Html code starts here:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Power Inc.</title>

<link rel="stylesheet" href="style.css">

<link rel="preconnect" href="https://fonts.googleapis.com">

<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

<link href="https://fonts.googleapis.com/css2?family=Share+Tech+Mono&family=VT323&display=swap" rel="stylesheet">

</head>

<body>

<div class="crt-scanlines"></div>

<div class="crt-vignette"></div>

<div class="pip-noise"></div>

<div class="pip-frame game-container hidden-on-load">

<header class="pip-header">

<div class="pip-title-group">

<h1 class="game-title">Power Inc.</h1>

<div class="pip-title-divider"></div>

</div>

<div class="game-info">

<div class="date-display">

<span id="gameDate">Initializing...</span>

</div>

<div class="ip-display player-ip">

<span class="label">Your IP</span>

<span id="influence-points">50</span>

</div>

<div class="ip-display ai-ip">

<span class="label">AI IP</span>

<span id="ai-influence-points">50</span>

</div>

</div>

<div class="stats-bar">

<div class="stat player">

<span>Countries: <span id="player-countries-count">0/4</span></span>

</div>

<div class="stat ai">

<span>AI Countries: <span id="ai-countries-count">0/4</span></span>

</div>

</div>

</header>

<main class="pip-main">

<section class="game-controls pip-panel">

<button id="pauseBtn" class="control-btn pip-btn">Pause</button>

<button id="playBtn" class="control-btn pip-btn">Play</button>

<button id="ffwdBtn" class="control-btn pip-btn">Fast Forward</button>

<button id="saveGameBtn" class="control-btn pip-btn accent">Save</button>

<button id="upgradesBtn" class="control-btn pip-btn">Upgrades</button>

<button id="shopBtn" class="control-btn pip-btn">Shop</button>

</section>

<section class="autosave-toggle pip-panel">

<label for="autosave-checkbox">Autosave</label>

<input type="checkbox" id="autosave-checkbox">

</section>

<section class="country-list-container pip-panel">

<ul id="countriesList" class="country-list"></ul>

</section>

<section class="log-container pip-panel">

<div class="panel-heading">Message Log</div>

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

</section>

</main>

</div>

<div id="start-menu-modal" class="modal">

<div class="modal-content pip-panel">

<div class="panel-heading">Power Inc. — Start</div>

<p>Begin a new game or continue your last session.</p>

<div class="pip-btn-row">

<button id="newGameBtn" class="btn pip-btn accent">New Game</button>

<button id="continueGameBtn" class="btn pip-btn" style="display: none;">Continue</button>

</div>

</div>

</div>

<div id="country-selection-modal" class="modal hidden">

<div class="modal-content pip-panel">

<span class="close-button">&times;</span>

<div class="panel-heading">Select Your Country</div>

<p>Choose the faction you will lead:</p>

<div id="country-selection-list" class="country-selection-grid"></div>

</div>

</div>

<div id="country-action-modal" class="modal hidden">

<div class="modal-content pip-panel">

<span class="close-button">&times;</span>

<div class="panel-heading" id="action-country-title">Country</div>

<div class="influence-percent">

<p>Your Influence: <span id="player-influence-percent">0%</span></p>

<p>AI Influence: <span id="ai-influence-percent">0%</span></p>

</div>

<div class="influence-controls">

<label for="investAmount">Invest IP</label>

<input type="number" id="investAmount" value="50" min="10" step="10">

<button id="action-influence-btn" class="btn pip-btn accent">Influence</button>

</div>

</div>

</div>

<div id="upgrade-modal" class="modal hidden">

<div class="modal-content pip-panel">

<span class="close-button">&times;</span>

<div class="panel-heading">Upgrades</div>

<ul id="upgrade-list" class="upgrade-list"></ul>

</div>

</div>

<div id="shop-modal" class="modal hidden">

<div class="modal-content pip-panel">

<span class="close-button">&times;</span>

<div class="panel-heading">Shop</div>

<ul id="shop-list" class="shop-list">

<li>

<span>Unlock Religion Branch</span>

<span>Cost: 100 IP</span>

<button class="buy-btn pip-btn">Buy</button>

<span class="buy-meta" data-item="unlock-religion"></span>

</li>

<li>

<span>Unlock Pinguinia</span>

<span>Cost: 200 IP</span>

<button class="buy-btn pip-btn">Buy</button>

<span class="buy-meta" data-item="unlock-pinguinia"></span>

</li>

</ul>

</div>

</div>

<div id="ultimate-modal" class="modal hidden">

<div class="modal-content pip-panel">

<span class="close-button">&times;</span>

<div class="panel-heading" id="ultimate-modal-title"></div>

<div id="ultimate-modal-body"></div>

</div>

</div>

<script src="game.js"></script>

</body>

</html>

Css code starts here:

:root {

--pip-bg: #0f130f;

--pip-panel: #111813;

--pip-border: #3a4037;

--pip-glow: #45ff5a;

--pip-glow-soft: #2ad15a;

--pip-amber: #ffcc66;

--pip-red: #ff5e5e;

--pip-muted: #7f8f7f;

--pip-text: #c4f9c9;

--pip-text-dim: #9dd7a2;

--pip-btn: #1a241d;

--pip-btn-border: #4a664e;

--pip-btn-hover: #243227;

--pip-accent: #2ee66b;

--pip-warning: #f0c674;

--shadow-strong: 0 0 24px rgba(69,255,90,0.12), inset 0 0 12px rgba(69,255,90,0.08);

--shadow-soft: 0 0 8px rgba(69,255,90,0.1), inset 0 0 4px rgba(69,255,90,0.1);

}

\* { box-sizing: border-box; }

html, body { height: 100%; }

body {

margin: 0;

padding: 20px;

display: flex;

justify-content: center;

align-items: flex-start;

min-height: 100vh;

background: radial-gradient(1200px 800px at 50% 10%, #0c0f0c 0%, #090b09 60%, #070907 100%);

color: var(--pip-text);

font-family: 'Share Tech Mono', monospace;

letter-spacing: 0.02em;

}

.crt-scanlines {

pointer-events: none;

position: fixed;

inset: 0;

background: repeating-linear-gradient(0deg, rgba(255,255,255,0.02), rgba(255,255,255,0.02) 1px, rgba(0,0,0,0.02) 2px, rgba(0,0,0,0.02) 3px);

mix-blend-mode: soft-light;

z-index: 1000;

}

.crt-vignette {

pointer-events: none;

position: fixed;

inset: 0;

background: radial-gradient(circle at 50% 50%, transparent 55%, rgba(0,0,0,0.25) 90%);

z-index: 999;

}

.pip-noise {

pointer-events: none;

position: fixed;

inset: 0;

background-image: url('data:image/svg+xml;utf8,<svg xmlns="http://www.w3.org/2000/svg" opacity="0.045"><filter id="n"><feTurbulence type="fractalNoise" baseFrequency="0.65" numOctaves="2" stitchTiles="stitch"/></filter><rect width="100%" height="100%" filter="url(%23n)" /></svg>');

background-size: cover;

mix-blend-mode: overlay;

z-index: 998;

}

body.modal-open { overflow: hidden; }

.pip-frame {

width: 100%;

max-width: 1000px;

background: var(--pip-panel);

border: 3px solid var(--pip-border);

border-radius: 10px;

box-shadow: var(--shadow-strong);

position: relative;

overflow: hidden;

}

.pip-frame::before {

content: '';

position: absolute;

inset: 0;

background: linear-gradient(180deg, rgba(69,255,90,0.06), rgba(69,255,90,0.01));

pointer-events: none;

}

.pip-header { padding: 16px 20px 0 20px; }

.pip-title-group {

display: flex;

flex-direction: column;

align-items: center;

margin-bottom: 12px;

}

.game-title {

margin: 0;

font-size: 34px;

color: var(--pip-glow);

text-shadow: 0 0 12px rgba(69,255,90,0.6), 0 0 2px rgba(69,255,90,0.9);

font-family: 'VT323', monospace;

letter-spacing: 0.06em;

}

.pip-title-divider {

width: 100%;

height: 6px;

margin-top: 10px;

background:

radial-gradient(circle at 10px 3px, rgba(69,255,90,0.9) 2px, transparent 3px) left top / 40px 100% repeat-x,

linear-gradient(90deg, rgba(69,255,90,0.25) 0%, rgba(69,255,90,0.05) 100%);

opacity: 0.85;

}

.game-info {

display: flex;

justify-content: space-around;

margin: 8px 10px 12px;

padding: 8px;

border: 2px solid var(--pip-border);

border-radius: 6px;

background: linear-gradient(180deg, rgba(10,16,12,0.65), rgba(10,16,12,0.35));

}

.date-display,

.ip-display {

display: inline-flex;

flex-direction: column;

align-items: center;

padding: 6px 10px;

min-width: 120px;

border: 2px solid var(--pip-border);

background: var(--pip-btn);

border-radius: 6px;

box-shadow: var(--shadow-soft);

}

.date-display span,

.ip-display span { text-shadow: 0 0 6px rgba(69,255,90,0.5); }

.ip-display .label {

color: var(--pip-text-dim);

font-size: 12px;

text-transform: uppercase;

}

.stats-bar {

display: flex;

justify-content: space-around;

padding: 10px;

border-top: 2px solid var(--pip-border);

background: linear-gradient(180deg, rgba(12,18,14,0.85), rgba(12,18,14,0.4));

}

.stat span {

font-weight: 700;

color: var(--pip-text);

text-shadow: 0 0 6px rgba(69,255,90,0.3);

}

.stat.player span { color: var(--pip-glow); }

.stat.ai span { color: var(--pip-amber); }

.pip-main { padding: 16px; }

.pip-panel {

border: 2px solid var(--pip-border);

background: linear-gradient(180deg, rgba(9,14,11,0.85), rgba(9,14,11,0.5));

border-radius: 8px;

padding: 14px;

margin-bottom: 14px;

box-shadow: var(--shadow-soft);

position: relative;

}

.panel-heading {

font-family: 'VT323', monospace;

color: var(--pip-amber);

font-size: 22px;

letter-spacing: 0.04em;

margin-bottom: 10px;

text-shadow: 0 0 8px rgba(255,204,102,0.5);

display: inline-block;

border-bottom: 2px solid rgba(255,204,102,0.3);

padding-bottom: 2px;

}

.pip-btn, .control-btn, .btn {

background: var(--pip-btn);

color: var(--pip-text);

border: 2px solid var(--pip-btn-border);

border-radius: 6px;

padding: 10px 16px;

cursor: pointer;

font-family: 'Share Tech Mono', monospace;

text-transform: uppercase;

letter-spacing: 0.06em;

box-shadow: 0 2px 0 #0b0f0c, inset 0 0 10px rgba(69,255,90,0.05);

transition: background 0.12s ease, transform 0.05s ease, box-shadow 0.12s ease;

}

.pip-btn:hover, .control-btn:hover, .btn:hover {

background: var(--pip-btn-hover);

box-shadow: 0 2px 0 #0b0f0c, 0 0 12px rgba(69,255,90,0.2);

}

.pip-btn:active, .control-btn:active, .btn:active {

transform: translateY(1px);

box-shadow: inset 0 2px 8px rgba(0,0,0,0.5);

}

.pip-btn.accent {

border-color: var(--pip-accent);

color: var(--pip-accent);

text-shadow: 0 0 8px rgba(46,230,107,0.7);

}

.game-controls {

display: grid;

grid-template-columns: repeat(6, minmax(0, 1fr));

gap: 10px;

}

.autosave-toggle { display: flex; align-items: center; gap: 10px; }

.autosave-toggle input[type="checkbox"] { width: 18px; height: 18px; accent-color: var(--pip-accent); }

.country-list-container { padding: 0; }

.country-list {

list-style: none;

padding: 10px;

margin: 0;

display: grid;

gap: 10px;

}

.country-btn {

display: flex;

flex-direction: column;

width: 100%;

background: linear-gradient(180deg, rgba(20,28,22,0.85), rgba(20,28,22,0.55));

border: 2px solid var(--pip-border);

color: var(--pip-text);

padding: 12px;

border-radius: 6px;

cursor: pointer;

text-align: left;

transition: background 0.12s ease, border-color 0.12s ease, transform 0.05s ease;

}

.country-btn:hover {

background: linear-gradient(180deg, rgba(29,39,32,0.85), rgba(29,39,32,0.55));

border-color: var(--pip-glow-soft);

}

.country-btn.player-owned { border-color: var(--pip-glow); }

.country-btn.ai-owned { border-color: var(--pip-amber); }

.country-btn.inactive-country { opacity: 0.5; cursor: not-allowed; }

.country-btn.lockout { border-color: var(--pip-warning); opacity: 0.8; cursor: not-allowed; }

.country-header {

display: flex;

justify-content: space-between;

align-items: center;

font-weight: 700;

margin-bottom: 8px;

color: var(--pip-text);

}

.ownership {

font-size: 0.85em;

padding: 2px 8px;

border-radius: 12px;

border: 1px solid var(--pip-border);

color: var(--pip-text-dim);

}

.player-owned .ownership { color: var(--pip-glow); }

.ai-owned .ownership { color: var(--pip-amber); }

.inf-bars { width: 100%; display: grid; gap: 6px; }

.bar {

height: 10px;

background: #0e140f;

border: 1px solid var(--pip-border);

border-radius: 10px;

overflow: hidden;

position: relative;

box-shadow: inset 0 0 6px rgba(0,0,0,0.6);

}

.bar-fill { height: 100%; transition: width 0.5s ease-in-out; }

.bar-fill.player { background: linear-gradient(90deg, rgba(69,255,90,0.75), rgba(69,255,90,0.95)); box-shadow: 0 0 12px rgba(69,255,90,0.4); }

.bar-fill.ai { background: linear-gradient(90deg, rgba(255,204,102,0.75), rgba(255,204,102,0.95)); box-shadow: 0 0 12px rgba(255,204,102,0.35); }

.log-container .panel-heading { margin-bottom: 6px; }

.message-log {

height: 220px;

overflow-y: auto;

background: #0c120d;

border: 2px solid var(--pip-border);

border-radius: 6px;

padding: 10px;

box-shadow: inset 0 0 10px rgba(0,0,0,0.7);

overscroll-behavior: contain;

}

.message-log p {

margin: 5px 0;

font-size: 0.95rem;

line-height: 1.35;

color: var(--pip-text);

text-shadow: 0 0 6px rgba(69,255,90,0.2);

}

/\* Modals above overlays \*/

.modal {

position: fixed;

z-index: 1101;

inset: 0;

display: grid;

place-items: center;

background: rgba(7, 9, 7, 0.75);

transition: opacity 0.2s ease, visibility 0.2s ease;

}

.modal.hidden { opacity: 0; visibility: hidden; }

.modal .modal-content {

width: 92%;

max-width: 640px;

position: relative;

}

/\* Upgrades scroll containment \*/

#upgrade-modal .modal-content {

max-height: 85vh;

overflow-y: auto;

overscroll-behavior: contain;

}

#upgrade-list {

list-style: none;

padding: 0;

margin: 0;

max-height: 70vh;

overflow-y: auto;

overscroll-behavior: contain;

}

.upgrade-list li, .shop-list li {

background: linear-gradient(180deg, rgba(20,28,22,0.85), rgba(20,28,22,0.55));

padding: 12px;

margin-bottom: 10px;

border-radius: 6px;

border: 2px solid var(--pip-border);

display: grid;

grid-template-columns: 1fr auto auto;

align-items: center;

gap: 10px;

}

.upgrade-list h4 {

margin: 16px 0 6px;

color: var(--pip-glow);

text-shadow: 0 0 8px rgba(69,255,90,0.5);

font-weight: 700;

}

.btn-small { padding: 8px 12px; font-size: 0.9rem; }

.upgrade-list button:disabled { opacity: 0.5; cursor: not-allowed; }

.upgrade-list li.ultimate {

background: linear-gradient(180deg, rgba(32,24,40,0.8), rgba(32,24,40,0.5));

border-color: #6b4a7a;

box-shadow: inset 0 0 10px rgba(107,74,122,0.25);

}

.close-button {

position: absolute;

top: 8px;

right: 12px;

font-size: 26px;

font-weight: 700;

cursor: pointer;

color: var(--pip-text);

text-shadow: 0 0 10px rgba(69,255,90,0.4);

}

.country-selection-grid {

display: grid;

grid-template-columns: repeat(auto-fit, minmax(160px, 1fr));

gap: 12px;

margin-top: 12px;

}

.country-selection-btn {

padding: 16px;

border: 2px solid var(--pip-border);

background: linear-gradient(180deg, rgba(18,25,20,0.85), rgba(18,25,20,0.55));

color: var(--pip-text);

cursor: pointer;

border-radius: 6px;

font-size: 16px;

font-family: 'Share Tech Mono', monospace;

text-align: center;

box-shadow: var(--shadow-soft);

transition: transform 0.05s ease, background 0.12s ease, border-color 0.12s ease;

}

.country-selection-btn:hover {

transform: translateY(-1px);

background: linear-gradient(180deg, rgba(24,34,28,0.85), rgba(24,34,28,0.55));

border-color: var(--pip-glow-soft);

}

.country-selection-btn:active { transform: translateY(0); }

.country-selection-btn.disabled,

.country-selection-btn[disabled] {

opacity: 0.5;

cursor: not-allowed;

filter: grayscale(30%);

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

}

.influence-percent {

display: flex;

justify-content: space-around;

margin: 12px 0;

color: var(--pip-text);

}

.influence-controls {

display: grid;

grid-template-columns: 1fr auto;

gap: 10px;

align-items: center;

justify-content: center;

}

.influence-controls input[type="number"] {

width: 120px;

padding: 8px 10px;

background: #0d140f;

color: var(--pip-text);

border: 2px solid var(--pip-border);

border-radius: 6px;

font-family: 'Share Tech Mono', monospace;

outline: none;

}

.game-container.hidden-on-load { display: none; }

.shop-list .buy-meta { display: none; }

Js code starts here:

// Game data and state

const gameState = {

influencePoints: 50,

aiInfluencePoints: 50,

ai: { influencedCountries: [], unlockedTraits: [], personality: '' },

player: { influencedCountries: [], unlockedTraits: [] },

countries: [],

gameStarted: false,

ipGenerationBaseRate: 1,

eventOccurred: false,

shop: { unlockedPinguinia: false, unlockedReligion: false },

ipStoppageActive: false,

ipStoppageRemainingTicks: 0,

purchasedUltimateUpgrades: { 'IP Stoppage': false, 'Government Bribe': false, 'Aliens are here!': false, 'Believe!': false },

isAutosaveEnabled: false,

autosaveIntervalId: null,

currentDate: new Date('1984-10-11'),

gameSpeedMultiplier: 1,

firstAIControlledCountry: null,

playerCountry: null,

aiCountry: null,

countrySelectionPhase: false,

wasRunningBeforeModal: false

};

// Numerical resistance values

const resistanceValues = { 'Low': 100, 'Medium': 200, 'High': 300 };

// Upgrade costs

const upgradeCosts = { 'tier1': 50, 'tier2': 100, 'specialized': 250, 'ultimate': 500 };

const upgrades = {

propaganda: {

'Media Control': { cost: upgradeCosts.tier1, unlocked: false, requires: null, multiplier: 1.1, category: 'Propaganda', description: "Establishes full media control, providing a 10% boost to Propaganda influence gains." },

'Targeted Messaging': { cost: upgradeCosts.tier2, unlocked: false, requires: 'Media Control', multiplier: 1.2, category: 'Propaganda', description: "Allows for pinpoint messaging, providing a 20% boost to Propaganda influence gains." },

'Viral Disinformation': { cost: upgradeCosts.specialized, unlocked: false, requires: 'Targeted Messaging', multiplier: 1.3, category: 'Propaganda', description: "Unleashes a powerful disinformation campaign, providing a 30% boost to Propaganda influence gains." },

'Aliens are here!': { cost: 600, unlocked: false, requires: 'Viral Disinformation', specialAction: true, description: "Temporarily stops all influence attempts on an un-owned country for 3 in-game weeks." }

},

military: {

'Military Presence': { cost: upgradeCosts.tier1, unlocked: false, requires: null, multiplier: 1.2, category: 'Military', description: "Intimidates a country with a show of force, providing a 20% boost to Military influence gains." },

'Cyber Operations': { cost: upgradeCosts.tier2, unlocked: false, requires: 'Military Presence', multiplier: 1.3, category: 'Military', description: "Launches cyber-attacks to disrupt a country's defenses, providing a 30% boost to Military influence gains." },

'Sovereign Intervention': { cost: upgradeCosts.specialized, unlocked: false, requires: 'Cyber Operations', multiplier: 1.4, category: 'Military', description: "Performs a covert military action to rapidly gain Military influence, providing a 40% boost." },

'IP Stoppage': { cost: 700, unlocked: false, requires: 'Sovereign Intervention', specialAction: true, description: "Stops AI IP generation for 3 in-game weeks." }

},

diplomatic: {

'Alliance Building': { cost: upgradeCosts.tier1, unlocked: false, requires: null, multiplier: 1.1, category: 'Diplomatic', description: "Establishes diplomatic ties, providing a 10% boost to Diplomatic influence gains." },

'Regional Summit': { cost: upgradeCosts.tier2, unlocked: false, requires: 'Alliance Building', multiplier: 1.2, category: 'Diplomatic', description: "Hosts a major summit to further diplomatic influence, providing a 20% boost." },

'Sanctions Hammer': { cost: upgradeCosts.specialized, unlocked: false, requires: 'Regional Summit', multiplier: 1.3, category: 'Diplomatic', description: "Imposes powerful economic sanctions to cripple a country's resistance, providing a 30% boost to Diplomatic influence." },

'Alliance Breaker': { cost: upgradeCosts.ultimate, unlocked: false, requires: 'Sanctions Hammer', multiplier: 1.5, category: 'Diplomatic', description: "A powerful diplomatic tool that allows you to influence countries fully controlled by the AI." }

},

economic: {

'Trade Deals': { cost: upgradeCosts.tier1, unlocked: false, requires: null, multiplier: 1.3, category: 'Economic', description: "Secures new trade deals, providing a 30% boost to Economic influence gains." },

'Resource Leverage': { cost: upgradeCosts.tier2, unlocked: false, requires: 'Trade Deals', multiplier: 1.4, category: 'Economic', description: "Gains control over a country's key resources, providing a 40% boost to Economic influence gains." },

'Monopolization': { cost: upgradeCosts.specialized, unlocked: false, requires: 'Resource Leverage', multiplier: 1.5, category: 'Economic', description: "Takes complete control of a country's economy, granting a 50% boost to Economic influence gains." },

'Government Bribe': { cost: 800, unlocked: false, requires: 'Monopolization', specialAction: true, description: "Removes 50% of opponent's influence in a country and adds it to yours." }

},

religion: {

'Word of God': { cost: 100, unlocked: false, requires: null, multiplier: 1.0, description: "Reduces all resistance values of a targeted country by 10% on first influence attempt.", category: 'Religion' },

'Church of Holy Molly': { cost: 200, unlocked: false, requires: 'Word of God', multiplier: 1.15, description: "Increases influence gains by 15% across all actions.", category: 'Religion' },

'Messiah': { cost: 300, unlocked: false, requires: 'Church of Holy Molly', multiplier: 1.05, description: "Passively increases your base IP generation speed by 5%.", category: 'Religion' },

'Believe!': { cost: 1000, unlocked: false, requires: 'Messiah', specialAction: true, description: "A one-off action that resets the AI's first 100% influenced country." }

}

};

// UI refs

const influencePointsEl = document.getElementById('influence-points');

const aiInfluencePointsEl = document.getElementById('ai-influence-points');

const playerCountriesCountEl = document.getElementById('player-countries-count');

const aiCountriesCountEl = document.getElementById('ai-countries-count');

const messageLogEl = document.getElementById('message-log');

const upgradeModal = document.getElementById('upgrade-modal');

const shopModal = document.getElementById('shop-modal');

const closeButtons = document.querySelectorAll('.close-button');

const countryActionModal = document.getElementById('country-action-modal');

const actionCountryTitle = document.getElementById('action-country-title');

const playerInfluencePercentEl = document.getElementById('player-influence-percent');

const aiInfluencePercentEl = document.getElementById('ai-influence-percent');

const actionInfluenceBtn = document.getElementById('action-influence-btn');

const countryListEl = document.getElementById('countriesList');

const gameDateEl = document.getElementById('gameDate');

const pauseBtn = document.getElementById('pauseBtn');

const playBtn = document.getElementById('playBtn');

const ffwdBtn = document.getElementById('ffwdBtn');

const upgradesBtn = document.getElementById('upgradesBtn');

const shopBtn = document.getElementById('shopBtn');

const investAmountInput = document.getElementById('investAmount');

const ultimateModal = document.getElementById('ultimate-modal');

const ultimateModalTitle = document.getElementById('ultimate-modal-title');

const ultimateModalBody = document.getElementById('ultimate-modal-body');

const gameContainer = document.querySelector('.game-container');

const startMenuModal = document.getElementById('start-menu-modal');

const newGameBtn = document.getElementById('newGameBtn');

const continueGameBtn = document.getElementById('continueGameBtn');

const saveGameBtn = document.getElementById('saveGameBtn');

const autosaveCheckbox = document.getElementById('autosave-checkbox');

const countrySelectionModal = document.getElementById('country-selection-modal');

const countrySelectionList = document.getElementById('country-selection-list');

let selectedCountryName = null;

let gameIntervalId = null;

// Pause-on-modal helpers

function pauseGame() {

clearInterval(gameIntervalId);

gameIntervalId = null;

gameState.gameStarted = false;

logMessage("Game paused.");

}

function pauseForModalIfRunning() {

gameState.wasRunningBeforeModal = gameIntervalId !== null;

if (gameState.wasRunningBeforeModal) pauseGame();

}

function resumeAfterAllModalsClosed() {

const anyOpen = document.querySelectorAll('.modal:not(.hidden)').length > 0;

if (!anyOpen && gameState.wasRunningBeforeModal && !gameState.countrySelectionPhase) {

gameState.wasRunningBeforeModal = false;

startGameLoop();

}

}

function openModal(modalEl) {

pauseForModalIfRunning();

modalEl.classList.remove('hidden');

document.body.classList.add('modal-open');

}

function closeModal(modalEl) {

modalEl.classList.add('hidden');

const anyOpen = document.querySelectorAll('.modal:not(.hidden)').length > 0;

if (!anyOpen) {

document.body.classList.remove('modal-open');

resumeAfterAllModalsClosed();

}

}

// Backdrop close only; stopPropagation inside content

[countryActionModal, upgradeModal, shopModal, ultimateModal, countrySelectionModal, startMenuModal].forEach((modal) => {

if (!modal) return;

modal.addEventListener('click', (e) => { if (e.target === modal) closeModal(modal); });

const content = modal.querySelector('.modal-content');

if (content) content.addEventListener('click', (e) => e.stopPropagation());

});

// Optional ESC to close topmost modal

document.addEventListener('keydown', (e) => {

if (e.key !== 'Escape') return;

const openModals = Array.from(document.querySelectorAll('.modal:not(.hidden)'));

const top = openModals[openModals.length - 1];

if (top) closeModal(top);

});

// UI

function updateUI() {

influencePointsEl.textContent = Math.floor(gameState.influencePoints);

aiInfluencePointsEl.textContent = Math.floor(gameState.aiInfluencePoints);

playerCountriesCountEl.textContent = gameState.player.influencedCountries.length + '/4';

aiCountriesCountEl.textContent = gameState.ai.influencedCountries.length + '/4';

renderCountryList();

}

function updateDateDisplay() {

const year = gameState.currentDate.getFullYear();

const month = gameState.currentDate.toLocaleDateString('en-US', { month: 'long' });

const day = gameState.currentDate.getDate();

gameDateEl.textContent = `${year}, ${month} ${day}`;

}

function logMessage(message, isStrong = false) {

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

p.innerHTML = isStrong ? `<strong>${message}</strong>` : message;

messageLogEl.prepend(p);

}

function getOwnership(country) {

if (country.playerInfluence >= 100) return "Player";

if (country.aiInfluence >= 100) return "AI";

return "Neutral";

}

function renderCountryList() {

countryListEl.innerHTML = '';

gameState.countries.forEach(country => {

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

wrapper.className = 'country-btn';

wrapper.dataset.countryId = country.name.toLowerCase();

if (country.lockoutRemainingTicks > 0) wrapper.classList.add('lockout');

const ownership = getOwnership(country);

wrapper.classList.add(ownership.toLowerCase() + '-owned');

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

header.className = 'country-header';

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

nameEl.textContent = country.name;

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

ownEl.className = 'ownership';

ownEl.textContent = `Owner: ${ownership}`;

header.appendChild(nameEl);

header.appendChild(ownEl);

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

bars.className = 'inf-bars';

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

barPlayer.className = 'bar';

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

barPlayerFill.className = 'bar-fill player';

barPlayerFill.style.width = `${country.playerInfluence}%`;

barPlayer.appendChild(barPlayerFill);

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

barAI.className = 'bar';

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

barAIFill.className = 'bar-fill ai';

barAIFill.style.width = `${country.aiInfluence}%`;

barAI.appendChild(barAIFill);

bars.appendChild(barPlayer);

bars.appendChild(barAI);

wrapper.appendChild(header);

wrapper.appendChild(bars);

if (country.name === 'Pinguinia' && !gameState.shop.unlockedPinguinia) {

wrapper.classList.add('inactive-country');

wrapper.addEventListener('click', () => logMessage("You must unlock Pinguinia from the shop before influencing it."));

} else if (country.lockoutRemainingTicks > 0) {

wrapper.addEventListener('click', () => logMessage(`This country is under lockdown. ${country.lockoutRemainingTicks} days remaining.`));

} else {

wrapper.addEventListener('click', () => handleCountryClick(country.name));

}

countryListEl.appendChild(wrapper);

});

}

// Save/Load

function saveGame() {

try {

const saveState = {

gameState: {

influencePoints: gameState.influencePoints,

aiInfluencePoints: gameState.aiInfluencePoints,

ipStoppageRemainingTicks: gameState.ipStoppageRemainingTicks,

purchasedUltimateUpgrades: gameState.purchasedUltimateUpgrades,

gameStarted: gameState.gameStarted,

ipGenerationBaseRate: gameState.ipGenerationBaseRate,

eventOccurred: gameState.eventOccurred,

shop: gameState.shop,

isAutosaveEnabled: gameState.isAutosaveEnabled,

currentDate: gameState.currentDate,

gameSpeedMultiplier: gameState.gameSpeedMultiplier,

firstAIControlledCountry: gameState.firstAIControlledCountry,

playerCountry: gameState.playerCountry,

aiCountry: gameState.aiCountry

},

upgrades: JSON.parse(JSON.stringify(upgrades)),

countries: gameState.countries

};

localStorage.setItem('powerIncSave', JSON.stringify(saveState));

logMessage("Game saved!", true);

} catch (e) {

logMessage("Error saving game. Your browser might not support localStorage.", true);

}

}

function loadGame() {

try {

const savedData = localStorage.getItem('powerIncSave');

if (savedData) {

const saveState = JSON.parse(savedData);

Object.assign(gameState, saveState.gameState);

gameState.currentDate = new Date(saveState.gameState.currentDate);

for (const category in upgrades) {

if (saveState.upgrades[category]) Object.assign(upgrades[category], saveState.upgrades[category]);

}

gameState.countries = saveState.countries;

logMessage("Game loaded successfully!", true);

gameContainer.classList.remove('hidden-on-load');

openModal(startMenuModal); // ensure body modal state is consistent

closeModal(startMenuModal);

startGameLoop();

autosaveCheckbox.checked = gameState.isAutosaveEnabled;

if (gameState.isAutosaveEnabled) startAutosaveTimer();

updateUI();

return true;

}

} catch (e) {

logMessage("Error loading saved game. Starting a new one.", true);

localStorage.removeItem('powerIncSave');

}

return false;

}

// New game

function startNewGame() {

gameState.influencePoints = 50;

gameState.aiInfluencePoints = 50;

gameState.player.influencedCountries = [];

gameState.ai.influencedCountries = [];

gameState.player.unlockedTraits = [];

gameState.ai.unlockedTraits = [];

gameState.ipStoppageRemainingTicks = 0;

gameState.purchasedUltimateUpgrades = { 'IP Stoppage': false, 'Government Bribe': false, 'Aliens are here!': false, 'Believe!': false };

gameState.shop.unlockedPinguinia = false;

gameState.shop.unlockedReligion = false;

gameState.isAutosaveEnabled = autosaveCheckbox.checked;

gameState.currentDate = new Date('1984-10-11');

gameState.gameSpeedMultiplier = 1;

gameState.firstAIControlledCountry = null;

gameState.playerCountry = null;

gameState.aiCountry = null;

gameState.countrySelectionPhase = true;

for (const category in upgrades) for (const traitName in upgrades[category]) upgrades[category][traitName].unlocked = false;

const countryNames = ["United States", "Canada", "Mexico", "Brazil"];

const resistances = ["Military", "Economic", "Propaganda"];

const countryData = countryNames.map(name => {

const stats = {};

const assignedResistanceKeys = ["Low", "Medium", "High"].sort(() => 0.5 - Math.random());

resistances.forEach((res, i) => { stats[res] = resistanceValues[assignedResistanceKeys[i]]; });

return { name, resistance: stats, owner: 'Neutral', playerInfluence: 0, aiInfluence: 0, lockoutRemainingTicks: 0, historicalAIP: 0 };

});

const pinguinia = { name: "Pinguinia", resistance: { Military: 300, Economic: 300, Propaganda: 300 }, owner: 'Neutral', playerInfluence: 0, aiInfluence: 0, lockoutRemainingTicks: 0, historicalAIP: 0 };

gameState.countries = [...countryData, pinguinia];

gameContainer.classList.remove('hidden-on-load');

closeModal(startMenuModal);

openModal(countrySelectionModal);

showCountrySelection();

logMessage("Starting a new game.");

}

// Country selection

function showCountrySelection() {

countrySelectionList.innerHTML = '';

gameState.countries.forEach(country => {

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

button.className = 'country-selection-btn';

button.textContent = country.name;

button.dataset.countryName = country.name;

const lockedPinguinia = (country.name === 'Pinguinia' && !gameState.shop.unlockedPinguinia);

if (lockedPinguinia) {

button.classList.add('disabled');

button.setAttribute('disabled', 'disabled');

} else {

button.addEventListener('click', () => selectPlayerCountry(country.name));

}

countrySelectionList.appendChild(button);

});

}

function selectPlayerCountry(countryName) {

if (countryName === 'Pinguinia' && !gameState.shop.unlockedPinguinia) {

logMessage("You must unlock Pinguinia from the shop before selecting it.");

return;

}

gameState.playerCountry = countryName;

const playerCountry = gameState.countries.find(c => c.name === countryName);

playerCountry.playerInfluence = 100;

playerCountry.owner = 'Player';

gameState.player.influencedCountries.push(countryName);

logMessage(`You have selected ${countryName} as your country!`, true);

selectAICountry();

closeModal(countrySelectionModal);

completeCountrySelection();

}

function selectAICountry() {

const available = gameState.countries.filter(c => {

if (c.name === gameState.playerCountry) return false;

if (c.name === 'Pinguinia' && !gameState.shop.unlockedPinguinia) return false;

return true;

});

const selected = available[Math.floor(Math.random() \* available.length)];

gameState.aiCountry = selected.name;

const aiCountry = gameState.countries.find(c => c.name === selected.name);

aiCountry.aiInfluence = 100;

aiCountry.owner = 'AI';

gameState.ai.influencedCountries.push(selected.name);

const vals = Object.values(aiCountry.resistance);

const max = Math.max(...vals);

const type = Object.keys(aiCountry.resistance).find(k => aiCountry.resistance[k] === max);

gameState.ai.personality = (type === 'Military') ? 'Aggressive' : (type === 'Economic') ? 'Economist' : 'Defensive';

logMessage(`AI has selected ${selected.name} as their country!`, true);

logMessage(`AI personality: ${gameState.ai.personality}`, true);

}

function completeCountrySelection() {

gameState.countrySelectionPhase = false;

updateUI();

startGameLoop();

if (gameState.isAutosaveEnabled) startAutosaveTimer();

logMessage("Both countries have been selected. The game begins now!", true);

}

// Controls

function startGameLoop() {

if (gameState.countrySelectionPhase) { logMessage("Select countries to begin."); return; }

if (gameIntervalId) clearInterval(gameIntervalId);

const intervalDuration = 1000 / gameState.gameSpeedMultiplier;

gameIntervalId = setInterval(gameLoop, intervalDuration);

gameState.gameStarted = true;

logMessage("Game started.");

}

function ffwdToggle() {

gameState.gameSpeedMultiplier = gameState.gameSpeedMultiplier === 1 ? 2 : 1;

logMessage(`Game speed x${gameState.gameSpeedMultiplier}.`);

startGameLoop();

}

// Influence and AI

const BASE\_INFLUENCE\_GAIN = 50;

const INFLUENCE\_DECAY\_RATE = 1;

function getPlayerInfluenceMultiplier(influenceType) {

let multiplier = 1;

for (const category in upgrades) for (const traitName in upgrades[category]) {

const trait = upgrades[category][traitName];

if (trait.unlocked) {

if (trait.category !== 'Religion' && traitName !== 'Messiah') multiplier \*= trait.multiplier;

else if (traitName === 'Church of Holy Molly') multiplier \*= trait.multiplier;

}

}

return multiplier;

}

function influenceCountry(countryName, influenceType, playerType, amount) {

const country = gameState.countries.find(c => c.name === countryName);

if (playerType === 'Player') {

if (gameState.influencePoints < amount) { logMessage(`Not enough IP to influence ${countryName}. Need ${amount} IP.`); closeModal(countryActionModal); return; }

gameState.influencePoints -= amount;

} else {

if (gameState.aiInfluencePoints < amount) { logMessage(`AI is unable to influence ${countryName}. Not enough IP.`); return; }

gameState.aiInfluencePoints -= amount;

}

let resistanceValue = country.resistance[influenceType];

if (upgrades.religion['Word of God'].unlocked) resistanceValue = Math.max(0, resistanceValue \* 0.9);

let influenceGain = (BASE\_INFLUENCE\_GAIN / 50) \* amount;

if (playerType === 'Player') influenceGain \*= getPlayerInfluenceMultiplier(influenceType);

if (playerType === 'Player') {

country.playerInfluence = Math.min(100, country.playerInfluence + influenceGain);

country.aiInfluence = Math.max(0, country.aiInfluence - (influenceGain / 2));

logMessage(`You gained ${influenceGain.toFixed(0)}% influence in ${country.name}.`);

} else {

country.aiInfluence = Math.min(100, country.aiInfluence + influenceGain);

country.playerInfluence = Math.max(0, country.playerInfluence - (influenceGain / 2));

logMessage(`AI gained ${influenceGain.toFixed(0)}% influence in ${country.name}.`);

}

closeModal(countryActionModal);

updateUI();

checkWin();

}

function handleCountryClick(countryName) {

selectedCountryName = countryName;

const country = gameState.countries.find(c => c.name === countryName);

if (gameState.purchasedUltimateUpgrades['Government Bribe']) { openUltimateModal('Government Bribe', country); return; }

if (gameState.purchasedUltimateUpgrades['Aliens are here!']) { openUltimateModal('Aliens are here!', country); return; }

if (country.owner === 'Player') { logMessage(`${country.name} is already under your full influence.`); return; }

if (country.owner === 'AI' && !upgrades.diplomatic['Alliance Breaker'].unlocked) { logMessage(`You cannot influence ${country.name}. The AI has full control. You need 'Alliance Breaker'.`); return; }

actionCountryTitle.textContent = `Actions for ${countryName}`;

playerInfluencePercentEl.textContent = `${country.playerInfluence.toFixed(0)}%`;

aiInfluencePercentEl.textContent = `${country.aiInfluence.toFixed(0)}%`;

openModal(countryActionModal);

}

function decayForeignInfluence() {

gameState.countries.forEach(country => {

if (country.owner === 'Neutral' && country.playerInfluence > 0 && country.aiInfluence > 0) {

if (country.playerInfluence > country.aiInfluence) country.aiInfluence = Math.max(0, country.aiInfluence - INFLUENCE\_DECAY\_RATE);

else if (country.aiInfluence > country.playerInfluence) country.playerInfluence = Math.max(0, country.playerInfluence - INFLUENCE\_DECAY\_RATE);

}

});

}

function calculateTotalMultiplier() {

let total = 1;

for (const category in upgrades) for (const traitName in upgrades[category]) {

const trait = upgrades[category][traitName];

if (trait.unlocked) {

if (trait.category !== 'Religion' && traitName !== 'Messiah') total \*= trait.multiplier;

else if (traitName === 'Church of Holy Molly') total \*= trait.multiplier;

}

}

return total;

}

function gameEvents() {

const eventProbability = gameState.eventOccurred ? 0.01 : 0.05;

if (Math.random() < eventProbability) {

const neutral = gameState.countries.filter(c => c.owner === 'Neutral');

const owned = gameState.countries.filter(c => c.owner !== 'Neutral');

const ev = Math.floor(Math.random() \* 3);

switch (ev) {

case 0:

if (neutral.length > 0) {

const c = neutral[Math.floor(Math.random() \* neutral.length)];

for (const t in c.resistance) c.resistance[t] = Math.max(50, c.resistance[t] \* 0.8);

logMessage(`[Event] Internal strife weakens ${c.name}.`);

}

break;

case 1:

if (owned.length > 0) {

const c2 = owned[Math.floor(Math.random() \* owned.length)];

for (const t in c2.resistance) c2.resistance[t] = Math.min(500, c2.resistance[t] \* 1.2);

logMessage(`[Event] Crisis strengthens ${c2.name}.`);

}

break;

case 2:

const who = Math.random() < 0.5 ? 'Player' : 'AI';

if (who === 'Player') { gameState.influencePoints += 100; logMessage(`[Event] Foreign aid: +100 IP.`); }

else { gameState.aiInfluencePoints += 100; logMessage(`[Event] AI foreign aid: +100 IP.`); }

break;

}

gameState.eventOccurred = true;

}

}

function gameLoop() {

const playerBonus = gameState.player.influencedCountries.length;

const aiBonus = gameState.ai.influencedCountries.length;

let baseRate = gameState.ipGenerationBaseRate;

if (upgrades.religion['Messiah'].unlocked) baseRate \*= upgrades.religion['Messiah'].multiplier;

const playerIPGain = (baseRate + playerBonus) \* calculateTotalMultiplier() \* gameState.gameSpeedMultiplier;

if (gameState.ipStoppageRemainingTicks > 0) {

gameState.ipStoppageRemainingTicks--;

logMessage(`AI IP generation stopped. ${gameState.ipStoppageRemainingTicks} days left.`);

} else {

const aiIPGain = (gameState.ipGenerationBaseRate + aiBonus) \* gameState.gameSpeedMultiplier;

gameState.aiInfluencePoints += aiIPGain;

}

gameState.influencePoints += playerIPGain;

gameState.countries.forEach(c => { if (c.lockoutRemainingTicks > 0) c.lockoutRemainingTicks--; });

decayForeignInfluence();

aiTakeTurn();

gameEvents();

gameState.currentDate.setDate(gameState.currentDate.getDate() + 1);

updateUI();

updateDateDisplay();

}

function aiChooseTarget() {

const neutral = gameState.countries.filter(c => c.owner === 'Neutral' && c.name !== 'Pinguinia' && c.lockoutRemainingTicks === 0);

const contested = gameState.countries.filter(c => c.playerInfluence > 0 && c.owner !== 'Player' && c.lockoutRemainingTicks === 0);

if (contested.length > 0) return contested[Math.floor(Math.random() \* contested.length)];

if (neutral.length > 0) return neutral[Math.floor(Math.random() \* neutral.length)];

return null;

}

function aiTakeTurn() {

const owned = gameState.countries.find(c => c.owner === 'AI');

if (owned) {

for (const branch in upgrades) {

const ultimate = Object.values(upgrades[branch]).find(u => u.specialAction);

if (ultimate && !ultimate.unlocked && upgrades[branch][ultimate.requires]?.unlocked && gameState.aiInfluencePoints >= ultimate.cost) {

const stat = owned.resistance[branch] || 100;

const r = Math.random();

let prob = 0;

if (stat === 300) prob = 0.6; else if (stat === 200) prob = 0.3; else if (stat === 100) prob = 0.1;

if (r < prob) { ultimate.unlocked = true; gameState.aiInfluencePoints -= ultimate.cost; logMessage(`AI unlocked '${ultimate.description}'`); return; }

}

}

}

const target = aiChooseTarget();

if (!target) { checkWin(); return; }

const budget = Math.floor(Math.min(gameState.aiInfluencePoints, 50));

if (budget <= 0) return;

let type = 'Propaganda';

if (gameState.ai.personality === 'Aggressive') type = 'Military';

else if (gameState.ai.personality === 'Economist') type = 'Economic';

else if (gameState.ai.personality === 'Defensive') type = 'Propaganda';

influenceCountry(target.name, type, 'AI', budget);

}

function checkWin() {

const playerWin = gameState.countries.filter(c => c.name !== 'Pinguinia').every(c => c.playerInfluence >= 100);

const aiWin = gameState.countries.filter(c => c.name !== 'Pinguinia').every(c => c.aiInfluence >= 100);

if (!gameState.firstAIControlledCountry) {

const aiControlled = gameState.countries.find(c => c.aiInfluence >= 100);

if (aiControlled) {

gameState.firstAIControlledCountry = aiControlled;

if (upgrades.religion['Believe!'].unlocked) {

logMessage("The AI has taken a country! 'Believe!' is activated!", true);

setTimeout(() => {

gameState.firstAIControlledCountry.playerInfluence = 0;

gameState.firstAIControlledCountry.aiInfluence = 0;

logMessage(`"Have I not commanded you? Be strong and courageous..." — Joshua 1:9`, true);

logMessage(`'${gameState.firstAIControlledCountry.name}' has been reset!`, true);

gameState.firstAIControlledCountry = null;

upgrades.religion['Believe!'].unlocked = false;

updateUI();

}, 2000);

}

}

}

if (playerWin) { pauseGame(); logMessage("You influenced all countries. You win!", true); return true; }

if (aiWin) { pauseGame(); logMessage("The AI influenced all countries. You lose.", true); return true; }

return false;

}

function openUltimateModal(upgradeName, country) {

ultimateModalTitle.textContent = upgradeName;

ultimateModalBody.innerHTML = '';

if (upgradeName === 'Government Bribe') {

ultimateModalBody.innerHTML = `

<p>Use 'Government Bribe' on ${country.name}?</p>

<p>Reduce AI influence by ${(country.aiInfluence / 2).toFixed(0)}% and gain the same.</p>

<button id="useBribeBtn" class="btn pip-btn accent">Use Bribe</button>

`;

openModal(ultimateModal);

document.getElementById('useBribeBtn').addEventListener('click', () => { useGovernmentBribe(country); closeModal(ultimateModal); });

} else if (upgradeName === 'Aliens are here!') {

ultimateModalBody.innerHTML = `

<p>Deploy 'Aliens are here!' on ${country.name}?</p>

<p>Stops all influence attempts here for 3 weeks.</p>

<button id="useAliensBtn" class="btn pip-btn accent">Confirm</button>

`;

openModal(ultimateModal);

document.getElementById('useAliensBtn').addEventListener('click', () => { useAliensAreHere(country); closeModal(ultimateModal); });

}

}

function useIPStoppage() {

if (gameState.purchasedUltimateUpgrades['IP Stoppage'] === 'used') { logMessage('IP Stoppage already used.', false); return; }

gameState.ipStoppageRemainingTicks = 21;

gameState.purchasedUltimateUpgrades['IP Stoppage'] = 'used';

logMessage("AI IP generation halted for 3 weeks!", true);

updateUI();

closeModal(upgradeModal);

}

function useGovernmentBribe(country) {

if (gameState.purchasedUltimateUpgrades['Government Bribe'] === 'used') { logMessage('Government Bribe already used.', false); return; }

const reductionAmount = country.aiInfluence / 2;

country.aiInfluence -= reductionAmount;

country.playerInfluence += reductionAmount;

logMessage(`Bribed ${country.name}. AI -${reductionAmount.toFixed(0)}%.`, true);

gameState.purchasedUltimateUpgrades['Government Bribe'] = 'used';

updateUI();

closeModal(upgradeModal);

}

function useAliensAreHere(country) {

if (gameState.purchasedUltimateUpgrades['Aliens are here!'] === 'used') { logMessage('Aliens are here! already used.', false); return; }

country.lockoutRemainingTicks = 21;

logMessage(`Aliens deployed on ${country.name}. 3 weeks lockout.`, true);

gameState.purchasedUltimateUpgrades['Aliens are here!'] = 'used';

updateUI();

closeModal(upgradeModal);

}

function renderUpgrades() {

const list = document.getElementById('upgrade-list');

list.innerHTML = '';

for (const category in upgrades) {

if (category === 'religion' && !gameState.shop.unlockedReligion) continue;

const categoryHeader = document.createElement('h4');

categoryHeader.textContent = category.charAt(0).toUpperCase() + category.slice(1);

list.appendChild(categoryHeader);

for (const traitName in upgrades[category]) {

const trait = upgrades[category][traitName];

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

if (trait.specialAction) li.classList.add('ultimate');

const canPurchase = !trait.unlocked && (!trait.requires || upgrades[category][trait.requires].unlocked) && gameState.influencePoints >= trait.cost;

const buttonText = trait.unlocked ? 'Unlocked' : (canPurchase ? 'Buy' : 'Unavailable');

li.innerHTML = `

<span class="upgrade-name">${traitName}</span>

<span>Cost: ${trait.cost} IP</span>

<button class="btn btn-small pip-btn" ${canPurchase ? '' : 'disabled'}>${buttonText}</button>

`;

if (canPurchase) {

li.querySelector('button').addEventListener('click', () => {

gameState.influencePoints -= trait.cost;

trait.unlocked = true;

gameState.player.unlockedTraits.push(traitName);

logMessage(`Unlocked "${traitName}".`);

if (trait.specialAction) {

gameState.purchasedUltimateUpgrades[traitName] = true;

if (traitName === 'Believe!') logMessage(`"The Lord is my strength and my shield..." — Joshua 1:9`, true);

if (traitName === 'IP Stoppage') useIPStoppage();

}

closeModal(upgradeModal);

updateUI();

});

}

list.appendChild(li);

}

}

}

// Init

function init() {

const savedData = localStorage.getItem('powerIncSave');

continueGameBtn.style.display = savedData ? 'block' : 'none';

startMenuModal.classList.remove('hidden');

document.body.classList.add('modal-open');

gameContainer.classList.add('hidden-on-load');

newGameBtn.addEventListener('click', () => { localStorage.removeItem('powerIncSave'); startNewGame(); });

continueGameBtn.addEventListener('click', () => loadGame());

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

const influenceType = 'Propaganda';

const amount = Math.max(1, Math.floor(Number(investAmountInput.value || 0)));

influenceCountry(selectedCountryName, influenceType, 'Player', amount);

});

closeButtons.forEach(button => {

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

[countryActionModal, upgradeModal, shopModal, ultimateModal, countrySelectionModal, startMenuModal].forEach(m => closeModal(m));

});

});

pauseBtn.addEventListener('click', pauseGame);

playBtn.addEventListener('click', startGameLoop);

ffwdBtn.addEventListener('click', ffwdToggle);

upgradesBtn.addEventListener('click', () => { openModal(upgradeModal); renderUpgrades(); });

shopBtn.addEventListener('click', () => { openModal(shopModal); });

saveGameBtn.addEventListener('click', saveGame);

autosaveCheckbox.addEventListener('change', (e) => { gameState.isAutosaveEnabled = e.target.checked; if (gameState.isAutosaveEnabled) startAutosaveTimer(); else stopAutosaveTimer(); });

window.addEventListener('beforeunload', () => { if (gameState.isAutosaveEnabled) saveGame(); });

document.querySelectorAll('#shop-list li').forEach(li => {

const btn = li.querySelector('.buy-btn');

const meta = li.querySelector('.buy-meta');

if (!btn || !meta) return;

const item = meta.dataset.item;

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

switch(item) {

case 'unlock-religion':

if (gameState.influencePoints >= 100) { gameState.influencePoints -= 100; gameState.shop.unlockedReligion = true; logMessage("Religion branch unlocked!"); }

else logMessage("Not enough IP to unlock Religion.");

break;

case 'unlock-pinguinia':

if (gameState.influencePoints >= 200) { gameState.influencePoints -= 200; gameState.shop.unlockedPinguinia = true; logMessage("Pinguinia unlocked!"); }

else logMessage("Not enough IP to unlock Pinguinia.");

break;

}

closeModal(shopModal);

updateUI();

});

});

updateUI();

updateDateDisplay();

}

init();