/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* game.js – All JS logic for Fall of Empires

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// ================================

// MAP & TERRITORY RENDERING

// ================================

// FULL territory definitions

const territories = {

"A1": [[21,37], [18,20], [42,5], [46,14]],

"A2": [[22,37], [46,15], [62,19], [51,43], [30,47]],

"A3": [[63,17], [85,30], [67,51], [52,43]],

"A4": [[32,47], [30,54], [37,71], [63,67], [66,53], [50,45]],

"A5": [[30,54], [10,73], [8,102], [38,72]],

"B1": [[60,86], [14,118], [29,141], [53,140], [69,129], [74,120]],

"B2": [[62,86], [76,119], [119,106], [109,69], [84,72]],

"B3": [[110,67], [121,106], [143,102], [125,58]],

"B4": [[123,55], [153,37], [167,61], [158,103], [146,102]],

"B5": [[154,35], [166,60], [187,71], [221,68], [228,18]],

"B6": [[189,72], [222,68], [219,110], [207,116]],

"B7": [[30,143], [43,167], [71,158], [54,141]],

"B8": [[54,139], [69,131], [84,147], [75,158]],

"B9": [[69,130], [77,120], [130,104], [121,146], [85,150]],

"B10": [[133,106], [122,146], [146,155], [185,116], [157,105]],

"B11": [[42,169], [34,202], [62,193], [60,165]],

"B12": [[62,165], [63,191], [82,175]],

"B13": [[62,163], [74,159], [83,150], [121,148], [135,154], [125,181], [99,192], [84,175]],

"B14": [[136,154], [126,181], [158,185], [181,171], [180,155], [163,137], [147,155]],

"B15": [[34,204], [30,222], [46,234], [74,226], [64,195]],

"B16": [[65,191], [82,177], [98,191], [125,182], [138,183], [118,203], [119,229], [75,224]],

"B17": [[138,185], [119,203], [119,230], [138,224], [158,188]],

"B18": [[45,234], [53,241], [46,269], [75,277], [93,227], [75,225]],

"B19": [[95,226], [76,276], [116,269], [130,227], [117,231]],

"C1": [[45,272], [21,290], [34,312], [51,298], [68,300], [66,277]],

"C2": [[68,276], [73,324], [92,322], [100,306], [89,295], [94,276], [77,278]],

"C3": [[92,287], [90,293], [102,304], [97,317], [138,313], [118,281]],

"C4": [[52,298], [34,311], [37,331], [71,325], [68,301]],

"C5": [[71,327], [53,332], [58,346], [77,339]],

"C6": [[73,327], [90,359], [113,358], [139,315], [98,318], [90,323]],

"C7": [[36,333], [26,373], [41,384], [64,365], [84,373], [88,358], [77,341], [59,347], [50,332]],

"C8": [[89,359], [86,374], [141,382], [113,360]],

"D1": [[246,88], [232,100], [245,116], [263,98]],

"D2": [[296,85], [274,123], [278,143], [292,141], [303,108], [326,117], [347,109], [343,69], [321,64], [319,84]],

"D3": [[255,155], [242,164], [234,180], [247,200], [248,215], [259,215]],

"D4": [[257,155], [260,214], [278,202], [303,206], [288,174], [298,162], [267,173]],

"D5": [[300,162], [289,175], [303,207], [322,211], [338,197], [343,172], [306,161]],

"D6": [[302,160], [326,117], [351,157], [345,172]],

"D7": [[327,117], [349,111], [375,117], [383,115], [398,133], [389,175], [352,157]],

"D8": [[340,197], [350,157], [390,174], [380,209], [361,220]],

"D9": [[337,197], [320,211], [319,221], [297,235], [319,237], [324,250], [359,221]],

"D10": [[380,210], [343,236], [346,257], [374,260], [396,254], [411,232], [400,227]],

"D11": [[342,236], [320,257], [334,281], [357,286], [375,275], [375,260], [346,258]],

"D12": [[397,255], [377,261], [376,275], [407,291], [419,273]],

"D13": [[411,231], [397,253], [420,271], [453,256], [447,242], [431,234]],

"D14": [[375,275], [357,287], [365,309], [408,291]],

"D15": [[421,274], [409,290], [426,321], [449,296]],

"D16": [[421,273], [449,294], [469,290], [484,273], [493,242], [472,256], [451,257]],

"E1": [[229,220], [221,262], [249,252], [250,220]],

"E2": [[251,221], [250,251], [272,236]],

"E3": [[221,258], [189,282], [203,297], [225,287], [217,276]],

"E4": [[222,262], [219,274], [226,287], [241,283], [248,287], [257,268], [245,262], [236,257]],

"E5": [[272,240], [250,251], [237,257], [257,266], [282,253]],

"E6": [[242,282], [223,286], [205,297], [231,310], [246,288]],

"E7": [[283,254], [258,266], [262,277], [288,262]],

"E8": [[257,269], [232,311], [242,330], [267,306], [270,292]],

"E9": [[288,263], [264,278], [271,291], [270,306], [275,316], [282,315], [292,306], [300,300], [309,276]],

"E10": [[268,305], [242,332], [244,343], [255,350], [274,317]],

"E11": [[275,317], [257,350], [265,357], [288,340], [282,316]],

"E12": [[293,307], [283,315], [289,339], [300,351], [318,335], [313,321], [304,326]],

"E13": [[294,305], [313,295], [340,304], [357,320], [320,335], [313,320], [304,325]],

"E14": [[266,358], [277,378], [286,379], [299,352], [288,340]],

"F1": [[363,352], [344,370], [389,380], [385,371], [395,355]],

"F2": [[398,355], [387,370], [393,385], [432,395], [447,351], [432,334], [424,356]],

"F3": [[445,362], [434,393], [464,382]],

"F4": [[448,350], [446,359], [466,382], [486,370], [472,343]],

"F5": [[474,341], [514,307], [518,348], [487,370]],

"G1": [[369,57], [344,69], [349,108], [374,116], [391,111]],

"G2": [[401,22], [369,55], [381,84], [407,80], [426,44]],

"G3": [[384,115], [397,133], [424,138], [439,115], [443,86], [422,93], [407,81], [381,85], [391,111]],

"G4": [[421,58], [408,79], [421,92], [443,85]],

"G5": [[467,12], [445,27], [428,43], [421,57], [445,86], [495,32]],

"G6": [[401,21], [416,7], [443,26], [427,41]],

"G7": [[497,30], [445,84], [463,93], [475,85], [513,85], [539,56]],

"G8": [[397,135], [390,174], [412,183], [435,186], [444,180], [424,138]],

"G9": [[424,136], [440,117], [472,139], [443,178]],

"G10": [[445,83], [440,115], [473,139], [488,128], [469,92], [462,96]],

"G11": [[527,129], [531,93], [514,87], [477,85], [469,91], [489,128]],

"G12": [[389,176], [381,209], [399,227], [413,206], [411,183]],

"G13": [[412,184], [434,187], [438,200], [427,205], [439,217], [433,234], [402,227], [414,206]],

"H1": [[471,162], [454,178], [457,201], [468,203], [467,188], [478,181], [480,166]],

"H2": [[481,165], [479,180], [469,188], [470,202], [493,193], [486,166]],

"H3": [[486,164], [500,155], [515,170], [512,184], [496,193]],

"H4": [[520,144], [503,155], [517,168], [512,184], [531,184], [539,163]],

"H5": [[457,203], [447,227], [462,226], [468,205]],

"H6": [[469,206], [464,228], [474,241], [488,231], [483,222], [493,196]],

"H7": [[531,185], [512,185], [495,195], [489,212], [512,214], [528,205]],

"H8": [[485,221], [489,213], [510,215], [515,230], [512,237], [488,232]]

};

// Neighbor mapping

const neighborMapping = {

"A1": ["A2"],

"A2": ["A1", "A3", "A4"],

"A3": ["A2", "A4", "B3"],

"A4": ["A2", "A3", "A5"],

"A5": ["A4", "B1"],

"B1": ["B2", "B7", "A5", "B8", "B9"],

"B2": ["B1", "B3", "B9"],

"B3": ["B2", "B4", "B9", "B10"],

"B4": ["B3", "B5", "B10"],

"B5": ["B4", "B6"],

"B6": ["B5", "D1"],

"B7": ["B1", "B8", "B11", "B12", "B13"],

"B8": ["B7", "B9", "B1", "B13"],

"B9": ["B8", "B10", "B13", "B1", "B2", "B3"],

"B10": ["B4", "B3", "B9", "B13", "B14"],

"B11": ["B7", "B12", "B15"],

"B12": ["B11", "B13", "B15", "B16"],

"B13": ["B8", "B9", "B10", "B12", "B14", "B16"],

"B14": ["B10", "B13", "B16", "B17"],

"B15": ["B11", "B16", "B18"],

"B16": ["B12", "B13", "B15", "B14", "B17", "B18", "B19"],

"B17": ["B14", "B16", "B19"],

"B18": ["B15", "B16", "C1", "B19", "C2"],

"B19": ["B17", "B18", "B16"],

"C1": ["C2", "C4", "B18"],

"C2": ["C1", "C3", "C4", "C5", "B19", "B18", "C6"],

"C3": ["C2", "C6", "E3"],

"C4": ["C1", "C2", "C5", "C7"],

"C5": ["C2", "C4", "C6", "C7"],

"C6": ["C3", "C5", "C8", "C2", "C7"],

"C7": ["C4", "C8", "C5", "C6"],

"C8": ["C6", "C7"],

"D1": ["D2", "B6"],

"D2": ["D1", "G1", "D7", "D6"],

"D3": ["D4", "E1"],

"D4": ["D3", "D5"],

"D5": ["D4", "D6", "D8", "D9"],

"D6": ["D5", "D7", "D8", "D2"],

"D7": ["D2", "D6", "D8", "G1", "G3", "G8"],

"D8": ["D7", "D9", "D10", "D5", "D6", "G12", "G8"],

"D9": ["D8", "D10", "D11", "D5"],

"D10": ["D8", "D9", "D11", "D12", "G12", "D13", "G13"],

"D11": ["D9", "D10", "D12", "D14"],

"D12": ["D10", "D13", "D14", "D11", "D15"],

"D13": ["G13", "D12", "D10", "D16"],

"D14": ["D12", "D15", "D11", "E13"],

"D15": ["D12", "D14", "D16", "F2"],

"D16": ["D13", "D15", "H8", "F5"],

"E1": ["E2", "E3", "E4", "E5", "D3"],

"E2": ["E1", "E5"],

"E3": ["E1", "E6", "E4"],

"E4": ["E3", "E5", "E6", "E1", "E8"],

"E5": ["E1", "E2", "E4", "E7"],

"E6": ["E3", "E4", "E8"],

"E7": ["E5", "E8", "E9"],

"E8": ["E6", "E7", "E9", "E10", "E4"],

"E9": ["E7", "E8", "E10", "E11", "E12", "E13"],

"E10": ["E8", "E9", "E11"],

"E11": ["E9", "E10", "E12", "E14"],

"E12": ["E9", "E11", "E13", "E14"],

"E13": ["F1", "E12", "E9"],

"E14": ["E12", "E11"],

"F1": ["F2", "E13"],

"F2": ["F1", "F3", "F4", "D15"],

"F3": ["F2", "F4"],

"F4": ["F2", "F3", "F5"],

"F5": ["D16", "F4"],

"G1": ["G2", "G3", "D2", "D7"],

"G2": ["G1", "G3", "G5", "G6", "G4"],

"G3": ["G1", "G2", "G4", "G10", "G8", "G9", "D7"],

"G4": ["G2", "G3", "G5"],

"G5": ["G2", "G4", "G6", "G7"],

"G6": ["G2", "G5"],

"G7": ["G5", "G10", "G11"],

"G8": ["G3", "G9", "D7", "G12", "G13", "D8"],

"G9": ["G8", "G10", "G3"],

"G10": ["G7", "G3", "G9", "G11"],

"G11": ["G7", "G10", "H4"],

"G12": ["G8", "G13", "D8", "D10"],

"G13": ["G8", "G12", "D10", "D13"],

"H1": ["H2", "H5"],

"H2": ["H1", "H3", "H6"],

"H3": ["H2", "H4", "H7"],

"H4": ["H3", "H7", "G11"],

"H5": ["H1", "H6"],

"H6": ["H5", "H7", "H8", "H2"],

"H7": ["H3", "H4", "H6", "H8"],

"H8": ["H6", "H7", "D16"]

};

// Continent colors

const continentColors = {

'A': '#ffcccc',

'B': '#ccffcc',

'C': '#ccccff',

'D': '#ffcc99',

'E': '#ccffff',

'F': '#ffccff',

'G': '#ffffcc',

'H': '#d9d9d9'

};

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

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

// Compute scale

let allX = [], allY = [];

for (const key in territories) {

territories[key].forEach(pt => {

allX.push(pt[0]);

allY.push(pt[1]);

});

}

const minX = Math.min(...allX);

const maxX = Math.max(...allX);

const minY = Math.min(...allY);

const maxY = Math.max(...allY);

const padding = 20;

const scaleX = (canvas.width - 2 \* padding) / (maxX - minX);

const scaleY = (canvas.height - 2 \* padding) / (maxY - minY);

const scale = Math.min(scaleX, scaleY);

// Build territoryList

const territoryList = [];

for (const name in territories) {

const coords = territories[name];

const path = new Path2D();

coords.forEach((pt, i) => {

const x = padding + (pt[0] - minX) \* scale;

const y = padding + (pt[1] - minY) \* scale;

if (i === 0) path.moveTo(x, y);

else path.lineTo(x, y);

});

path.closePath();

let sumX = 0, sumY = 0;

coords.forEach(pt => {

sumX += pt[0];

sumY += pt[1];

});

const centroidX = padding + ((sumX / coords.length) - minX) \* scale;

const centroidY = padding + ((sumY / coords.length) - minY) \* scale;

territoryList.push({

name,

path,

centroidX,

centroidY,

group: name.charAt(0),

owner: "",

oil: 3 \* 100, // 300

votes: 1

});

}

const territoryMap = {};

territoryList.forEach(t => { territoryMap[t.name] = t; });

let selectedTerritoryName = null;

let hoveredTerritoryName = null;

// Draw neighbor lines

function drawNeighborLines() {

const drawnPairs = new Set();

territoryList.forEach(territory => {

const neighbors = neighborMapping[territory.name] || [];

neighbors.forEach(n => {

const neighbor = territoryMap[n];

if (!neighbor) return;

const pairKey = [territory.name, n].sort().join('-');

if (!drawnPairs.has(pairKey)) {

drawnPairs.add(pairKey);

ctx.save();

ctx.beginPath();

ctx.setLineDash([5, 5]);

ctx.strokeStyle = "#0000FF";

ctx.lineWidth = 1;

ctx.moveTo(territory.centroidX, territory.centroidY);

ctx.lineTo(neighbor.centroidX, neighbor.centroidY);

ctx.stroke();

ctx.restore();

}

});

});

}

// Only show neighbors in base-selection phase

function redrawMap() {

ctx.fillStyle = "black";

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

// Draw neighbor lines ONLY if isSetupPhase == true

if (isSetupPhase) {

drawNeighborLines();

}

territoryList.forEach(terr => {

// Fill

ctx.save();

if (terr.owner !== "") {

ctx.fillStyle = players[terr.owner].color;

ctx.globalAlpha = 0.6;

} else {

ctx.fillStyle = continentColors[terr.group] || "#fff";

ctx.globalAlpha = 0.3;

}

ctx.fill(terr.path);

ctx.restore();

// Outline

ctx.save();

ctx.shadowColor = "#0000FF";

ctx.shadowBlur = 10;

ctx.lineWidth = 1;

ctx.strokeStyle = "#0000FF";

ctx.stroke(terr.path);

ctx.restore();

// White border if it's the player's base

if (players[terr.owner] && players[terr.owner].baseTerritory === terr.name) {

ctx.save();

ctx.lineWidth = 3;

ctx.strokeStyle = "white";

ctx.stroke(terr.path);

ctx.restore();

}

// Label

ctx.fillStyle = "white";

ctx.font = "10px Arial";

ctx.textAlign = "center";

ctx.textBaseline = "middle";

ctx.fillText(terr.name, terr.centroidX, terr.centroidY);

});

// If selected or hovered

if (selectedTerritoryName) {

highlightTerritory(selectedTerritoryName, false);

}

if (hoveredTerritoryName && hoveredTerritoryName !== selectedTerritoryName) {

highlightTerritory(hoveredTerritoryName, true);

}

}

function highlightTerritory(name, dashed) {

const terr = territoryMap[name];

if (!terr) return;

ctx.save();

ctx.setLineDash(dashed ? [5, 5] : []);

ctx.lineWidth = 4;

ctx.shadowBlur = 10;

ctx.shadowColor = "#0000FF";

ctx.strokeStyle = "#0000FF";

ctx.stroke(terr.path);

ctx.restore();

}

// Tooltip

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

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

const rect = canvas.getBoundingClientRect();

const mouseX = e.clientX - rect.left;

const mouseY = e.clientY - rect.top;

hoveredTerritoryName = null;

for (const t of territoryList) {

if (ctx.isPointInPath(t.path, mouseX, mouseY)) {

hoveredTerritoryName = t.name;

break;

}

}

redrawMap();

if (hoveredTerritoryName) {

let terr = territoryMap[hoveredTerritoryName];

let ownerText = (terr.owner === "")

? "None"

: (players[terr.owner].name + " (Str: " + players[terr.owner].baseStr + ")");

tooltip.innerHTML = `<strong>${terr.name}</strong><br>

Oil: ${terr.oil}<br>

Votes: ${terr.votes}<br>

Owner: ${ownerText}`;

tooltip.style.left = (e.clientX + 5) + "px";

tooltip.style.top = (e.clientY + 5) + "px";

tooltip.style.display = "block";

} else {

tooltip.style.display = "none";

}

});

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

const rect = canvas.getBoundingClientRect();

const mouseX = e.clientX - rect.left;

const mouseY = e.clientY - rect.top;

let found = null;

for (const t of territoryList) {

if (ctx.isPointInPath(t.path, mouseX, mouseY)) {

found = t.name;

break;

}

}

if (isSetupPhase) {

if (found) {

const terr = territoryMap[found];

if (terr.owner !== "") {

alert("That territory is already taken. Choose another.");

} else {

selectedTerritoryName = found;

redrawMap();

}

}

} else {

selectedTerritoryName = (found && selectedTerritoryName !== found) ? found : null;

redrawMap();

}

});

// ================================

// GAME LOGIC

// ================================

let players = [];

let currentPlayerIndex = 0;

let isSetupPhase = true;

let turnsThisQuarter = 0;

// Using lower ROI: start at 20%

let currentROI = 0.2;

let currentOilPrice = 600; // start = 600

let messages = [];

// Return territory object if we have a selected name

function getSelectedTerritory() {

if (!selectedTerritoryName) return null;

return territoryMap[selectedTerritoryName];

}

// Adjacency check for expansions

function canExpandTo(territoryName, playerIndex) {

// Must be neighbor of some territory owned by playerIndex

let owned = territoryList.filter(t => t.owner === playerIndex);

for (let o of owned) {

if ((neighborMapping[o.name] || []).includes(territoryName)) {

return true;

}

}

return false;

}

// Start the game

function startGame() {

const chosenColor = document.getElementById('playerColor').value;

let numOpp = parseInt(document.getElementById('numOpponents').value, 10);

if (isNaN(numOpp) || numOpp < 0) numOpp = 0;

if (numOpp > 5) numOpp = 5;

players = [];

// Human

players.push({

name: "Player 1",

ap: 0, unity: 0, oil: 0, gold: 1200,

wisdom: 0, intel: 0, votes: 0,

baseStr: 15,

roi: 0.2,

baseTier: 1,

isAlive: true, color: chosenColor,

isAI: false, baseTerritory: ""

});

// AIs

const allColors = ["blue", "green", "red", "yellow", "purple", "orange"];

const remainColors = allColors.filter(c => c !== chosenColor);

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

players.push({

name: "AI " + (i + 1),

ap: 0, unity: 0, oil: 0, gold: 1200,

wisdom: 0, intel: 0, votes: 0,

baseStr: 15,

roi: 0.2,

baseTier: 1,

isAlive: true, color: remainColors[i % remainColors.length],

isAI: true, baseTerritory: ""

});

}

document.getElementById('gameSetupOverlay').style.display = "none";

refreshUI();

redrawMap();

if (players[currentPlayerIndex].isAI) {

setTimeout(aiTakeTurn, 1000);

}

}

// Confirm base selection

function confirmBaseSelection() {

let p = players[currentPlayerIndex];

let terr = getSelectedTerritory();

if (!terr) {

alert("Select a territory as your base first.");

return;

}

if (terr.owner !== "") {

alert("That territory is taken.");

return;

}

terr.owner = currentPlayerIndex;

p.baseTerritory = terr.name;

logMessage(`${p.name} selects ${terr.name} as base.`);

redrawMap();

selectedTerritoryName = null;

// If it's the human picking first, assign AI bases automatically

if (currentPlayerIndex === 0 && players.length > 1) {

let available = territoryList.filter(t => t.owner === "");

for (let i = 1; i < players.length; i++) {

if (available.length === 0) break;

let r = Math.floor(Math.random() \* available.length);

let aiTerr = available[r];

aiTerr.owner = i;

players[i].baseTerritory = aiTerr.name;

available.splice(r, 1);

logMessage(`${players[i].name} selects ${aiTerr.name} as base (auto).`);

}

isSetupPhase = false;

logMessage("Setup complete. Starting normal play.");

document.getElementById("setupPanel").style.display = "none";

currentPlayerIndex = 0;

refreshUI();

startTurn();

} else {

// If multi-human or so

currentPlayerIndex++;

if (currentPlayerIndex < players.length) {

logMessage(`Now it's ${players[currentPlayerIndex].name}'s turn to pick a base.`);

refreshUI();

} else {

isSetupPhase = false;

logMessage("Setup complete. Starting normal play.");

document.getElementById("setupPanel").style.display = "none";

currentPlayerIndex = 0;

refreshUI();

startTurn();

}

}

}

// Start Turn

function startTurn() {

let p = players[currentPlayerIndex];

p.ap = 5;

p.gold += 1200;

logMessage(`${p.name} starts turn with +5 AP, +$1200.`);

selectedTerritoryName = null;

redrawMap();

refreshUI();

if (p.isAI) {

setTimeout(aiTakeTurn, 1200);

}

}

// End Turn

function endTurn() {

currentPlayerIndex = (currentPlayerIndex + 1) % players.length;

turnsThisQuarter++;

if (turnsThisQuarter >= players.length) {

endQuarter();

turnsThisQuarter = 0;

}

logMessage(`End turn. Next: ${players[currentPlayerIndex].name}`);

startTurn();

}

// End Quarter

function endQuarter() {

// Everyone collects ROI

for (let pl of players) {

if (!pl.isAlive) continue;

let roiGain = Math.floor(pl.gold \* pl.roi);

pl.gold += roiGain;

logMessage(`${pl.name} collects ROI: $${roiGain} at ${Math.round(pl.roi\*100)}%.`);

}

// Reroll ROI: 1/3 each 0.1, 0.2, 0.3

let r1 = Math.floor(Math.random() \* 3);

currentROI = (r1===0) ? 0.1 : (r1===1)?0.2:0.3;

// Reroll Oil: 1/3 each 300,600,900

let r2 = Math.floor(Math.random() \* 3);

currentOilPrice = (r2===0)?300 : (r2===1)?600 : 900;

// Reset each player's ROI

for (let pl of players) {

if (pl.isAlive) {

pl.roi = currentROI;

}

}

logMessage(`New Quarter: ROI=${Math.round(currentROI\*100)}%, Oil=$${currentOilPrice}`);

refreshUI();

}

// AI logic

function aiTakeTurn() {

let ai = players[currentPlayerIndex];

// Attack/Expand if can

if (ai.ap >= 2 && ai.gold >= 900) {

// pick neighbor territory with best oil

let owned = territoryList.filter(t => t.owner === currentPlayerIndex);

let cands = [];

owned.forEach(o => {

let neighs = neighborMapping[o.name] || [];

neighs.forEach(nm => {

let c = territoryMap[nm];

if (c && c.owner !== currentPlayerIndex) {

cands.push(c);

}

});

});

cands = [...new Set(cands)];

if (cands.length > 0) {

cands.sort((a,b) => b.oil - a.oil);

selectedTerritoryName = cands[0].name;

actionAttackExpand(true);

}

}

setTimeout(endTurn, 1200);

}

// ================================

// ACTIONS

// ================================

function actionAttackExpand(isAI=false) {

let p = players[currentPlayerIndex];

if (p.ap < 2) {

logMessage(`${p.name} lacks 2 AP to Attack/Expand.`);

return;

}

if (p.gold < 900) {

logMessage(`${p.name} lacks $900 to Attack/Expand.`);

return;

}

let terr = getSelectedTerritory();

if (!terr) {

logMessage("No territory selected.");

return;

}

if (!isAI && !canExpandTo(terr.name, currentPlayerIndex)) {

logMessage("You can only expand/attack adjacent territory you own!");

return;

}

if (terr.owner === currentPlayerIndex) {

logMessage("You already own that territory.");

return;

}

// Pay cost

p.ap -= 2;

p.gold -= 900;

// If unowned => free

if (terr.owner === "") {

p.oil += terr.oil;

p.votes += terr.votes;

terr.owner = currentPlayerIndex;

logMessage(`${p.name} expands into ${terr.name}, +${terr.oil} Oil, +${terr.votes} Votes.`);

} else {

// Attack

let atk = rollDice(3);

let def = rollDice(3);

logMessage(`${p.name} attacks ${terr.name} (atk=${atk}, def=${def}).`);

if (atk > def) {

logMessage("Success! Territory captured.");

terr.owner = currentPlayerIndex;

} else {

logMessage("Failed to capture.");

}

}

redrawMap();

refreshUI();

}

function actionSpyDef() {

let p = players[currentPlayerIndex];

if (p.ap < 2) {

logMessage(`${p.name} lacks 2 AP for Spy/Def.`);

return;

}

if (p.intel < 1) {

logMessage(`${p.name} needs 1 Intel for Spy/Def.`);

return;

}

p.ap -= 2;

p.intel--;

// pick random other player

let others = players.map((pl,i) => i).filter(i => i!==currentPlayerIndex && players[i].isAlive);

if (others.length === 0) {

logMessage("No players to spy on!");

return;

}

let targetIndex = others[Math.floor(Math.random()\*others.length)];

let tp = players[targetIndex];

let spyDiv = document.getElementById("spyInfo");

spyDiv.style.backgroundColor = tp.color;

spyDiv.innerHTML = `

<strong>Spied on: ${tp.name}</strong><br>

BaseStr: ${tp.baseStr}, Tier: ${tp.baseTier}<br>

Gold: ${tp.gold}, Oil: ${tp.oil}, AP: ${tp.ap}<br>

Unity: ${tp.unity}, Wisdom: ${tp.wisdom}, Intel: ${tp.intel}<br>

ROI: ${Math.round(tp.roi \* 100)}%

`;

logMessage(`${p.name} spies on ${tp.name}; stats revealed.`);

refreshUI();

}

function upgradeBase() {

let p = players[currentPlayerIndex];

const cost = p.baseTier \* 300;

if (p.gold < cost) {

logMessage(`${p.name} needs $${cost} to upgrade base.`);

return;

}

if (p.unity < 1) {

logMessage(`${p.name} needs 1 Unity to upgrade base.`);

return;

}

p.gold -= cost;

p.unity--;

p.baseTier++;

// +15 baseStr, cap at 65

p.baseStr = Math.min(p.baseStr + 15, 65);

logMessage(`${p.name} upgrades base to Tier ${p.baseTier}, baseStr=${p.baseStr}.`);

refreshUI();

}

function actionBuyOil() {

let p = players[currentPlayerIndex];

if (p.gold < currentOilPrice) {

logMessage(`${p.name} can't afford $${currentOilPrice} to buy 300 Oil.`);

return;

}

p.gold -= currentOilPrice;

p.oil += 300;

logMessage(`${p.name} buys 300 Oil for $${currentOilPrice}.`);

refreshUI();

}

function actionSellOil() {

let p = players[currentPlayerIndex];

if (p.oil < 300) {

logMessage(`${p.name} doesn't have 300 Oil to sell.`);

return;

}

p.oil -= 300;

p.gold += currentOilPrice;

logMessage(`${p.name} sells 300 Oil for $${currentOilPrice}.`);

refreshUI();

}

function actionBuyIntel() {

let p = players[currentPlayerIndex];

if (p.ap < 1) {

logMessage(`${p.name} lacks 1 AP to buy Intel.`);

return;

}

if (p.oil < 900) {

logMessage(`${p.name} needs 900 Oil to buy Intel.`);

return;

}

p.ap--;

p.oil -= 900;

p.intel++;

logMessage(`${p.name} bought +1 Intel.`);

refreshUI();

}

function actionGetWise() {

let p = players[currentPlayerIndex];

if (p.ap < 1) {

logMessage(`${p.name} needs 1 AP to get wise.`);

return;

}

if (p.gold < 900) {

logMessage(`${p.name} needs $900 to get wise.`);

return;

}

p.ap--;

p.gold -= 900;

p.wisdom++;

logMessage(`${p.name} gained +1 Wisdom.`);

refreshUI();

}

function actionCreateUnity() {

let p = players[currentPlayerIndex];

if (p.wisdom < 1 || p.intel < 1) {

logMessage(`${p.name} needs 1 Wisdom & 1 Intel to create Unity.`);

return;

}

p.wisdom--;

p.intel--;

p.unity++;

logMessage(`${p.name} created +1 Unity.`);

refreshUI();

}

// ROI invests +0.2 => +20% instead of +200%

function actionInvestROI() {

let p = players[currentPlayerIndex];

if (p.wisdom < 5) {

logMessage(`${p.name} needs 5 Wisdom to invest ROI.`);

return;

}

p.wisdom -= 5;

p.roi += 0.2; // big jump

logMessage(`${p.name} invests in ROI, now=${Math.round(p.roi\*100)}%.`);

refreshUI();

}

// Utility

function rollDice(numDice, bonus=0){

let total = 0;

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

total += Math.floor(Math.random()\*6)+1;

}

return total + bonus;

}

// Invert the scroll in the log – show newest on top

function logMessage(msg) {

// Insert the newest message at the front

messages.unshift(msg);

// Show the latest 3 messages, newest first

const recent = messages.slice(0, 3);

document.getElementById('log').innerHTML = recent.map(m => "> " + m).join("<br>");

console.log(msg);

}

// Dynamically highlight actions

function refreshUI() {

const p = players[currentPlayerIndex];

// Update stat displays

document.getElementById('apDisplay').innerText = p.ap;

document.getElementById('unityDisplay').innerText = p.unity;

document.getElementById('oilDisplay').innerText = p.oil;

document.getElementById('goldDisplay').innerText = p.gold;

document.getElementById('wisdomDisplay').innerText = p.wisdom;

document.getElementById('intelDisplay').innerText = p.intel;

document.getElementById('votesDisplay').innerText = p.votes;

document.getElementById('baseStrDisplay').innerText = p.baseStr;

document.getElementById('roiDisplay').innerText = "+" + Math.round(p.roi \* 100) + "%";

document.getElementById('tierDisplay').innerText = p.baseTier;

redrawMap();

highlightAvailableActions();

}

function highlightAvailableActions() {

// remove highlight from all

const btnIDs = [

'attackExpandBtn','spyDefBtn','upgradeBaseBtn','endTurnBtn',

'buyOilBtn','sellOilBtn','buyIntelBtn','getWiseBtn','createUnityBtn','investRoiBtn'

];

btnIDs.forEach(id => {

const b = document.getElementById(id);

if (b) b.classList.remove("highlighted");

});

const p = players[currentPlayerIndex];

// Attack/Expand => AP>=2 && gold>=900

if (p.ap>=2 && p.gold>=900) {

document.getElementById('attackExpandBtn').classList.add("highlighted");

}

// Spy/Def => ap>=2 && intel>=1

if (p.ap>=2 && p.intel>=1) {

document.getElementById('spyDefBtn').classList.add("highlighted");

}

// Upgrade base => gold>= tier×300 && unity>=1 && baseStr<65

const upgCost = p.baseTier\*300;

if (p.gold>=upgCost && p.unity>=1 && p.baseStr<65) {

document.getElementById('upgradeBaseBtn').classList.add("highlighted");

}

// BuyOil => p.gold>= currentOilPrice

if (p.gold>=currentOilPrice) {

document.getElementById('buyOilBtn').classList.add("highlighted");

}

// SellOil => p.oil>=300

if (p.oil>=300) {

document.getElementById('sellOilBtn').classList.add("highlighted");

}

// BuyIntel => p.ap>=1 && p.oil>=900

if (p.ap>=1 && p.oil>=900) {

document.getElementById('buyIntelBtn').classList.add("highlighted");

}

// GetWise => p.ap>=1 && p.gold>=900

if (p.ap>=1 && p.gold>=900) {

document.getElementById('getWiseBtn').classList.add("highlighted");

}

// CreateUnity => p.wisdom>=1 && p.intel>=1

if (p.wisdom>=1 && p.intel>=1) {

document.getElementById('createUnityBtn').classList.add("highlighted");

}

// InvestROI => p.wisdom>=5

if (p.wisdom>=5) {

document.getElementById('investRoiBtn').classList.add("highlighted");

}

// End turn => always highlight

document.getElementById('endTurnBtn').classList.add("highlighted");

}

// onload equivalent

window.onload = function() {

refreshUI();

redrawMap();

};