Q1

#include <iostream>

#include <string>

#include <limits>

#include <ctime>

#include <cstdlib>

using namespace std;

// ========== ANSI COLOR CODES ==========

const string RESET = "\033[0m";

const string RED = "\033[31m";

const string GREEN = "\033[32m";

const string YELLOW = "\033[33m";

const string BLUE = "\033[34m";

const string MAGENTA = "\033[35m";

const string CYAN = "\033[36m";

const string BOLD = "\033[1m";

const string UNDERLINE = "\033[4m";

// ========== GAME CONSTANTS ==========

const int MAX\_KINGDOMS = 2;

const int MAP\_SIZE = 5;

const int MAX\_ALLIANCES = 10;

// ========== KINGDOM CLASS ==========

class Kingdom {

public:

string name;

int armyStrength;

int resources;

int x, y;

string color;

Kingdom(string n = "", int a = 0, int r = 0, int posX = 0, int posY = 0, string col = RESET)

: name(n), armyStrength(a), resources(r), x(posX), y(posY), color(col) {

}

void displayStatus() const {

cout << color << BOLD << "\n=================\n";

cout << "||" << name << " Kingdom Status ||\n";

cout << "||====================||\n";

cout << "|| Army: " << armyStrength << " soldiers ||\n";

cout << "|| Gold: " << resources << " coins ||\n";

cout << "|| Position: (" << x << "," << y << ") ||\n";

cout << "||==================||" << RESET << "\n";

}

void recruitSoldiers(int amount) {

int cost = amount \* 2;

if (resources >= cost) {

armyStrength += amount;

resources -= cost;

cout << GREEN << "\n✔ Recruited " << amount << " soldiers for " << cost << " gold!" << RESET << "\n";

}

else {

cout << RED << "\n✘ Not enough gold to recruit soldiers!" << RESET << "\n";

}

}

};

// ========== MESSAGE CLASS ==========

class Message {

public:

string sender, receiver, content;

Message(string s = "", string r = "", string c = "")

: sender(s), receiver(r), content(c) {

}

void display() const {

cout << CYAN << "\n📜 " << sender << " sends to " << receiver << ":\n";

cout << "« " << content << " »" << RESET << "\n";

}

};

// ========== ALLIANCE MANAGER ==========

class AllianceManager {

struct Alliance {

string kingdom1;

string kingdom2;

} alliances[MAX\_ALLIANCES];

int allianceCount = 0;

public:

void formAlliance(const string& a, const string& b) {

if (allianceCount < MAX\_ALLIANCES) {

alliances[allianceCount].kingdom1 = a;

alliances[allianceCount].kingdom2 = b;

allianceCount++;

cout << GREEN << "\n✨ " << a << " and " << b << " have formed an alliance! ✨" << RESET << "\n";

}

else {

cout << RED << "\nMaximum alliances reached!" << RESET << "\n";

}

}

void breakAlliance(const string& a, const string& b) {

for (int i = 0; i < allianceCount; i++) {

if ((alliances[i].kingdom1 == a && alliances[i].kingdom2 == b) ||

(alliances[i].kingdom1 == b && alliances[i].kingdom2 == a)) {

// Shift remaining alliances down

for (int j = i; j < allianceCount - 1; j++) {

alliances[j] = alliances[j + 1];

}

allianceCount--;

cout << RED << "\n💔 " << a << " and " << b << " have broken their alliance!" << RESET << "\n";

return;

}

}

cout << YELLOW << "\nNo alliance exists between " << a << " and " << b << "." << RESET << "\n";

}

bool checkAlliance(const string& a, const string& b) const {

for (int i = 0; i < allianceCount; i++) {

if ((alliances[i].kingdom1 == a && alliances[i].kingdom2 == b) ||

(alliances[i].kingdom1 == b && alliances[i].kingdom2 == a)) {

return true;

}

}

return false;

}

};

// ========== MARKET CLASS ==========

class Market {

public:

void trade(Kingdom& from, Kingdom& to, int amount) {

if (from.resources >= amount) {

from.resources -= amount;

to.resources += amount;

cout << GREEN << "\n💰 Trade successful! " << from.name

<< " sent " << amount << " gold to " << to.name << RESET << "\n";

}

else {

cout << RED << "\n❌ " << from.name << " doesn't have enough gold!" << RESET << "\n";

}

}

void smuggle(Kingdom& from, Kingdom& to, int amount) {

if (from.resources >= amount) {

// 20% chance of getting caught

if (rand() % 5 == 0) {

cout << RED << "\n🛑 Smuggling failed! " << from.name

<< " was caught and fined " << amount / 2 << " gold!" << RESET << "\n";

from.resources -= amount / 2;

}

else {

from.resources -= amount;

to.resources += amount;

cout << MAGENTA << "\n🎭 Smuggle successful! " << amount

<< " gold delivered to " << to.name << RESET << "\n";

}

}

else {

cout << RED << "\n❌ " << from.name << " doesn't have enough gold!" << RESET << "\n";

}

}

};

// ========== CONFLICT MANAGER ==========

class ConflictManager {

public:

void attack(Kingdom& attacker, Kingdom& defender, AllianceManager& allianceManager) {

if (allianceManager.checkAlliance(attacker.name, defender.name)) {

cout << YELLOW << "\n🕊 Allies cannot attack each other!" << RESET << "\n";

return;

}

cout << RED << "\n⚔ " << attacker.name << " attacks " << defender.name << "! ⚔" << RESET << "\n";

// Calculate battle outcome with some randomness

int attackerPower = attacker.armyStrength + (rand() % 20);

int defenderPower = defender.armyStrength + (rand() % 20);

if (attackerPower > defenderPower) {

int loot = defender.resources / 3;

attacker.resources += loot;

defender.resources -= loot;

cout << RED << "🏆 " << attacker.name << " wins! Looted "

<< loot << " gold!" << RESET << "\n";

}

else if (attackerPower < defenderPower) {

int counterLoot = attacker.resources / 4;

defender.resources += counterLoot;

attacker.resources -= counterLoot;

cout << GREEN << "🛡 " << defender.name << " defends successfully! Captured "

<< counterLoot << " gold!" << RESET << "\n";

}

else {

cout << YELLOW << "🤝 Battle ends in stalemate!" << RESET << "\n";

}

// Army losses

attacker.armyStrength -= attacker.armyStrength / 10;

defender.armyStrength -= defender.armyStrength / 10;

}

};

// ========== GAME MAP ==========

class GameMap {

string grid[MAP\_SIZE][MAP\_SIZE];

public:

GameMap() {

// Initialize empty map

for (int y = 0; y < MAP\_SIZE; y++) {

for (int x = 0; x < MAP\_SIZE; x++) {

grid[y][x] = "·";

}

}

}

void placeKingdom(const Kingdom& k) {

grid[k.y][k.x] = k.color + BOLD + k.name.substr(0, 1) + RESET;

}

void moveKingdom(Kingdom& k, int newX, int newY) {

if (newX >= 0 && newX < MAP\_SIZE && newY >= 0 && newY < MAP\_SIZE) {

grid[k.y][k.x] = "·";

k.x = newX;

k.y = newY;

grid[k.y][k.x] = k.color + BOLD + k.name.substr(0, 1) + RESET;

cout << GREEN << "\n🗺 " << k.name << " moved to (" << newX << ", " << newY << ")" << RESET << "\n";

}

else {

cout << RED << "\n❌ Invalid coordinates!" << RESET << "\n";

}

}

void display() const {

cout << BLUE << "\n K I N G D O M M A P\n";

cout << " ";

for (int x = 0; x < MAP\_SIZE; x++) cout << x << " ";

cout << "\n";

for (int y = 0; y < MAP\_SIZE; y++) {

cout << y << " ";

for (int x = 0; x < MAP\_SIZE; x++) {

cout << grid[y][x] << " ";

}

cout << "\n";

}

cout << "\n" << UNDERLINE << "Legend:" << RESET << "\n";

cout << "· - Empty territory\n";

cout << BOLD << "A" << RESET << " - Alpha Kingdom\n";

cout << BOLD << "B" << RESET << " - Beta Kingdom\n";

}

};

// ========== UTILITY FUNCTIONS ==========

void displayHeader() {

cout << BOLD << BLUE << R"(

\_\_\_\_\_ \_ \_ \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_

/ \_\_\_| \ | | | \_\_ \ / \_\_\_\_| \_| \_\_ \

| | \_| \| | | | | | | || || | | | | |\_) |

| | |\_ | . ` | | | | | | || || | | | | \_\_\_/

| || | |\ || || || || || |\_\_\_\_ | || |

\|| \||/\||\_|

)" << RESET << "\n";

}

void clearScreen() {

cout << "\033[2J\033[1;1H";

}

int getValidInput(int min, int max) {

int choice;

while (true) {

cout << "Choice [" << min << "-" << max << "]: ";

cin >> choice;

if (cin.fail() || choice < min || choice > max) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << RED << "Invalid input!" << RESET << "\n";

}

else {

cin.ignore();

return choice;

}

}

}

// ========== GAME MENUS ==========

void mainMenu() {

cout << "\n" << UNDERLINE << "Main Menu" << RESET << "\n";

cout << "1. Kingdom Management\n";

cout << "2. Diplomacy\n";

cout << "3. Military Actions\n";

cout << "4. View Map\n";

cout << "5. End Turn\n";

cout << "6. Exit Game\n";

}

void kingdomMenu() {

cout << "\n" << UNDERLINE << "Kingdom Management" << RESET << "\n";

cout << "1. Recruit Soldiers\n";

cout << "2. Send Message\n";

cout << "3. Trade Resources\n";

cout << "4. Smuggle Resources\n";

cout << "5. Back to Main\n";

}

void diplomacyMenu() {

cout << "\n" << UNDERLINE << "Diplomacy" << RESET << "\n";

cout << "1. Form Alliance\n";

cout << "2. Break Alliance\n";

cout << "3. Back to Main\n";

}

void militaryMenu() {

cout << "\n" << UNDERLINE << "Military Actions" << RESET << "\n";

cout << "1. Launch Attack\n";

cout << "2. Move Forces\n";

cout << "3. Back to Main\n";

}

// ========== MAIN GAME LOOP ==========

int main() {

srand(time(0)); // Seed random number generator

// Initialize kingdoms

Kingdom kingdoms[MAX\_KINGDOMS] = {

Kingdom("Alpha", 100, 500, 0, 0, RED),

Kingdom("Beta", 80, 400, 4, 4, BLUE)

};

GameMap gameMap;

gameMap.placeKingdom(kingdoms[0]);

gameMap.placeKingdom(kingdoms[1]);

AllianceManager allianceManager;

Market market;

ConflictManager conflictManager;

int turn = 1;

bool gameRunning = true;

while (gameRunning) {

clearScreen();

displayHeader();

cout << YELLOW << "\n========= TURN " << turn << " =========" << RESET << "\n";

kingdoms[0].displayStatus();

kingdoms[1].displayStatus();

mainMenu();

int mainChoice = getValidInput(1, 6);

switch (mainChoice) {

case 1: { // Kingdom Management

kingdomMenu();

int kingdomChoice = getValidInput(1, 5);

switch (kingdomChoice) {

case 1: { // Recruit Soldiers

cout << "\nRecruit for which kingdom?\n";

cout << "1. Alpha Kingdom\n";

cout << "2. Beta Kingdom\n";

int recruitChoice = getValidInput(1, 2) - 1;

cout << "How many soldiers to recruit? (2 gold each): ";

int amount = getValidInput(1, 1000);

kingdoms[recruitChoice].recruitSoldiers(amount);

break;

}

case 2: { // Send Message

string msg;

cout << "\nEnter your message: ";

getline(cin, msg);

Message message(kingdoms[0].name, kingdoms[1].name, msg);

message.display();

break;

}

case 3: { // Trade Resources

cout << "\nTrade between:\n";

cout << "1. Alpha → Beta\n";

cout << "2. Beta → Alpha\n";

int tradeChoice = getValidInput(1, 2);

cout << "Amount to trade: ";

int amount = getValidInput(1, 10000);

if (tradeChoice == 1) market.trade(kingdoms[0], kingdoms[1], amount);

else market.trade(kingdoms[1], kingdoms[0], amount);

break;

}

case 4: { // Smuggle Resources

cout << "\nSmuggle between:\n";

cout << "1. Alpha → Beta\n";

cout << "2. Beta → Alpha\n";

int smuggleChoice = getValidInput(1, 2);

cout << "Amount to smuggle: ";

int amount = getValidInput(1, 10000);

if (smuggleChoice == 1) market.smuggle(kingdoms[0], kingdoms[1], amount);

else market.smuggle(kingdoms[1], kingdoms[0], amount);

break;

}

}

break;

}

case 2: { // Diplomacy

diplomacyMenu();

int diplomacyChoice = getValidInput(1, 3);

if (diplomacyChoice == 1) {

allianceManager.formAlliance(kingdoms[0].name, kingdoms[1].name);

}

else if (diplomacyChoice == 2) {

allianceManager.breakAlliance(kingdoms[0].name, kingdoms[1].name);

}

break;

}

case 3: { // Military Actions

militaryMenu();

int militaryChoice = getValidInput(1, 3);

if (militaryChoice == 1) { // Attack

cout << "\nWho attacks whom?\n";

cout << "1. Alpha → Beta\n";

cout << "2. Beta → Alpha\n";

int attackChoice = getValidInput(1, 2);

if (attackChoice == 1) conflictManager.attack(kingdoms[0], kingdoms[1], allianceManager);

else conflictManager.attack(kingdoms[1], kingdoms[0], allianceManager);

}

else if (militaryChoice == 2) { // Move

cout << "\nMove which kingdom=\n";

cout << "1. Alpha Kingdom\n";

cout << "2. Beta Kingdom\n";

int moveChoice = getValidInput(1, 2) - 1;

cout << "New X coordinate (0-4): ";

int newX = getValidInput(0, MAP\_SIZE - 1);

cout << "New Y coordinate (0-4): ";

int newY = getValidInput(0, MAP\_SIZE - 1);

gameMap.moveKingdom(kingdoms[moveChoice], newX, newY);

}

break;

}

case 4: { // View Map

gameMap.display();

break;

}

case 5: { // End Turn

// Passive income each turn

kingdoms[0].resources += 50;

kingdoms[1].resources += 50;

cout << GREEN << "\n💰 Both kingdoms receive 50 gold from taxes!" << RESET << "\n";

turn++;

break;

}

case 6: { // Exit Game

gameRunning = false;

cout << GREEN << "\nThanks for playing Kingdom Conquest!" << RESET << "\n";

break;

}

}

// Pause before continuing

if (gameRunning) {

cout << "\nPress Enter to continue...";

cin.ignore();

}

}

return 0;

}

Output:

