sticky! Your attack is severely reduced!`);

break;

case 'rootBind':

addCombatLog(`${state.combat.currentEnemy.name} used Root Bind! You cannot flee!`);

break;

case 'quake':

state.player.combat.currentHealth = Math.max(1, state.player.combat.currentHealth - 20);

addCombatLog(`${state.combat.currentEnemy.name} caused a Quake! You took heavy damage!`);

break;

case 'tremor':

const tremorDamage = Math.floor(state.player.combat.health \* 0.15);

state.player.combat.currentHealth = Math.max(1, state.player.combat.currentHealth - tremorDamage);

addCombatLog(`${state.combat.currentEnemy.name} unleashed a TREMOR! You took ${tremorDamage} damage!`);

break;

case 'darkness':

state.player.combat.attack = Math.max(1, Math.floor(state.player.combat.attack \* 0.5));

addCombatLog(`${state.combat.currentEnemy.name} shrouded you in DARKNESS! Your attack is halved!`);

break;

default:

addCombatLog(`${state.combat.currentEnemy.name} used a mysterious special move!`);

break;

}

updateCombatUI();

}

async function generateTownCrierQuest() {

const prompt = `You are a Town Crier in a quirky lumberjack game. Announce a new, simple quest for a level ${state.player.level} player. The quest should involve collecting basic resources like wood, stone, or planks. Create a short, dramatic announcement for the quest. Format the response as a single JSON object with "questText", "requirements", and "reward" keys. Requirements should be a JSON object mapping resource names to amounts. The reward should be a JSON object with gold and xp.`;

const schema = {

type: "OBJECT",

properties: {

"questText": { "type": "STRING" },

"requirements": { "type": "OBJECT" },

"reward": { "type": "OBJECT" }

},

required: ["questText", "requirements", "reward"]

};

const mission = await getGeminiPoweredMission(prompt, schema);

if (mission) {

state.townCrierQuest.active = true;

state.townCrierQuest.prompt = mission.questText;

state.townCrierQuest.requirements = mission.requirements;

state.townCrierQuest.reward = mission.reward;

updateQuestsUI();

} else {

showMessage("The Town Crier seems to have lost his voice. Try again later.", "error");

}

}

function completeTownCrierQuest() {

const mission = state.townCrierQuest;

if (!mission.active) return;

for (const res in mission.requirements) {

if ((state.resources[res] || 0) < mission.requirements[res]) {

showMessage('You do not have the required items for the Town Crier!', 'error');

return;

}

}

for (const res in mission.requirements) {

state.resources[res] -= mission.requirements[res];

}

for (const rew in mission.reward) {

if (rew === 'gold') state.resources.gold += mission.reward[rew];

if (rew === 'xp') addXp(mission.reward[rew]);

}

showMessage('Quest Complete! The Town Crier thanks you.', 'success');

playSound(sfx.powerup);

mission.active = false;

mission.completed = true; // Or reset it if you want repeatable quests

updateQuestsUI();

}

async function haveADream() {

const now = Date.now();

const twentyFourHours = 24 \* 60 \* 60 \* 1000;

if (now - state.lastDreamTime < twentyFourHours) {

showMessage("You've already recorded a dream today. Rest and try again tomorrow.", "info");

return;

}

const dreamBtn = document.getElementById('dream-btn');

const dreamOutput = document.getElementById('dream-output');

dreamBtn.disabled = true;

dreamOutput.classList.remove('hidden');

dreamOutput.textContent = 'You close your eyes and drift away...';

const prompt = `You are a dream interpreter in a quirky lumberjack idle game. The player is level ${state.player.level}. Generate a short, strange, and slightly surreal dream sequence for the player. The dream could be about anything from talking trees to rivers of gold. Keep it to 2-3 sentences.`;

const dreamText = await callGemini(prompt);

if (dreamText) {

dreamOutput.textContent = `Your dream: "${dreamText}"`;

// Give a small random buff for dreaming

const buffChance = Math.random();

if (buffChance < 0.3) {

state.damageBoostEndTime = Date.now() + 5 \* 60 \* 1000; // 5 min damage boost

showMessage("You feel inspired by your dream! (+Damage for 5 min)", "bonus");

} else if (buffChance < 0.6) {

state.goldBoostEndTime = Date.now() + 5 \* 60 \* 1000; // 5 min gold boost

showMessage("Your dream fills you with ambition! (+Gold for 5 min)", "bonus");

} else {

addXp(CONFIG.xpForLevel(state.player.level) \* 0.1); // 10% of next level XP

showMessage("You wake up feeling enlightened by your dream! (+XP)", "bonus");

}

} else {

dreamOutput.textContent = "You slept soundly, but remember no dreams.";

}

state.lastDreamTime = now;

dreamBtn.disabled = false;

}

// Make functions globally accessible for HTML onclick handlers

window.initiateCombatFromList = initiateCombatFromList;

window.initiateDialogue = initiateDialogue;

</script>

</body>

</html>