// Global Ideology Game - Enhanced Version

// Improved with error handling, mobile support, and better performance

// Game Module

const GlobalIdeology = (function() {

// Element Selection with error handling

const elements = {};

function getElement(id) {

const element = document.getElementById(id);

if (!element) {

console.error(`Element with id '${id}' not found`);

return null;

}

return element;

}

function initializeElements() {

const elementIds = [

'player-ip-count', 'player-munee-count', 'ai-ip-count',

'country-grid', 'game-log-list', 'selection-grid',

'modal-player-ip', 'loading-screen', 'country-selection-screen',

'main-game-screen', 'main-menu-modal', 'daily-bonus-modal',

'bonus-title', 'bonus-message', 'bonus-description',

'bonus-modal-close-btn', 'tutorial-modal', 'tutorial-title',

'tutorial-description', 'tutorial-next-btn', 'tutorial-back-btn',

'tutorial-close-btn', 'tutorial-options', 'dont-show-again-checkbox',

'pause-btn', 'play-btn', 'ffwd-btn', 'upgrade-btn', 'shop-btn',

'save-game-btn', 'autosave-checkbox', 'modal-start-btn',

'modal-continue-btn', 'add-munee-btn', 'country-modal',

'upgrade-modal', 'shop-modal', 'country-name', 'country-description',

'stat-military', 'stat-economic', 'stat-diplomatic', 'stat-ideological',

'ip-amount', 'upgrade-country-btn', 'upgrade-tree-container',

'shop-container', 'shopTabs'

];

elementIds.forEach(id => {

elements[id] = getElement(id);

});

// Initialize modals array

elements.modals = [

elements['main-menu-modal'],

elements['country-modal'],

elements['upgrade-modal'],

elements['shop-modal'],

elements['daily-bonus-modal'],

elements['tutorial-modal']

].filter(modal => modal !== null);

}

// Game State & Variables

let gameState = {

playerIP: 100,

playerMunee: 0,

aiIP: 100,

playerCountry: '',

aiCountry: '',

aiPersonality: '',

countries: [],

upgrades: {},

gameSpeed: 'pause',

turn: 0,

log: [],

pinguiniaUnlocked: false,

religionUnlocked: false,

terrorismUnlocked: false,

ipGenerationRate: 1,

ipGenerationMultiplier: 1,

influencePowerMultiplier: 1,

aiIPGenerationMultiplier: 1,

isAutosaveEnabled: false,

aiLastTarget: null,

playerLastTarget: null,

dailyBonusCycle: 0,

specialBonusIndex: 0,

lastBonusDate: null,

nationalistSurgeCountry: null,

espionageNetworkActive: false

};

let autosaveIntervalId = null;

let currentTutorialStep = 0;

let pauseOnTabSwitch = false;

let gameInterval = null;

// Game Data

const TUTORIAL\_STEPS = [

{

title: "Welcome to Global Ideology!",

description: "Your mission is simple: achieve global domination by strategically outmanoeuvring an AI opponent and gaining 100% influence over all countries. The world is yours for the taking."

},

{

title: "The Main Screen",

description: "This is your main command screen. Here, you can monitor your resources, track the influence battle on the world grid, and see the latest events in the game log."

},

{

title: "Your Resources",

description: "Your primary resource is Influence Points (IP), which generate automatically. Use IP to influence countries and purchase upgrades. You can earn Munee (Ξ), the premium currency, to buy powerful boosts and unlock new features from the Shop."

},

{

title: "The Country Grid",

description: "The world is divided into countries, each with unique resistances. Click on any country tile to view its stats and influence it. As you gain influence, the blue bar on the tile will grow. The AI's influence is shown in red."

},

{

title: "Country Resistances",

description: "Every country has four resistance stats: Military, Economic, Diplomatic, and Ideological. To influence a country, you must spend IP against one of these resistances. The lower a country's resistance is to a specific type, the more effective your IP will be."

},

{

title: "The AI Opponent",

description: "You are not alone in this world. A strategic AI is also working to spread its ideology. The AI will constantly generate IP and attempt to influence countries, just like you. Keep an eye on its progress and adapt your strategy to counter its moves."

},

{

title: "Upgrades and The Shop",

description: "Use the Upgrade and Shop buttons to gain an advantage. The Upgrade screen allows you to spend IP on permanent bonuses. In the Shop, you can spend Munee to unlock new upgrade trees or even a new country to influence."

}

];

const COUNTRIES\_DATA = [

{ name: 'Aethelgard', military: 400, economic: 400, diplomatic: 400, ideological: 100, description: 'A sprawling federation of states with a highly diverse population and a powerful, though sometimes divided, central government. Its immense size and complex politics make it a challenging target for a single, focused influence campaign.' },

{ name: 'Guldor', military: 200, economic: 200, diplomatic: 400, ideological: 200, description: 'An ancient and influential kingdom with a rich history of trade, exploration, and cultural dominance. Though now a modern state, its diplomatic and economic power is built upon centuries of established relationships and immense wealth.' },

{ name: 'Koryn', military: 300, economic: 200, diplomatic: 200, ideological: 300, description: 'A large, resource-rich nation with a centralized, authoritarian government. It has a proud military history and is a major player on the world stage, but its rigid political structure makes it resistant to outside ideological influence.' },

{ name: 'Sinopeia', military: 300, economic: 400, diplomatic: 300, ideological: 300, description: 'A vast and ancient empire with a rapidly expanding economy and a disciplined, centrally controlled society. Its sheer scale and technological ambition make it a global powerhouse, but its resistance to foreign ideologies is unmatched.' },

{ name: 'Juche', military: 100, economic: 100, diplomatic: 100, ideological: 400, description: 'A fiercely isolated and insular nation, fortified by a rigid political doctrine and a cult of personality. Its people are largely inaccessible to outside influence, making it one of the most difficult targets to penetrate.' },

{ name: 'Bharat', military: 300, economic: 200, diplomatic: 200, ideological: 300, description: 'A colossal democracy known for its vibrant culture, massive population, and burgeoning technology sector. Its internal diversity and complex political landscape create both great strength and numerous vulnerabilities.' },

{ name: 'La Plata', military: 300, economic: 100, diplomatic: 100, ideological: 100, description: 'A country of great natural beauty and vast agricultural wealth, but with a history of political instability and economic turmoil. Its strategic location makes it a key player in its region, but it is often susceptible to economic influence.' },

{ name: 'Pannonia', military: 100, economic: 100, diplomatic: 200, ideological: 200, description: 'A landlocked European state with a proud history and a distinct cultural identity. It is a member of major diplomatic alliances, but its internal politics often put it at odds with its neighbors, making it a difficult and unpredictable target.' },

{ name: 'Helvetia', military: 100, economic: 400, diplomatic: 300, ideological: 200, description: 'A neutral and highly prosperous alpine republic. Its global financial influence is immense, and its unique political structure and fortified borders make it an extremely difficult target for any form of military or ideological influence.' },

{ name: 'Nippon', military: 300, economic: 400, diplomatic: 400, ideological: 300, description: 'An isolated island nation with a hyper-advanced economy and a highly disciplined society. While a leader in technology, its aging population and dependence on foreign resources can be exploited.' },

{ name: 'Isolde', military: 100, economic: 300, diplomatic: 200, ideological: 200, description: 'A remote island nation of fire and ice, with an active volcanic landscape and abundant geothermal energy. Its small, close-knit population is known for its cultural heritage and self-reliance, making it highly resistant to outside cultural or ideological influence.' },

{ name: 'Borealis', military: 300, economic: 300, diplomatic: 300, ideological: 300, description: 'A sprawling northern dominion rich in natural resources and defined by its vast, untamed wilderness. Its diverse population is spread across its southern border, fostering a decentralized political structure that is both tolerant and difficult to influence as a single entity.' },

{ name: 'Pinguinia', military: 500, economic: 500, diplomatic: 500, ideological: 500, description: 'A frigid polar nation nestled among colossal ice shelves and snow-swept mountains. Though its climate is harsh, its inhabitants have forged a technologically advanced and fiercely cooperative society. Pinguinia\'s strategic value lies not in its resources, which are scarce, but in its unparalleled scientific prowess and its political neutrality. It is a coveted prize for any faction, as it grants unparalleled diplomatic and technological advantages.', purchasable: true }

];

const UPGRADES\_DATA = {

ideology: [

{ name: 'Charismatic leadership', cost: 100, requires: [] },

{ name: 'Doctrinal purity', cost: 300, requires: ['Charismatic leadership'] },

{ name: 'Social engineering', cost: 500, requires: ['Doctrinal purity'] },

{ name: 'Utopia', cost: 1100, requires: ['Social engineering'] }

],

military: [

{ name: 'Military Parade', cost: 100, requires: [] },

{ name: 'Cyber Operations', cost: 300, requires: ['Military Parade'] },

{ name: 'Sovereign Intervention', cost: 500, requires: ['Cyber Operations'] },

{ name: 'Bombing infrastructure', cost: 1100, requires: ['Sovereign Intervention'] }

],

diplomatic: [

{ name: 'Embassy', cost: 50, requires: [] },

{ name: 'Regional Summit', cost: 250, requires: ['Embassy'] },

{ name: 'Sanctions Hammer', cost: 550, requires: ['Regional Summit'] },

{ name: 'Alliance Breaker', cost: 1100, requires: ['Sanctions Hammer'] }

],

economic: [

{ name: 'Trade Deals', cost: 150, requires: [] },

{ name: 'Resource Leverage', cost: 300, requires: ['Trade Deals'] },

{ name: 'Monopolization', cost: 600, requires: ['Resource Leverage'] },

{ name: 'Government Bribe', cost: 1100, requires: ['Monopolization'] }

],

religion: [

{ name: 'Word of God', cost: 100, requires: [] },

{ name: 'Church of Holy Molly', cost: 200, requires: ['Word of God'] },

{ name: 'Messiah', cost: 300, requires: ['Church of Holy Molly'] },

{ name: 'Believe!', cost: 1000, requires: ['Messiah'] }

],

terrorism: [

{ name: 'Dormant cell', cost: 100, requires: [] },

{ name: 'Asset degradation', cost: 300, requires: ['Dormant cell'] },

{ name: 'Operation Cyclone', cost: 590, requires: ['Asset degradation'] },

{ name: 'Hybrid threat', cost: 1000, requires: ['Operation Cyclone'] }

]

};

const SHOP\_ITEMS = {

unlockables: [

{ name: 'Pinguinia Country', muneeCost: 5.00, item: 'pinguinia', unlocked: () => gameState.pinguiniaUnlocked },

{ name: 'Religion Upgrade Tree', muneeCost: 3.00, item: 'religion', unlocked: () => gameState.religionUnlocked },

{ name: 'Terrorism Upgrade Tree', muneeCost: 4.00, item: 'terrorism', unlocked: () => gameState.terrorismUnlocked }

],

boosts: [

{ name: 'IP Surge (x2 IP Generation)', muneeCost: 1.50, item: 'ip\_surge', duration: 30000 },

{ name: 'Double Influence (x2 Influence Power)', muneeCost: 2.50, item: 'double\_influence', duration: 60000 }

]

};

const AI\_PERSONALITIES = ['Aggressor', 'Opportunist', 'Strategist'];

const WORLD\_EVENTS = [

{

name: 'Political Unrest',

type: 'negative',

effect: (country) => {

country.playerInfluence \*= 0.75;

country.aiInfluence \*= 0.75;

}

},

{

name: 'Natural Disaster',

type: 'negative',

effect: (country) => {

const oldMil = country.military;

const oldEco = country.economic;

const oldDip = country.diplomatic;

const oldIde = country.ideological;

country.military += 100;

country.economic += 100;

country.diplomatic += 100;

country.ideological -= 50;

setTimeout(() => {

country.military = oldMil;

country.economic = oldEco;

country.diplomatic = oldDip;

country.ideological = oldIde;

}, 30000);

}

},

{

name: 'Economic Boom',

type: 'positive',

effect: (country) => {

const oldEconomicResistance = country.economic;

country.economic = oldEconomicResistance \* 0.75;

setTimeout(() => {

country.economic = oldEconomicResistance;

}, 30000);

}

},

{

name: 'Alliance Formation',

type: 'positive',

effect: (country) => {

country.diplomatic \*= 1.5;

}

},

{

name: 'Terrorist Attack',

type: 'negative',

effect: (country) => {

country.playerInfluence \*= 0.5;

country.aiInfluence \*= 0.5;

country.military += 100;

}

}

];

const DAILY\_IP\_BONUSES = [50, 50, 100, 100, 150, 200];

const SPECIAL\_BONUSES = [

'Intelligence Leak',

'Espionage Network',

'Economic Sabotage',

'AI Counter-Influence',

'The "Great Wall" Protocol',

'Global Unity',

'Super Siphon',

'Nationalist Surge',

'Targeted Resistance Collapse',

'Global Perception Boost',

'Diplomatic Coup',

'IP Jackpot'

];

// Utility Functions

function logMessage(message) {

if (!elements['game-log-list']) return;

gameState.log.unshift(message);

if (gameState.log.length > 5) {

gameState.log.pop();

}

renderLog();

}

function renderLog() {

if (!elements['game-log-list']) return;

elements['game-log-list'].innerHTML = '';

gameState.log.forEach(msg => {

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

li.textContent = msg;

elements['game-log-list'].appendChild(li);

});

}

function openModal(modal) {

if (!modal) return;

elements.modals.forEach(m => {

if (m) {

m.classList.remove('active');

m.close && m.close();

}

});

modal.classList.add('active');

if (modal.showModal) {

modal.showModal();

} else {

modal.style.display = 'flex';

}

}

function closeModal(modal) {

if (!modal) return;

modal.classList.remove('active');

if (modal.close) {

modal.close();

} else {

modal.style.display = 'none';

}

}

function showScreen(screenId) {

document.querySelectorAll('.screen').forEach(screen => {

screen.classList.remove('active-screen');

screen.hidden = true;

});

const screen = elements[screenId];

if (screen) {

screen.classList.add('active-screen');

screen.hidden = false;

}

}

function updateUI() {

// Update player stats

if (elements['player-ip-count']) {

elements['player-ip-count'].textContent = gameState.playerIP.toFixed(2);

}

if (elements['player-munee-count']) {

elements['player-munee-count'].textContent = gameState.playerMunee.toFixed(2);

}

if (elements['ai-ip-count']) {

elements['ai-ip-count'].textContent = gameState.aiIP.toFixed(2);

}

if (elements['modal-player-ip']) {

elements['modal-player-ip'].textContent = gameState.playerIP.toFixed(2);

}

renderCountryGrid();

}

function getResistanceColor(value) {

if (value < 200) {

return 'lime';

} else if (value < 400) {

return 'yellow';

} else {

return 'red';

}

}

// Game Initialization

function startLoading() {

let progress = 0;

const loadingInterval = setInterval(() => {

progress += 10;

if (elements['loadingProgress']) {

elements['loadingProgress'].style.width = progress + '%';

}

if (progress >= 100) {

clearInterval(loadingInterval);

if (elements['loadingBar']) {

elements['loadingBar'].style.display = 'none';

}

setTimeout(() => {

openModal(elements['main-menu-modal']);

if (localStorage.getItem('globalIdeologySave')) {

if (elements['modal-continue-btn']) {

elements['modal-continue-btn'].disabled = false;

}

}

}, 500);

}

}, 300);

}

function startTutorial() {

currentTutorialStep = 0;

updateTutorialModal();

openModal(elements['tutorial-modal']);

}

function updateTutorialModal() {

const step = TUTORIAL\_STEPS[currentTutorialStep];

if (!step || !elements['tutorial-title'] || !elements['tutorial-description']) return;

elements['tutorial-title'].textContent = step.title;

elements['tutorial-description'].textContent = step.description;

if (elements['tutorial-back-btn']) {

elements['tutorial-back-btn'].style.display = currentTutorialStep > 0 ? 'block' : 'none';

}

if (elements['tutorial-next-btn']) {

elements['tutorial-next-btn'].textContent = currentTutorialStep === TUTORIAL\_STEPS.length - 1 ? 'Finish' : 'Next';

}

if (elements['tutorial-options']) {

elements['tutorial-options'].style.display = currentTutorialStep === TUTORIAL\_STEPS.length - 1 ? 'block' : 'none';

}

}

function endTutorial() {

closeModal(elements['tutorial-modal']);

localStorage.setItem('hasSeenTutorial', true);

showCountrySelection();

}

function showCountrySelection() {

showScreen('country-selection-screen');

renderCountrySelectionGrid();

}

function renderCountrySelectionGrid() {

if (!elements['selection-grid']) return;

elements['selection-grid'].innerHTML = '';

const countriesToSelect = COUNTRIES\_DATA.filter(c => !c.purchasable);

countriesToSelect.forEach(country => {

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

tile.classList.add('country-tile');

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

infoBtn.classList.add('info-btn');

infoBtn.textContent = 'i';

infoBtn.addEventListener('click', (e) => {

e.stopPropagation();

openCountryModal(country, true);

});

tile.appendChild(infoBtn);

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

nameEl.textContent = country.name;

tile.appendChild(nameEl);

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

elements['selection-grid'].appendChild(tile);

});

}

function selectPlayerCountry(countryName) {

gameState.playerCountry = countryName;

let aiCountry;

do {

aiCountry = COUNTRIES\_DATA[Math.floor(Math.random() \* COUNTRIES\_DATA.length)];

} while (aiCountry.name === gameState.playerCountry || aiCountry.purchasable);

gameState.aiCountry = aiCountry.name;

gameState.aiPersonality = AI\_PERSONALITIES[Math.floor(Math.random() \* AI\_PERSONALITIES.length)];

startGame();

}

function startGame(isLoadedGame = false) {

showScreen('main-game-screen');

if (!isLoadedGame) {

gameState.countries = COUNTRIES\_DATA.map(c => ({

...c,

playerInfluence: c.name === gameState.playerCountry ? 100 : 0,

aiInfluence: c.name === gameState.aiCountry ? 100 : 0,

isNationalistSurge: false

}));

// Only load the default upgrade trees at the start of the game

const initialUpgrades = ['ideology', 'military', 'diplomatic', 'economic'];

gameState.upgrades = {};

initialUpgrades.forEach(branch => {

gameState.upgrades[branch] = UPGRADES\_DATA[branch].map(u => ({

...u,

active: false

}));

});

logMessage(`Game started. You are in control of ${gameState.playerCountry}. The AI controls ${gameState.aiCountry}.`);

logMessage(`AI personality is: ${gameState.aiPersonality}`);

} else {

logMessage('Game loaded successfully.');

logMessage(`AI personality is: ${gameState.aiPersonality}`);

}

checkRealTimeBonus();

updateUI();

gameLoop();

logMessage('Click on a country to influence it.');

}

// Game Logic

function renderCountryGrid() {

if (!elements['country-grid']) return;

elements['country-grid'].innerHTML = '';

gameState.countries.forEach(country => {

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

tile.classList.add('country-tile');

tile.dataset.name = country.name;

const isLocked = country.purchasable && !gameState.pinguiniaUnlocked;

if (isLocked) {

tile.classList.add('locked');

}

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

nameEl.textContent = isLocked ? '???' : country.name;

tile.appendChild(nameEl);

if (!isLocked) {

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

influenceBar.classList.add('influence-bar');

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

playerBar.classList.add('player');

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

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

aiBar.classList.add('ai');

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

influenceBar.appendChild(playerBar);

influenceBar.appendChild(aiBar);

tile.appendChild(influenceBar);

}

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

infoBtn.classList.add('info-btn');

infoBtn.textContent = 'i';

infoBtn.addEventListener('click', (e) => {

e.stopPropagation();

openCountryModal(country, true);

});

tile.appendChild(infoBtn);

if (!isLocked) {

tile.addEventListener('click', () => openCountryModal(country));

} else {

tile.addEventListener('click', () => logMessage(`${country.name} is locked. Buy it in the shop first!`));

}

elements['country-grid'].appendChild(tile);

});

}

function openCountryModal(country, isInfoOnly = false) {

if (!elements['country-name'] || !elements['country-description']) return;

elements['country-name'].textContent = country.name;

elements['country-description'].textContent = country.description;

// Update stats with colors

const stats = ['military', 'economic', 'diplomatic', 'ideological'];

stats.forEach(stat => {

const element = elements[`stat-${stat}`];

if (element) {

element.textContent = country[stat];

element.style.color = getResistanceColor(country[stat]);

}

});

// Update player IP in modal

if (elements['modal-player-ip']) {

elements['modal-player-ip'].textContent = gameState.playerIP.toFixed(2);

}

const actionsContainer = elements['country-actions-container'];

const upgradeBtn = elements['upgrade-country-btn'];

if (isInfoOnly) {

if (actionsContainer) actionsContainer.style.display = 'none';

if (upgradeBtn) upgradeBtn.style.display = 'none';

} else {

if (actionsContainer) actionsContainer.style.display = 'block';

if (upgradeBtn) {

upgradeBtn.style.display = country.name === gameState.playerCountry ? 'block' : 'none';

upgradeBtn.onclick = () => {

openModal(elements['upgrade-modal']);

renderUpgrades();

};

}

}

// Set up influence buttons

const influenceBtns = document.querySelectorAll('#action-buttons button');

influenceBtns.forEach(btn => {

btn.onclick = () => influenceCountry(

country.name,

btn.dataset.type,

parseInt(elements['ip-amount'].value || 40)

);

});

openModal(elements['country-modal']);

}

function influenceCountry(countryName, type, amount) {

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

if (!country) return;

if (countryName === gameState.playerCountry) {

logMessage("You already fully control your starting country.");

return;

}

if (amount < 40) {

logMessage("You must spend a minimum of 40 IP to influence a country.");

return;

}

let costMultiplier = 1;

if (country.playerInfluence >= 40 && country.aiInfluence >= 40) {

if (country.playerInfluence >= 80 || country.aiInfluence >= 80) {

costMultiplier = 2.5;

} else if (country.playerInfluence >= 60 || country.aiInfluence >= 60) {

costMultiplier = 2.0;

} else {

costMultiplier = 1.5;

}

logMessage(`Contested influence detected in ${country.name}. Action cost is multiplied by ${costMultiplier}!`);

}

const ipCost = amount \* costMultiplier \* gameState.influencePowerMultiplier;

if (gameState.playerIP < ipCost) {

logMessage("Not enough IP to influence this country.");

return;

}

gameState.playerIP -= ipCost;

const resistance = country[type.toLowerCase()];

const influenceGain = amount / resistance;

const aiInfluenceLoss = influenceGain \* 0.3;

country.aiInfluence = Math.max(0, country.aiInfluence - aiInfluenceLoss);

country.playerInfluence += influenceGain \* 10;

logMessage(`Player influenced ${country.name} with ${ipCost.toFixed(2)} IP using ${type}.`);

gameState.playerLastTarget = country.name;

updateUI();

checkWinCondition();

}

function aiTurn() {

let targetCountry = null;

let influenceType = '';

let amount = 0;

const playerCountries = gameState.countries.filter(c =>

c.playerInfluence > c.aiInfluence && c.name !== gameState.playerCountry

);

const neutralCountries = gameState.countries.filter(c =>

c.playerInfluence === c.aiInfluence &&

c.name !== gameState.playerCountry &&

c.name !== gameState.aiCountry &&

!c.purchasable

);

// Find country affected by event

const affectedCountry = gameState.countries.find(c => c.isAffectedByEvent);

if (affectedCountry) {

targetCountry = affectedCountry;

const resistances = ['military', 'economic', 'diplomatic', 'ideological'];

influenceType = resistances.sort((a, b) => affectedCountry[a] - affectedCountry[b])[0];

amount = 100 + Math.random() \* 60;

} else {

switch (gameState.aiPersonality) {

case 'Aggressor':

if (playerCountries.length > 0) {

targetCountry = playerCountries[Math.floor(Math.random() \* playerCountries.length)];

} else {

targetCountry = neutralCountries[Math.floor(Math.random() \* neutralCountries.length)];

}

influenceType = ['military', 'terrorism'][Math.floor(Math.random() \* 2)];

amount = 100 + Math.random() \* 60;

break;

case 'Opportunist':

const allCountries = gameState.countries.filter(c => c.name !== gameState.playerCountry);

targetCountry = allCountries.sort((a, b) => {

const lowestResA = Math.min(a.military, a.economic, a.diplomatic, a.ideological);

const lowestResB = Math.min(b.military, b.economic, b.diplomatic, b.ideological);

return lowestResA - lowestResB;

})[0];

const resistances = ['military', 'economic', 'diplomatic', 'ideological'];

influenceType = resistances.sort((a, b) => targetCountry[a] - targetCountry[b])[0];

amount = 40 + Math.random() \* 40;

break;

case 'Strategist':

const highValueTargets = gameState.countries.filter(c =>

c.economic > 300 && c.playerInfluence < 50 && c.aiInfluence < 50

);

if (highValueTargets.length > 0) {

targetCountry = highValueTargets[Math.floor(Math.random() \* highValueTargets.length)];

influenceType = 'economic';

} else if (neutralCountries.length > 0) {

targetCountry = neutralCountries[Math.floor(Math.random() \* neutralCountries.length)];

influenceType = 'economic';

} else {

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

influenceType = 'economic';

}

amount = 60 + Math.random() \* 80;

break;

}

}

if (!targetCountry) {

const allCountries = gameState.countries.filter(c => c.name !== gameState.playerCountry);

targetCountry = allCountries[Math.floor(Math.random() \* allCountries.length)];

influenceType = ['military', 'economic', 'diplomatic', 'ideological'][Math.floor(Math.random() \* 4)];

amount = 40;

}

let costMultiplier = 1;

if (targetCountry.playerInfluence >= 40 && targetCountry.aiInfluence >= 40) {

if (targetCountry.playerInfluence >= 80 || targetCountry.aiInfluence >= 80) {

costMultiplier = 2.5;

} else if (targetCountry.playerInfluence >= 60 || targetCountry.aiInfluence >= 60) {

costMultiplier = 2.0;

} else {

costMultiplier = 1.5;

}

}

const ipCost = amount \* costMultiplier \* gameState.aiIPGenerationMultiplier;

if (targetCountry.isNationalistSurge) {

logMessage(`AI attempted to influence ${targetCountry.name}, but was blocked by the "Nationalist Surge" bonus.`);

return;

}

amount = Math.min(ipCost, gameState.aiIP);

if (amount >= 40) {

gameState.aiIP -= amount;

const resistance = targetCountry[influenceType.toLowerCase()];

const influenceGain = amount / resistance;

targetCountry.aiInfluence += influenceGain \* 10;

if (gameState.espionageNetworkActive) {

logMessage(`[INTELLIGENCE LEAK] The AI is targeting ${targetCountry.name} with ${influenceType} influence.`);

}

logMessage(`AI (${gameState.aiPersonality}) influenced ${targetCountry.name} with ${amount.toFixed(2)} IP using ${influenceType}.`);

gameState.aiLastTarget = targetCountry.name;

updateUI();

checkLossCondition();

}

}

function gameLoop() {

if (gameState.gameSpeed === 'pause') {

return;

}

// Generate IP

gameState.playerIP += (gameState.ipGenerationRate + (gameState.espionageNetworkActive ? 1 : 0)) \* gameState.ipGenerationMultiplier;

gameState.aiIP += gameState.ipGenerationRate \* gameState.aiIPGenerationMultiplier;

// AI turn

aiTurn();

// Check for world events

checkWorldEvents();

// Update UI and increment turn counter

updateUI();

gameState.turn++;

// Schedule next turn

const interval = gameState.gameSpeed === 'play' ? 1000 : 500;

gameInterval = setTimeout(gameLoop, interval);

}

function checkWorldEvents() {

// Only check for a world event every 10 turns

if (gameState.turn % 10 === 0) {

const roll = Math.random();

if (roll < 0.03) {

const availableCountries = gameState.countries.filter(c =>

!c.purchasable && c.name !== gameState.playerCountry && c.name !== gameState.aiCountry

);

if (availableCountries.length === 0) return;

const event = WORLD\_EVENTS[Math.floor(Math.random() \* WORLD\_EVENTS.length)];

const country = availableCountries[Math.floor(Math.random() \* availableCountries.length)];

logMessage(`[EVENT] ${event.name} has occurred in ${country.name}.`);

event.effect(country);

if (event.type === 'negative') {

const countryWithSurge = gameState.countries.find(c => c.isNationalistSurge);

if (countryWithSurge) {

countryWithSurge.isNationalistSurge = false;

logMessage(`[NATIONALIST SURGE ENDED] The negative event in ${country.name} has ended the "Nationalist Surge" bonus in ${countryWithSurge.name}.`);

}

}

}

}

}

function setGameSpeed(speed) {

clearTimeout(gameInterval);

gameState.gameSpeed = speed;

// Update button states

if (elements['pause-btn'] && elements['play-btn'] && elements['ffwd-btn']) {

elements['pause-btn'].setAttribute('aria-pressed', speed === 'pause');

elements['play-btn'].setAttribute('aria-pressed', speed === 'play');

elements['ffwd-btn'].setAttribute('aria-pressed', speed === 'ffwd');

}

if (speed !== 'pause') {

gameLoop();

}

logMessage(`Game speed set to ${speed}.`);

}

function checkRealTimeBonus() {

const today = new Date().toDateString();

if (gameState.lastBonusDate === today) {

return;

}

gameState.lastBonusDate = today;

gameState.dailyBonusCycle++;

if (gameState.dailyBonusCycle % 7 === 0) {

const bonusName = SPECIAL\_BONUSES[gameState.specialBonusIndex];

logMessage(`[DAILY BONUS] You have received a special bonus: ${bonusName}!`);

showDailyBonusModal(`Special Bonus: ${bonusName}!`, `You have received a special bonus: ${bonusName}!`);

activateSpecialBonus(bonusName);

gameState.specialBonusIndex++;

if (gameState.specialBonusIndex >= SPECIAL\_BONUSES.length) {

gameState.specialBonusIndex = 0;

}

} else {

const ipBonus = DAILY\_IP\_BONUSES[(gameState.dailyBonusCycle - 1) % 6];

gameState.playerIP += ipBonus;

logMessage(`[DAILY BONUS] You have received a ${ipBonus} IP bonus for day ${gameState.dailyBonusCycle}.`);

showDailyBonusModal("Daily IP Bonus!", `You have received a <strong>${ipBonus} IP</strong> bonus.`);

}

saveGame();

}

function showDailyBonusModal(title, message) {

if (!elements['bonus-title'] || !elements['bonus-message']) return;

elements['bonus-title'].textContent = title;

elements['bonus-message'].innerHTML = message;

openModal(elements['daily-bonus-modal']);

}

function activateSpecialBonus(bonusName) {

switch(bonusName) {

case 'Intelligence Leak':

gameState.espionageNetworkActive = true;

logMessage("[SPECIAL BONUS] Intelligence Leak activated! You can now see AI moves.");

break;

case 'Espionage Network':

gameState.ipGenerationRate += 1;

logMessage("[SPECIAL BONUS] Espionage Network activated! +1 to IP generation rate.");

break;

case 'Nationalist Surge':

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

if (country) {

country.isNationalistSurge = true;

gameState.nationalistSurgeCountry = country.name;

logMessage(`[SPECIAL BONUS] "Nationalist Surge" has been applied to ${country.name}. It will last until the next negative event.`);

}

break;

case 'IP Jackpot':

const jackpotAmount = 500;

gameState.playerIP += jackpotAmount;

logMessage(`[SPECIAL BONUS] IP Jackpot! You received ${jackpotAmount} IP!`);

break;

default:

logMessage(`[SPECIAL BONUS] "${bonusName}" is not yet implemented.`);

}

}

// Save/Load System

function saveGame() {

try {

localStorage.setItem('globalIdeologySave', JSON.stringify(gameState));

logMessage('Game saved successfully!');

} catch (e) {

logMessage('Error saving game. Local storage may be full or disabled.');

console.error('Save error:', e);

}

}

function loadGame() {

try {

const savedState = JSON.parse(localStorage.getItem('globalIdeologySave'));

if (savedState) {

gameState = savedState;

setGameSpeed(gameState.gameSpeed);

if (gameState.isAutosaveEnabled) {

if (elements['autosave-checkbox']) {

elements['autosave-checkbox'].checked = true;

}

startAutosaveTimer();

}

startGame(true);

}

} catch (e) {

logMessage('Error loading game. Save file may be corrupted.');

console.error('Load error:', e);

}

}

function startAutosaveTimer() {

if (autosaveIntervalId) {

clearInterval(autosaveIntervalId);

}

autosaveIntervalId = setInterval(() => {

saveGame();

}, 5 \* 60 \* 1000); // 5 minutes

logMessage('Autosave enabled (every 5 minutes).');

}

function stopAutosaveTimer() {

if (autosaveIntervalId) {

clearInterval(autosaveIntervalId);

autosaveIntervalId = null;

}

logMessage('Autosave disabled.');

}

// Upgrade System

function renderUpgrades() {

if (!elements['upgrade-tree-container']) return;

elements['upgrade-tree-container'].innerHTML = '';

for (const branch in gameState.upgrades) {

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

branchEl.classList.add('upgrade-branch');

const branchTitle = document.createElement('h3');

branchTitle.textContent = branch.charAt(0).toUpperCase() + branch.slice(1);

branchEl.appendChild(branchTitle);

gameState.upgrades[branch].forEach(upgrade => {

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

upgradeEl.classList.add('upgrade-item');

upgradeEl.classList.add(upgrade.active ? 'active' : 'inactive');

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

upgradeName.textContent = upgrade.name;

upgradeEl.appendChild(upgradeName);

const costEl = document.createElement('span');

costEl.textContent = `Cost: ${upgrade.cost} IP`;

upgradeEl.appendChild(costEl);

// Show requirements

if (upgrade.requires.length > 0) {

const reqEl = document.createElement('small');

reqEl.textContent = `Requires: ${upgrade.requires.join(', ')}`;

reqEl.style.display = 'block';

reqEl.style.marginTop = '5px';

upgradeEl.appendChild(reqEl);

}

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

buyBtn.textContent = upgrade.active ? 'Active' : 'Buy';

buyBtn.disabled = upgrade.active || gameState.playerIP < upgrade.cost;

if (!upgrade.active) {

// Check if requirements are met

const allUpgrades = Object.values(gameState.upgrades).flat();

const requirementsMet = upgrade.requires.every(req => {

const requiredUpgrade = allUpgrades.find(u => u.name === req);

return requiredUpgrade && requiredUpgrade.active;

});

if (!requirementsMet) {

buyBtn.disabled = true;

buyBtn.title = 'Requirements not met';

}

}

buyBtn.onclick = () => buyUpgrade(branch, upgrade.name);

upgradeEl.appendChild(buyBtn);

branchEl.appendChild(upgradeEl);

});

elements['upgrade-tree-container'].appendChild(branchEl);

}

}

function buyUpgrade(branch, upgradeName) {

const upgrade = gameState.upgrades[branch].find(u => u.name === upgradeName);

if (!upgrade || upgrade.active || gameState.playerIP < upgrade.cost) {

logMessage(`Cannot buy ${upgradeName}.`);

return;

}

// Check requirements

const allUpgrades = Object.values(gameState.upgrades).flat();

const requirementsMet = upgrade.requires.every(req => {

const requiredUpgrade = allUpgrades.find(u => u.name === req);

return requiredUpgrade && requiredUpgrade.active;

});

if (!requirementsMet) {

logMessage(`Cannot buy ${upgradeName}. Missing requirements: ${upgrade.requires.join(', ')}`);

return;

}

gameState.playerIP -= upgrade.cost;

upgrade.active = true;

// Apply upgrade effects

applyUpgradeEffect(upgradeName);

logMessage(`Upgrade activated: ${upgradeName}.`);

updateUI();

renderUpgrades();

}

function applyUpgradeEffect(upgradeName) {

switch(upgradeName) {

case 'Charismatic leadership':

gameState.influencePowerMultiplier \*= 1.2;

break;

case 'Doctrinal purity':

gameState.ipGenerationMultiplier \*= 1.3;

break;

case 'Military Parade':

// Military upgrades affect AI targeting

break;

case 'Cyber Operations':

gameState.aiIPGenerationMultiplier \*= 0.9;

break;

case 'Trade Deals':

gameState.ipGenerationRate += 0.5;

break;

case 'Resource Leverage':

gameState.influencePowerMultiplier \*= 1.15;

break;

}

}

// Shop System

function showShopTab(tab) {

if (!elements['shop-container']) return;

elements['shop-container'].innerHTML = '';

const items = SHOP\_ITEMS[tab];

if (!items) return;

// Update tab states

document.querySelectorAll('#shopTabs button').forEach(btn => {

btn.setAttribute('aria-selected', btn.textContent.toLowerCase() === tab);

});

// Show/hide tab panels

document.querySelectorAll('#shop-container > section').forEach(section => {

section.hidden = section.id !== `${tab}-tab`;

});

items.forEach(item => {

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

itemEl.classList.add('shop-item');

itemEl.innerHTML = `

<p><strong>${item.name}</strong></p>

<p>Cost: Ξ${item.muneeCost.toFixed(2)}</p>

<button class="buy-btn" data-item="${item.item}">Buy</button>

`;

const buyBtn = itemEl.querySelector('.buy-btn');

if (item.unlocked && item.unlocked()) {

buyBtn.disabled = true;

buyBtn.textContent = "Unlocked";

buyBtn.classList.add('disabled');

}

buyBtn.onclick = () => {

if (item.item === 'ip\_surge' || item.item === 'double\_influence') {

buyBoost(item.name, item.muneeCost, item.duration, item.item);

} else {

buyShopItem(item.item, item.muneeCost);

}

};

elements['shop-container'].appendChild(itemEl);

});

}

function buyShopItem(item, cost) {

if (gameState.playerMunee < cost) {

logMessage(`Not enough Munee to buy ${item}. You need Ξ${cost.toFixed(2)}.`);

return;

}

gameState.playerMunee -= cost;

if (item === 'pinguinia') {

gameState.pinguiniaUnlocked = true;

const pinguiniaCountry = COUNTRIES\_DATA.find(c => c.name === 'Pinguinia');

if (pinguiniaCountry) {

gameState.countries.push({

...pinguiniaCountry,

playerInfluence: 0,

aiInfluence: 0

});

}

logMessage('Pinguinia has been unlocked and added to the world map!');

} else if (item === 'religion') {

gameState.religionUnlocked = true;

gameState.upgrades.religion = UPGRADES\_DATA.religion.map(u => ({ ...u, active: false }));

logMessage('Religion upgrade tree has been unlocked!');

} else if (item === 'terrorism') {

gameState.terrorismUnlocked = true;

gameState.upgrades.terrorism = UPGRADES\_DATA.terrorism.map(u => ({ ...u, active: false }));

logMessage('Terrorism upgrade tree has been unlocked!');

}

closeModal(elements['shop-modal']);

updateUI();

saveGame();

}

function buyBoost(name, cost, duration, item) {

if (gameState.playerMunee < cost) {

logMessage(`Not enough Munee to buy ${name}. You need Ξ${cost.toFixed(2)}.`);

return;

}

gameState.playerMunee -= cost;

logMessage(`${name} activated!`);

activateTemporaryBoost(item, duration);

closeModal(elements['shop-modal']);

updateUI();

saveGame();

}

function activateTemporaryBoost(item, duration) {

let oldValue;

if (item === 'ip\_surge') {

oldValue = gameState.ipGenerationMultiplier;

gameState.ipGenerationMultiplier \*= 2;

logMessage('IP generation doubled for 30 seconds!');

} else if (item === 'double\_influence') {

oldValue = gameState.influencePowerMultiplier;

gameState.influencePowerMultiplier \*= 2;

logMessage('Influence power doubled for 60 seconds!');

}

setTimeout(() => {

if (item === 'ip\_surge') {

gameState.ipGenerationMultiplier = oldValue;

} else if (item === 'double\_influence') {

gameState.influencePowerMultiplier = oldValue;

}

logMessage(`${item.replace('\_', ' ').toUpperCase()} boost has ended.`);

}, duration);

}

// Win/Loss Conditions

function checkWinCondition() {

const playerControlledCountries = gameState.countries.filter(c =>

c.playerInfluence > c.aiInfluence

);

if (playerControlledCountries.length === gameState.countries.length) {

showGameOverModal('Victory!', 'You have achieved global domination! All countries are under your influence.');

setGameSpeed('pause');

}

}

function checkLossCondition() {

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

if (playerCountry && playerCountry.aiInfluence >= 100) {

showGameOverModal('Defeat!', 'The AI has taken control of your starting country. Your ideology has been overthrown.');

setGameSpeed('pause');

}

}

function showGameOverModal(title, message) {

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

modal.className = 'modal active';

modal.innerHTML = `

<div class="modal-content">

<h2>${title}</h2>

<p>${message}</p>

<div class="modal-buttons">

<button id="restart-btn" class="menu-btn">New Game</button>

<button id="main-menu-btn" class="menu-btn">Main Menu</button>

</div>

</div>

`;

document.getElementById('game-container').appendChild(modal);

// Add event listeners

setTimeout(() => {

const restartBtn = document.getElementById('restart-btn');

const mainMenuBtn = document.getElementById('main-menu-btn');

if (restartBtn) {

restartBtn.onclick = () => {

modal.remove();

startGame();

};

}

if (mainMenuBtn) {

mainMenuBtn.onclick = () => {

modal.remove();

showScreen('loading-screen');

setTimeout(() => {

openModal(elements['main-menu-modal']);

}, 500);

};

}

}, 100);

}

// Event Listeners

function initializeEventListeners() {

// Main menu buttons

if (elements['modal-start-btn']) {

elements['modal-start-btn'].addEventListener('click', () => {

closeModal(elements['main-menu-modal']);

if (localStorage.getItem('hasSeenTutorial')) {

showCountrySelection();

} else {

startTutorial();

}

});

}

if (elements['modal-continue-btn']) {

elements['modal-continue-btn'].addEventListener('click', () => {

loadGame();

closeModal(elements['main-menu-modal']);

});

}

if (elements['add-munee-btn']) {

elements['add-munee-btn'].addEventListener('click', () => {

const amount = 10;

gameState.playerMunee += amount;

logMessage(`You added Ξ${amount.toFixed(2)} to your account.`);

updateUI();

});

}

// Tutorial buttons

if (elements['tutorial-next-btn']) {

elements['tutorial-next-btn'].addEventListener('click', () => {

currentTutorialStep++;

if (currentTutorialStep < TUTORIAL\_STEPS.length) {

updateTutorialModal();

} else {

endTutorial();

}

});

}

if (elements['tutorial-back-btn']) {

elements['tutorial-back-btn'].addEventListener('click', () => {

currentTutorialStep--;

updateTutorialModal();

});

}

if (elements['tutorial-close-btn']) {

elements['tutorial-close-btn'].addEventListener('click', () => {

endTutorial();

});

}

// Game control buttons

if (elements['pause-btn']) {

elements['pause-btn'].addEventListener('click', () => setGameSpeed('pause'));

}

if (elements['play-btn']) {

elements['play-btn'].addEventListener('click', () => setGameSpeed('play'));

}

if (elements['ffwd-btn']) {

elements['ffwd-btn'].addEventListener('click', () => setGameSpeed('ffwd'));

}

if (elements['upgrade-btn']) {

elements['upgrade-btn'].addEventListener('click', () => {

openModal(elements['upgrade-modal']);

renderUpgrades();

});

}

if (elements['shop-btn']) {

elements['shop-btn'].addEventListener('click', () => {

openModal(elements['shop-modal']);

showShopTab('unlockables');

});

}

// Save system

if (elements['save-game-btn']) {

elements['save-game-btn'].addEventListener('click', saveGame);

}

if (elements['autosave-checkbox']) {

elements['autosave-checkbox'].addEventListener('change', (e) => {

gameState.isAutosaveEnabled = e.target.checked;

if (gameState.isAutosaveEnabled) {

startAutosaveTimer();

} else {

stopAutosaveTimer();

}

saveGame();

});

}

// Shop tabs

if (elements['shopTabs']) {

elements['shopTabs'].addEventListener('click', (e) => {

if (e.target.tagName === 'BUTTON') {

showShopTab(e.target.textContent.toLowerCase());

}

});

}

// Modal close buttons

elements.modals.forEach(modal => {

const closeBtn = modal.querySelector('.close-btn');

if (closeBtn) {

closeBtn.addEventListener('click', () => closeModal(modal));

}

});

// Bonus modal

if (elements['bonus-modal-close-btn']) {

elements['bonus-modal-close-btn'].addEventListener('click', () => {

closeModal(elements['daily-bonus-modal']);

});

}

// Page visibility handling

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

if (gameState.gameSpeed !== 'pause') {

if (document.hidden) {

pauseOnTabSwitch = true;

setGameSpeed('pause');

logMessage('Game paused automatically.');

} else {

if (pauseOnTabSwitch) {

setGameSpeed('play');

logMessage('Game resumed automatically.');

pauseOnTabSwitch = false;

}

}

}

});

// Touch support for mobile

document.querySelectorAll('button').forEach(button => {

button.addEventListener('touchstart', function(e) {

this.style.transform = 'scale(0.95)';

});

button.addEventListener('touchend', function(e) {

this.style.transform = 'scale(1)';

});

});

}

// Initialize the game

function init() {

initializeElements();

initializeEventListeners();

startLoading();

// Check for saved game

if (localStorage.getItem('globalIdeologySave')) {

if (elements['modal-continue-btn']) {

elements['modal-continue-btn'].disabled = false;

}

}

}

// Public API

return {

init: init,

saveGame: saveGame,

loadGame: loadGame

};

})();

// Start the game when DOM is loaded

document.addEventListener('DOMContentLoaded', function() {

GlobalIdeology.init();

});

// Handle page unload

window.addEventListener('beforeunload', function(e) {

if (gameState && gameState.isAutosaveEnabled) {

GlobalIdeology.saveGame();

}

});