```
import React, { useState, useEffect, useCallback } from 'react';
import { Rocket, Star, Zap, Globe, Sword, Search, Settings, Package, Trophy,
RotateCcw, Fuel, Apple, Coins, Wrench } from 'lucide-react';

const SpaceCardGame = () => {
  // Game state
  const [gamePhase, setGamePhase] = useState('menu'); // menu, run, packs, upgrade,
inventory
  const [previousPhase, setPreviousPhase] = useState('menu'); // Track where we came
from
  const [turn, setTurn] = useState(0);

  // Notification system
  const [notification, setNotification] = useState(null);

  // Mission resources (reset each run)
  const [fuel, setFuel] = useState(20);
  const [food, setFood] = useState(15);
  const [maxFuel, setMaxFuel] = useState(20);
  const [maxFood, setMaxFood] = useState(15);

  // Persistent resources
  const [credits, setCredits] = useState(100);
  const [scrap, setScrap] = useState(0);
  const [energy, setEnergy] = useState(0);
  const [data, setData] = useState(0);
  const [prestigePoints, setPrestigePoints] = useState(0);

  // Skills
  const [skills, setSkills] = useState({
    explorer: 1,
    fighter: 1,
    settler: 1
  });

  // Inventory and ship
  const [inventory, setInventory] = useState([]);
  const [equippedCards, setEquippedCards] = useState({
    weapon: [],
    scanner: [],
    engine: [],
    habitat: [],
    shield: []
  });
  const [ship, setShip] = useState({
    name: 'Rookie Cruiser',
    fuelEfficiency: 1,
  weapons: 1,
  cargo: 1,
  level: 1,
  equipmentSlots: {
    weapon: 1,
    scanner: 1,
    engine: 1,
    habitat: 1,
    shield: 1
  }
});

// Progress
const [galaxiesExplored, setGalaxiesExplored] = useState(1);
const [planetsSettled, setPlanetsSettled] = useState(0);
const [battlesWon, setBattlesWon] = useState(0);
const [runNumber, setRunNumber] = useState(1);

// Mission log
const [missionLog, setMissionLog] = useState([]);

// Mission history
const [missionHistory, setMissionHistory] = useState([]);

// Current turn actions
const [currentActions, setCurrentActions] = useState([]);

// Show stuck popup when no actions are affordable
const [showStuckPopup, setShowStuckPopup] = useState(false);

// Mission end summary popup
const [showMissionSummary, setShowMissionSummary] = useState(false);
const [missionSummaryData, setMissionSummaryData] = useState(null);

// Card rarities and types
const rarities = {
  common: { name: 'Common', color: 'bg-gray-500', chance: 60 },
  uncommon: { name: 'Uncommon', color: 'bg-green-500', chance: 25 },
  rare: { name: 'Rare', color: 'bg-blue-500', chance: 10 },
  epic: { name: 'Epic', color: 'bg-purple-500', chance: 4 },
  legendary: { name: 'Legendary', color: 'bg-yellow-500', chance: 1 }
};

const cardTypes = {
  weapon: {
    name: 'Weapon',
    icon: Sword,
    category: 'fighter',
    equipEffect: 'Reduces food cost in Fighter actions',
    consumeEffect: 'Guarantees next Fighter action succeeds'
  },
  scanner: {
    name: 'Scanner',
    icon: Search,
    category: 'explorer',
    equipEffect: 'Increases Explorer action success rate',
    consumeEffect: 'Reveals exact success chances for all actions'
  },
  habitat: {
    name: 'Habitat Module',
    icon: Globe,
    category: 'settler',
    equipEffect: 'Increases Settler action rewards',
    consumeEffect: 'Restores food supplies'
  },
  engine: {
    name: 'Engine',
    icon: Zap,
    category: 'all',
    equipEffect: 'Reduces fuel costs for all actions',
    consumeEffect: 'Restores fuel supplies'
  },
  shield: {
    name: 'Shield',
    icon: Star,
    category: 'all',
    equipEffect: 'Reduces penalties from failed actions',
    consumeEffect: 'Prevents failure penalty on next action'
  }
};

// Action templates for each skill type
const actionTemplates = {
  explorer: [
    { name: 'Scan Nebula', description: 'Chart unknown stellar phenomena' },
    { name: 'Investigate Asteroid Field', description: 'Search for valuable minerals' },
    { name: 'Map Star System', description: 'Create detailed navigation charts' },
    { name: 'Deep Space Probe', description: 'Launch long-range reconnaissance' },
    { name: 'Explore Derelict Ship', description: 'Board abandoned vessel' },
    { name: 'Survey Planet', description: 'Conduct planetary analysis' },
    { name: 'Track Energy Signature', description: 'Follow mysterious signals' },
    { name: 'Chart Wormhole', description: 'Map unstable space-time rifts' }
  ],
  fighter: [
      { name: 'Pirate Convoy', description: 'Intercept raiding vessels' },
      { name: 'Defend Station', description: 'Protect civilian outpost' },
      { name: 'Bounty Hunt', description: 'Track dangerous fugitive' },
      { name: 'Escort Mission', description: 'Guard merchant vessel' },
      { name: 'Mercenary Work', description: 'Accept combat contract' },
      { name: 'Clear Space Lanes', description: 'Eliminate hostile forces' },
      { name: 'Rescue Operation', description: 'Extract trapped personnel' },
      { name: 'Patrol Sector', description: 'Maintain peace and order' }
    ],
    settler: [
      { name: 'Establish Outpost', description: 'Build new settlement' },
      { name: 'Trading Post', description: 'Create commercial hub' },
      { name: 'Mining Operation', description: 'Set up resource extraction' },
      { name: 'Diplomatic Mission', description: 'Negotiate with locals' },
      { name: 'Colony Support', description: 'Aid struggling settlement' },
      { name: 'Terraforming Project', description: 'Prepare planet for habitation' },
      { name: 'Research Station', description: 'Establish scientific facility' },
      { name: 'Cultural Exchange', description: 'Build relations with aliens' }
    ]
  };

  const riskLevels = {
    low: { name: 'Safe', color: 'text-green-400', multiplier: 0.7, successChance: 0.85 },
    medium: { name: 'Risky', color: 'text-yellow-400', multiplier: 1.0, successChance:
0.65 },
    high: { name: 'Dangerous', color: 'text-red-400', multiplier: 1.5, successChance: 0.45 }
  };

  // Auto-scroll mission log
  useEffect(() => {
    const logElement = document.getElementById('mission-log');
    if (logElement) {
      logElement.scrollTop = logElement.scrollHeight;
    }
  }, [missionLog]);

  // Card power values based on rarity
  const getCardPower = (rarity) => {
    switch(rarity) {
      case 'common': return { equip: 5, consume: 1 };
      case 'uncommon': return { equip: 10, consume: 2 };
      case 'rare': return { equip: 15, consume: 3 };
      case 'epic': return { equip: 25, consume: 4 };
      case 'legendary': return { equip: 40, consume: 5 };
      default: return { equip: 5, consume: 1 };
    }
  };
  // Add message to mission log
  const addToLog = (message) => {
    setMissionLog(prev => [...prev.slice(-4), `Turn ${turn + 1}: ${message}`]);
  };

  // Show notification
  const showNotification = (title, message, type = 'info') => {
    setNotification({ title, message, type });
    setTimeout(() => setNotification(null), 4000); // Auto-hide after 4 seconds
  };

  // Navigate to inventory with context
  const goToInventory = (fromPhase = 'menu') => {
    setPreviousPhase(fromPhase);
    setGamePhase('inventory');
  };

  // Go back from inventory
  const goBackFromInventory = () => {
    setGamePhase(previousPhase);
  };

  // Check if any actions are affordable
  const checkActionsAffordable = (actions) => {
    const anyAffordable = actions.some(action =>
      fuel >= action.costs.fuel && food >= action.costs.food && scrap >=
action.costs.scrap
    );

    if (!anyAffordable && actions.length > 0) {
      setShowStuckPopup(true);
    }

    return anyAffordable;
  };

  // Get equipment bonuses
  const getEquipmentBonuses = () => {
    const bonuses = {
      fuelCostReduction: 0,
      foodCostReduction: { fighter: 0, explorer: 0, settler: 0 },
      successBonus: { fighter: 0, explorer: 0, settler: 0 },
      rewardBonus: { fighter: 0, explorer: 0, settler: 0 },
      failurePenaltyReduction: 0
    };
  // Iterate through all equipment categories
  Object.values(equippedCards).flat().forEach(card => {
    switch (card.type) {
      case 'weapon':
        bonuses.foodCostReduction.fighter += Math.floor(card.equipPower / 10);
        break;
      case 'scanner':
        bonuses.successBonus.explorer += card.equipPower;
        break;
      case 'habitat':
        bonuses.rewardBonus.settler += card.equipPower;
        break;
      case 'engine':
        bonuses.fuelCostReduction += Math.floor(card.equipPower / 10);
        break;
      case 'shield':
        bonuses.failurePenaltyReduction += card.equipPower;
        break;
    }
  });

  return bonuses;
};

// Generate random card
const generateCard = () => {
  const rand = Math.random() * 100;
  let rarity = 'common';
  let cumulative = 0;

  for (const [key, value] of Object.entries(rarities)) {
    cumulative += value.chance;
    if (rand <= cumulative) {
      rarity = key;
      break;
    }
  }

  const typeKeys = Object.keys(cardTypes);
  const type = typeKeys[Math.floor(Math.random() * typeKeys.length)];
  const power = getCardPower(rarity);

  return {
    id: Date.now() + Math.random(),
    rarity,
    type,
    equipPower: power.equip,
      consumePower: power.consume,
      name: `${rarities[rarity].name} ${cardTypes[type].name}`,
      isEquipped: false
    };
  };

  // Generate three actions for current turn
  const generateTurnActions = () => {
    const actions = [];
    const skillTypes = ['explorer', 'fighter', 'settler'];
    const riskKeys = ['low', 'medium', 'high'];
    const bonuses = getEquipmentBonuses();

    skillTypes.forEach(skillType => {
      // Pick random action template
      const templates = actionTemplates[skillType];
      const template = templates[Math.floor(Math.random() * templates.length)];

      // Assign random risk level
      const riskKey = riskKeys[Math.floor(Math.random() * riskKeys.length)];
      const risk = riskLevels[riskKey];

      // Calculate costs and rewards based on skill level and risk
      const skillLevel = skills[skillType];
      const baseCosts = {
        explorer: { fuel: 3, food: 1, scrap: 0 },
        fighter: { fuel: 2, food: 3, scrap: 0 },
        settler: { fuel: 4, food: 3, scrap: 5 }
      };

      const baseRewards = {
        explorer: { credits: 12, data: 2, scrap: 0 },
        fighter: { credits: 18, data: 0, scrap: 3 },
        settler: { credits: 25, data: 0, scrap: 0 }
      };

      // Adjust costs based on ship, skills, and equipment
      let fuelCost = Math.max(1, Math.ceil(baseCosts[skillType].fuel * risk.multiplier) -
(ship.fuelEfficiency - 1) - bonuses.fuelCostReduction);
      let foodCost = Math.max(1, Math.ceil(baseCosts[skillType].food * risk.multiplier) -
Math.floor(skillLevel / 3) - bonuses.foodCostReduction[skillType]);
      let scrapCost = Math.max(0, Math.ceil(baseCosts[skillType].scrap * risk.multiplier) -
Math.floor(skillLevel / 2));

      // Calculate potential rewards with equipment bonuses
      let creditsReward = Math.floor(baseRewards[skillType].credits * risk.multiplier *
skillLevel * (1 + bonuses.rewardBonus[skillType] / 100));
```

```
      let dataReward = Math.floor(baseRewards[skillType].data * risk.multiplier *
skillLevel);
      let scrapReward = Math.floor(baseRewards[skillType].scrap * risk.multiplier *
skillLevel);

      // Adjust success chance with equipment bonuses
      let successChance = risk.successChance + (bonuses.successBonus[skillType] /
100);
      successChance = Math.min(0.95, successChance); // Cap at 95%

      actions.push({
        id: `${skillType}_${Date.now()}`,
        skillType,
        template,
        risk: riskKey,
        costs: { fuel: fuelCost, food: foodCost, scrap: scrapCost },
        rewards: { credits: creditsReward, data: dataReward, scrap: scrapReward },
        successChance
      });
    });

    return actions;
  };

  // Card management functions
  const equipCard = (cardId) => {
    const card = inventory.find(c => c.id === cardId);
    if (!card) return;

    const cardType = card.type;
    const currentSlots = equippedCards[cardType];
    const maxSlots = ship.equipmentSlots[cardType];

    if (currentSlots.length >= maxSlots) {
      return; // No available slots of this type
    }

    setInventory(prev => prev.map(c =>
      c.id === cardId ? { ...c, isEquipped: true } : c
    ));

    setEquippedCards(prev => ({
      ...prev,
      [cardType]: [...prev[cardType], card]
    }));
  };
```

```
const unequipCard = (cardId) => {
  const card = inventory.find(c => c.id === cardId);
  if (!card) return;

  const cardType = card.type;

  setInventory(prev => prev.map(c =>
    c.id === cardId ? { ...c, isEquipped: false } : c
  ));

  setEquippedCards(prev => ({
    ...prev,
    [cardType]: prev[cardType].filter(c => c.id !== cardId)
  }));
};

const replaceCard = (cardId, slotIndex) => {
  const card = inventory.find(c => c.id === cardId);
  if (!card) return;

  const cardType = card.type;

  // Remove the card being replaced (it's destroyed)
  setEquippedCards(prev => {
    const newSlots = [...prev[cardType]];
    newSlots[slotIndex] = card;
    return {
      ...prev,
      [cardType]: newSlots
    };
  });

  // Mark new card as equipped and remove from available inventory
  setInventory(prev => prev.map(c =>
    c.id === cardId ? { ...c, isEquipped: true } : c
  ));
};

const consumeCard = (cardId) => {
  const card = inventory.find(c => c.id === cardId);
  if (!card) return;

  // Apply consumable effect
  switch (card.type) {
    case 'weapon':
      addToLog(`💥 Used ${card.name} - next Fighter action guaranteed to succeed!`);
      showNotification('💥 Weapon Activated', `${card.name}\nNext Fighter action
guaranteed!`, 'success');
        break;
      case 'scanner':
        addToLog(`🔍 Used ${card.name} - revealed precise action success rates!`);
        showNotification('🔍 Scanner Activated', `${card.name}\nSuccess rates revealed!`,
'success');
        break;
      case 'habitat':
        const foodRestore = card.consumePower * 2;
        setFood(prev => Math.min(maxFood, prev + foodRestore));
        addToLog(`🏠 Used ${card.name} - restored ${foodRestore} food!`);
        showNotification('🏠 Food Restored', `${card.name}\n+${foodRestore} food
supplies`, 'success');
        break;
      case 'engine':
        const fuelRestore = card.consumePower * 2;
        setFuel(prev => Math.min(maxFuel, prev + fuelRestore));
        addToLog(`⚡ Used ${card.name} - restored ${fuelRestore} fuel!`);
        showNotification('⚡ Fuel Restored', `${card.name}\n+${fuelRestore} fuel
supplies`, 'success');
        break;
      case 'shield':
        addToLog(`🛡 Used ${card.name} - next action failure won't cause penalties!`);
        showNotification('🛡 Shield Activated', `${card.name}\nNext failure protected!`,
'success');
        break;
    }

    // Remove card from inventory
    setInventory(prev => prev.filter(c => c.id !== cardId));
  };

  // Open card pack
  const openPack = () => {
    if (credits >= 50) {
      setCredits(prev => prev - 50);
      const newCards = Array.from({ length: 5 }, () => generateCard());
      setInventory(prev => [...prev, ...newCards]);
    }
  };

  // Start run
  const startRun = () => {
    const baseFuel = 20 + (ship.level * 3);
    const baseFood = 15 + (ship.level * 2);
    setFuel(baseFuel);
    setFood(baseFood);
    setMaxFuel(baseFuel);
    setMaxFood(baseFood);
    setTurn(0);
    setMissionLog([`Mission ${runNumber} begins! Ship fueled and provisioned.`]);
    setCurrentActions(generateTurnActions());
    setGamePhase('run');
  };

  // End run
  const endRun = () => {
    const basePrestige = Math.floor((battlesWon + planetsSettled + galaxiesExplored) /
3);
    const efficiencyBonus = Math.floor(Math.max(0, 30 - turn) / 5);

    // Give some basic resources based on performance
    const baseScrap = Math.floor(turn / 3) + 1;
    const baseEnergy = Math.floor(turn / 4) + 1;
    const baseData = Math.floor(turn / 5) + 1;

    const summaryData = {
      runNumber,
      turns: turn,
      status: fuel <= 0 || food <= 0 ? 'Resources Depleted' : 'Mission Complete',
      gains: {
        prestige: basePrestige + efficiencyBonus,
        scrap: baseScrap,
        energy: baseEnergy,
        data: baseData
      }
    };

    setMissionSummaryData(summaryData);
    setShowMissionSummary(true);
    setShowStuckPopup(false);
  };

  // Confirm mission end and apply rewards
  const confirmMissionEnd = () => {
    if (!missionSummaryData) return;

    // Apply the rewards
    setScrap(prev => prev + missionSummaryData.gains.scrap);
    setEnergy(prev => prev + missionSummaryData.gains.energy);
    setData(prev => prev + missionSummaryData.gains.data);
    setPrestigePoints(prev => prev + missionSummaryData.gains.prestige);

    // Add to mission history
    const missionRecord = {
      id: missionSummaryData.runNumber,
      date: new Date().toLocaleString(),
      turns: missionSummaryData.turns,
      gains: missionSummaryData.gains,
      status: missionSummaryData.status
    };

    setMissionHistory(prev => [missionRecord, ...prev.slice(0, 19)]);
    setRunNumber(prev => prev + 1);

    addToLog(`Mission complete! Earned ${missionSummaryData.gains.prestige}
prestige points, ${missionSummaryData.gains.scrap} scrap, $
{missionSummaryData.gains.energy} energy, ${missionSummaryData.gains.data}
data.`);

    // Reset and return to menu
    setShowMissionSummary(false);
    setMissionSummaryData(null);
    setGamePhase('menu');
  };

  // Check if mission should end
  const checkMissionEnd = (newFuel, newFood) => {
    if (newFuel <= 0 || newFood <= 0) {
      addToLog("Mission critical! Out of essential supplies. Returning to base.");
      setTimeout(endRun, 1500);
      return true;
    }
    return false;
  };

  // Take action
  const takeAction = (action) => {
    // Check if we have enough resources
    if (fuel < action.costs.fuel || food < action.costs.food || scrap < action.costs.scrap) {
      return;
    }

    // Deduct costs
    const newFuel = fuel - action.costs.fuel;
    const newFood = food - action.costs.food;
    setFuel(newFuel);
    setFood(newFood);
    setScrap(prev => prev - action.costs.scrap);
    setTurn(prev => prev + 1);
    // Determine success using action's calculated success chance
    const success = Math.random() < action.successChance;
    const bonuses = getEquipmentBonuses();

    if (success) {
      // Apply rewards
      setCredits(prev => prev + action.rewards.credits);
      setData(prev => prev + action.rewards.data);
      setScrap(prev => prev + action.rewards.scrap);

      // Check for special achievements
      let achievementText = '';
      let notificationTitle = '✅ Success!';
      if (action.skillType === 'explorer' && Math.random() < 0.08 * skills.explorer) {
        setGalaxiesExplored(prev => prev + 1);
        achievementText = ' 🌌 New galaxy discovered!';
        notificationTitle = '🌌 Major Discovery!';
      } else if (action.skillType === 'fighter' && Math.random() < 0.15 * skills.fighter) {
        setBattlesWon(prev => prev + 1);
        achievementText = ' ⚔️ Epic victory achieved!';
        notificationTitle = '⚔️ Epic Victory!';
      } else if (action.skillType === 'settler' && Math.random() < 0.12 * skills.settler) {
        setPlanetsSettled(prev => prev + 1);
        achievementText = ' 🏛️ Planet successfully colonized!';
        notificationTitle = '🏛️ Colony Established!';
      }

      const logMessage = `✅ ${action.template.name} succeeded! +$
{action.rewards.credits} credits${action.rewards.data > 0 ? `, +${action.rewards.data}
data` : ''}${action.rewards.scrap > 0 ? `, +${action.rewards.scrap} scrap` : ''}.$
{achievementText}`;
      addToLog(logMessage);

      // Show notification
      const rewardText = `+${action.rewards.credits} credits${action.rewards.data > 0 ? `,
+${action.rewards.data} data` : ''}${action.rewards.scrap > 0 ? `, +$
{action.rewards.scrap} scrap` : ''}`;
      showNotification(notificationTitle, `${action.template.name}\n${rewardText}$
{achievementText}`, 'success');

    } else {
      // Handle failure with equipment penalty reduction
      let failureEffect = '';
      let notificationMessage = '';
      if (action.skillType === 'explorer') {
        const dataLoss = Math.max(0, Math.floor(Math.random() * 2) + 1 -
Math.floor(bonuses.failurePenaltyReduction / 20));
```

```
        setData(prev => Math.max(0, prev - dataLoss));
        failureEffect = dataLoss > 0 ? `Lost ${dataLoss} data from equipment
malfunction.` : 'Equipment damage prevented by protective systems.';
        notificationMessage = dataLoss > 0 ? `Equipment malfunction!\n-${dataLoss}
data` : 'Equipment protected by systems';
      } else if (action.skillType === 'fighter') {
        const foodLoss = Math.max(0, Math.floor(Math.random() * 3) + 1 -
Math.floor(bonuses.failurePenaltyReduction / 20));
        setFood(prev => Math.max(0, prev - foodLoss));
        failureEffect = foodLoss > 0 ? `Lost ${foodLoss} food from battle injuries.` : 'Injuries
prevented by protective systems.';
        notificationMessage = foodLoss > 0 ? `Battle injuries sustained!\n-${foodLoss}
food` : 'Injuries prevented by protection';
      } else if (action.skillType === 'settler') {
        const scrapLoss = Math.max(0, Math.floor(Math.random() * 4) + 2 -
Math.floor(bonuses.failurePenaltyReduction / 15));
        setScrap(prev => Math.max(0, prev - scrapLoss));
        failureEffect = scrapLoss > 0 ? `Lost ${scrapLoss} scrap from failed construction.` :
'Construction failure mitigated by protective systems.';
        notificationMessage = scrapLoss > 0 ? `Construction failed!\n-${scrapLoss}
scrap` : 'Failure mitigated by systems';
      }

      addToLog(`❌ ${action.template.name} failed! ${failureEffect}`);
      showNotification('❌ Action Failed', `${action.template.name}\n$
{notificationMessage}`, 'error');
    }

    // Generate new actions for next turn
    setTimeout(() => {
      if (newFuel > 0 && newFood > 0) {
        const newActions = generateTurnActions();
        setCurrentActions(newActions);
        checkActionsAffordable(newActions);
      }
    }, 100);

    // Check if mission should end
    checkMissionEnd(newFuel, newFood);
  };

  // Upgrades
  const upgradeSkill = (skill) => {
    const cost = skills[skill] * 10;
    if (prestigePoints >= cost) {
      setPrestigePoints(prev => prev - cost);
      setSkills(prev => ({ ...prev, [skill]: prev[skill] + 1 }));
    }
  };

  const upgradeShip = () => {
    const cost = ship.level * 20;
    if (scrap >= cost && energy >= cost/2) {
      setScrap(prev => prev - cost);
      setEnergy(prev => prev - cost/2);
      setShip(prev => ({
        ...prev,
        level: prev.level + 1,
        fuelEfficiency: prev.fuelEfficiency + 1,
        weapons: prev.weapons + 1,
        cargo: prev.cargo + 1,
        equipmentSlots: {
          weapon: prev.equipmentSlots.weapon + (prev.level % 2 === 0 ? 1 : 0),
          scanner: prev.equipmentSlots.scanner + (prev.level % 3 === 0 ? 1 : 0),
          engine: prev.equipmentSlots.engine + (prev.level % 4 === 0 ? 1 : 0),
          habitat: prev.equipmentSlots.habitat + (prev.level % 3 === 1 ? 1 : 0),
          shield: prev.equipmentSlots.shield + (prev.level % 5 === 0 ? 1 : 0)
        },
        name: prev.level === 1 ? 'Advanced Cruiser' : prev.level === 2 ? 'Battle Destroyer' :
'Legendary Dreadnought'
      }));
    }
  };

  // Prestige reset
  const prestige = () => {
    if (window.confirm('Are you sure you want to prestige? This will reset most progress
but give you permanent bonuses.')) {
      setCredits(100);
      setScrap(0);
      setEnergy(0);
      setData(0);
      setInventory([]);
      setEquippedCards({
        weapon: [],
        scanner: [],
        engine: [],
        habitat: [],
        shield: []
      });
      setShip({
        name: 'Rookie Cruiser',
        fuelEfficiency: 1,
        weapons: 1,
        cargo: 1,
        level: 1,
        equipmentSlots: {
          weapon: 1,
          scanner: 1,
          engine: 1,
          habitat: 1,
          shield: 1
        }
      });
      setGalaxiesExplored(1);
      setPlanetsSettled(0);
      setBattlesWon(0);
      setRunNumber(1);
      setGamePhase('menu');
    }
  };

  return (
    <div className="min-h-screen bg-gradient-to-b from-gray-900 via-blue-900 to-black
text-white p-4">
      <div className="max-w-6xl mx-auto">
        {/* Notification */}
        {notification && (
          <div className={`fixed top-4 left-1/2 transform -translate-x-1/2 z-50 p-4 rounded-
lg shadow-lg border-2 max-w-md w-full mx-4 ${
            notification.type === 'success' ? 'bg-green-800 border-green-600' :
            notification.type === 'error' ? 'bg-red-800 border-red-600' :
            'bg-blue-800 border-blue-600'
          }`}>
            <div className="flex justify-between items-start">
              <div>
                <h4 className="font-bold text-lg mb-1">{notification.title}</h4>
                <p className="text-sm whitespace-pre-line">{notification.message}</p>
              </div>
              <button
                onClick={() => setNotification(null)}
                className="text-white hover:text-gray-300 ml-2 text-xl"
              >
                ×
              </button>
            </div>
          </div>
        )}

        <header className="text-center mb-6">
          <h1 className="text-4xl font-bold text-blue-300 mb-2 flex items-center justify-
```

```
center gap-2">
          <Rocket className="text-yellow-400" />
          Stellar Expeditions
        </h1>
        <div className="flex justify-center gap-6 text-sm">
          <div className="flex items-center gap-1">
            <Coins className="text-yellow-400 w-4 h-4" />
            {credits}
          </div>
          <div className="flex items-center gap-1">
            <Wrench className="text-gray-400 w-4 h-4" />
            {scrap}
          </div>
          <div className="flex items-center gap-1">
            <span className="text-blue-400">⚡</span> {energy}
          </div>
          <div className="flex items-center gap-1">
            <span className="text-green-400">📊</span> {data}
          </div>
          <div className="flex items-center gap-1">
            <Trophy className="text-purple-400 w-4 h-4" /> {prestigePoints}
          </div>
        </div>
      </header>

      {/* Menu Phase */}
      {gamePhase === 'menu' && (
        <div className="space-y-6">
          <div className="bg-gray-800 rounded-lg p-6">
            <h2 className="text-2xl mb-4">Command Center - Run #{runNumber}</h2>
            <div className="grid grid-cols-1 md:grid-cols-5 gap-4">
              <button
                onClick={startRun}
                className="bg-blue-600 hover:bg-blue-700 p-4 rounded-lg transition-
colors"
              >
                🚀 Launch Mission
              </button>
              <button
                onClick={() => setGamePhase('packs')}
                className="bg-green-600 hover:bg-green-700 p-4 rounded-lg transition-
colors"
              >
                <Package className="inline mr-2" size={20} />
                Card Packs
              </button>
              <button
          onClick={() => goToInventory('menu')}
          className="bg-orange-600 hover:bg-orange-700 p-4 rounded-lg transition-
colors"
        >
          🎒 Inventory ({inventory.length})
        </button>
        <button
          onClick={() => setGamePhase('history')}
          className="bg-cyan-600 hover:bg-cyan-700 p-4 rounded-lg transition-
colors"
        >
          📜 Mission History ({missionHistory.length})
        </button>
        <button
          onClick={() => setGamePhase('upgrade')}
          className="bg-purple-600 hover:bg-purple-700 p-4 rounded-lg transition-
colors"
        >
          ⬆️ Upgrades
        </button>
      </div>
    </div>

    <div className="grid grid-cols-1 md:grid-cols-2 gap-6">
      <div className="bg-gray-800 rounded-lg p-6">
        <h3 className="text-xl mb-3">Ship Status</h3>
        <div className="space-y-2">
          <div><strong>{ship.name}</strong> (Level {ship.level})</div>
          <div>Fuel Efficiency: {ship.fuelEfficiency} | Weapons: {ship.weapons} |
Cargo: {ship.cargo}</div>
          <div className="text-sm">
            Equipment Slots:
            ⚔️{Object.values(equippedCards.weapon).length}/
{ship.equipmentSlots.weapon}
            🔍{Object.values(equippedCards.scanner).length}/
{ship.equipmentSlots.scanner}
            ⚡{Object.values(equippedCards.engine).length}/
{ship.equipmentSlots.engine}
            🏠{Object.values(equippedCards.habitat).length}/
{ship.equipmentSlots.habitat}
            🛡️{Object.values(equippedCards.shield).length}/
{ship.equipmentSlots.shield}
          </div>
          <div className="text-sm text-gray-400">Next mission: {20 + (ship.level * 3)}
fuel, {15 + (ship.level * 2)} food</div>
        </div>
      </div>
```

```
            <div className="bg-gray-800 rounded-lg p-6">
              <h3 className="text-xl mb-3">Skills</h3>
              <div className="space-y-2">
                <div>🔍 Explorer: Level {skills.explorer}</div>
                <div>⚔️ Fighter: Level {skills.fighter}</div>
                <div>🏛️ Settler: Level {skills.settler}</div>
              </div>
            </div>
          </div>

          <div className="bg-gray-800 rounded-lg p-6">
            <h3 className="text-xl mb-3">Career Statistics</h3>
            <div className="grid grid-cols-3 gap-4 text-center">
              <div>
                <div className="text-2xl">{galaxiesExplored}</div>
                <div className="text-sm text-gray-400">Galaxies Explored</div>
              </div>
              <div>
                <div className="text-2xl">{planetsSettled}</div>
                <div className="text-sm text-gray-400">Planets Settled</div>
              </div>
              <div>
                <div className="text-2xl">{battlesWon}</div>
                <div className="text-sm text-gray-400">Battles Won</div>
              </div>
            </div>
          </div>

          {prestigePoints >= 50 && (
            <div className="bg-purple-800 rounded-lg p-6">
              <button
                onClick={prestige}
                className="bg-purple-600 hover:bg-purple-700 p-3 rounded-lg transition-
colors flex items-center gap-2"
              >
                <RotateCcw size={20} />
                Prestige Reset (Unlock permanent bonuses)
              </button>
            </div>
          )}
        </div>
      )}

      {/* Run Phase */}
      {gamePhase === 'run' && (
        <div className="space-y-6">
```

```
<div className="bg-gray-800 rounded-lg p-6">
  <div className="flex justify-between items-center mb-4">
    <h2 className="text-2xl">Mission #{runNumber} - Turn {turn}</h2>
    <button
      onClick={endRun}
      className="bg-red-600 hover:bg-red-700 px-4 py-2 rounded-lg"
    >
      End Mission
    </button>
  </div>

  {/* Resource bars */}
  <div className="grid grid-cols-2 gap-4 mb-4">
    <div>
      <div className="flex items-center gap-2 mb-1">
        <Fuel className="w-4 h-4 text-orange-400" />
        <span>Fuel: {fuel}/{maxFuel}</span>
      </div>
      <div className="w-full bg-gray-700 rounded-full h-2">
        <div
          className="bg-orange-400 h-2 rounded-full transition-all duration-300"
          style={{ width: `${(fuel / maxFuel) * 100}%` }}
        ></div>
      </div>
    </div>

    <div>
      <div className="flex items-center gap-2 mb-1">
        <Apple className="w-4 h-4 text-green-400" />
        <span>Food: {food}/{maxFood}</span>
      </div>
      <div className="w-full bg-gray-700 rounded-full h-2">
        <div
          className="bg-green-400 h-2 rounded-full transition-all duration-300"
          style={{ width: `${(food / maxFood) * 100}%` }}
        ></div>
      </div>
    </div>
  </div>
</div>

{/* Current Turn Actions */}
<div className="bg-gray-800 rounded-lg p-6">
  <div className="flex justify-between items-center mb-4">
    <h3 className="text-xl">Choose Your Action</h3>
    <button
      onClick={() => goToInventory('run')}
```

```
                className="bg-orange-600 hover:bg-orange-700 px-3 py-1 rounded text-
sm"
              >
                🎒 Use Items
              </button>
            </div>
            <div className="grid grid-cols-1 md:grid-cols-3 gap-4">
              {currentActions.map((action, index) => {
                const canAfford = fuel >= action.costs.fuel && food >= action.costs.food &&
scrap >= action.costs.scrap;
                const risk = riskLevels[action.risk];
                const skillIcons = {
                  explorer: Search,
                  fighter: Sword,
                  settler: Globe
                };
                const ActionIcon = skillIcons[action.skillType];

                return (
                  <button
                    key={action.id}
                    onClick={() => takeAction(action)}
                    disabled={!canAfford}
                    className={`p-4 rounded-lg transition-colors text-left ${
                      canAfford
                        ? 'bg-gray-700 hover:bg-gray-600 border-2 border-gray-600
hover:border-gray-500'
                        : 'bg-gray-800 border-2 border-gray-700 opacity-50 cursor-not-allowed'
                    }`}
                  >
                    <div className="flex items-center gap-2 mb-2">
                      <ActionIcon className="w-5 h-5" />
                      <span className="font-bold text-lg">{action.template.name}</span>
                      <span className={`text-sm ${risk.color} ml-auto`}>({risk.name})</span>
                    </div>

                    <div className="text-sm text-gray-300 mb-3">
                      {action.template.description}
                    </div>

                    <div className="text-xs space-y-1">
                      <div className="text-red-300">
                        Costs: {action.costs.fuel} fuel, {action.costs.food} food
                        {action.costs.scrap > 0 && `, ${action.costs.scrap} scrap`}
                      </div>
                      <div className="text-green-300">
                        Rewards: {action.rewards.credits} credits
                        {action.rewards.data > 0 && `, ${action.rewards.data} data`}
                        {action.rewards.scrap > 0 && `, ${action.rewards.scrap} scrap`}
                      </div>
                      <div className="text-gray-400">
                        Success chance: {Math.floor(action.successChance * 100)}%
                      </div>
                    </div>
                  </button>
                );
              })}
            </div>

            {currentActions.length === 0 && (
              <div className="text-center text-gray-400 py-8">
                Generating new opportunities...
              </div>
            )}
          </div>

          {/* Stuck Popup */}
          {showStuckPopup && (
            <div className="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-
center z-50">
              <div className="bg-gray-800 border-2 border-red-600 rounded-lg p-6 max-
w-md mx-4">
                <h3 className="text-xl font-bold text-red-400 mb-4">⚠️ Insufficient
Resources</h3>
                <p className="text-gray-300 mb-6">
                  You don't have enough resources to take any of the available actions.
                  You can either end the mission now or use items to restore resources.
                </p>
                <div className="flex gap-3">
                  <button
                    onClick={endRun}
                    className="bg-red-600 hover:bg-red-700 px-4 py-2 rounded-lg
transition-colors"
                  >
                    End Mission
                  </button>
                  <button
                    onClick={() => {
                      setShowStuckPopup(false);
                      goToInventory('run');
                    }}
                    className="bg-orange-600 hover:bg-orange-700 px-4 py-2 rounded-lg
transition-colors"
                  >
                    Use Items ({inventory.filter(c => !c.isEquipped).length})
                  </button>
                  <button
                    onClick={() => setShowStuckPopup(false)}
                    className="bg-gray-600 hover:bg-gray-700 px-4 py-2 rounded-lg
transition-colors"
                  >
                    Cancel
                  </button>
                </div>
              </div>
            </div>
          )}

          {/* Mission Log */}
          <div className="bg-gray-800 rounded-lg p-4">
            <h3 className="text-lg mb-2">Mission Log</h3>
            <div
              id="mission-log"
              className="space-y-1 text-sm max-h-32 overflow-y-auto"
            >
              {missionLog.map((log, index) => (
                <div key={index} className="text-gray-300">{log}</div>
              ))}
            </div>
          </div>

          {/* Mission Summary Popup */}
          {showMissionSummary && missionSummaryData && (
            <div className="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-
          center z-50">
              <div className={`rounded-lg p-6 max-w-md mx-4 border-2 ${
                missionSummaryData.status === 'Mission Complete'
                  ? 'bg-green-900 border-green-600'
                  : 'bg-red-900 border-red-600'
              }`}>
                <h3 className="text-2xl font-bold mb-4 text-center">
                  {missionSummaryData.status === 'Mission Complete' ? '🎉 Mission
Complete!' : '⚠️ Mission Ended'}
                </h3>

                <div className="space-y-3 mb-6">
                  <div className="text-center">
                    <div className="text-lg font-bold">Mission
#{missionSummaryData.runNumber}</div>
                    <div className="text-gray-300">Completed in
{missionSummaryData.turns} turns</div>
```

```
                </div>

                <div className="bg-gray-800 rounded p-4">
                  <h4 className="font-bold mb-2">Rewards Earned:</h4>
                  <div className="grid grid-cols-2 gap-2 text-sm">
                    <div>
                      <span className="text-purple-400">Prestige:</span>
                      <span className="float-right font-bold">+
{missionSummaryData.gains.prestige}</span>
                    </div>
                    <div>
                      <span className="text-gray-400">Scrap:</span>
                      <span className="float-right font-bold">+
{missionSummaryData.gains.scrap}</span>
                    </div>
                    <div>
                      <span className="text-blue-400">Energy:</span>
                      <span className="float-right font-bold">+
{missionSummaryData.gains.energy}</span>
                    </div>
                    <div>
                      <span className="text-green-400">Data:</span>
                      <span className="float-right font-bold">+
{missionSummaryData.gains.data}</span>
                    </div>
                  </div>
                </div>
              </div>

              <button
                onClick={confirmMissionEnd}
                className={`w-full py-3 px-4 rounded-lg font-bold transition-colors ${
                  missionSummaryData.status === 'Mission Complete'
                    ? 'bg-green-600 hover:bg-green-700'
                    : 'bg-red-600 hover:bg-red-700'
                }`}
              >
                Continue
              </button>
            </div>
          </div>
        )}
      </div>
    )}

    {/* Mission History Phase */}
    {gamePhase === 'history' && (
      <div className="space-y-6">
        <div className="flex justify-between items-center">
          <h2 className="text-2xl">Mission History</h2>
          <button
            onClick={() => setGamePhase('menu')}
            className="bg-gray-600 hover:bg-gray-700 px-4 py-2 rounded-lg"
          >
            Back to Menu
          </button>
        </div>

        <div className="bg-gray-800 rounded-lg p-6">
          <h3 className="text-xl mb-4">Completed Missions ({missionHistory.length})</
h3>

          {missionHistory.length === 0 ? (
            <div className="text-center text-gray-400 py-8">
              No missions completed yet. Launch your first mission!
            </div>
          ) : (
            <div className="space-y-4 max-h-96 overflow-y-auto">
              {missionHistory.map((mission) => (
                <div key={mission.id} className={`p-4 rounded-lg border-2 ${
                  mission.status === 'Mission Complete'
                    ? 'bg-green-900 border-green-600'
                    : 'bg-red-900 border-red-600'
                }`}>
                  <div className="flex justify-between items-start mb-2">
                    <div>
                      <h4 className="font-bold text-lg">Mission #{mission.id}</h4>
                      <p className="text-sm text-gray-400">{mission.date}</p>
                    </div>
                    <div className={`px-2 py-1 rounded text-sm ${
                      mission.status === 'Mission Complete'
                        ? 'bg-green-700 text-green-200'
                        : 'bg-red-700 text-red-200'
                    }`}>
                      {mission.status}
                    </div>
                  </div>

                  <div className="grid grid-cols-2 md:grid-cols-3 gap-4 text-sm">
                    <div>
                      <span className="text-gray-400">Duration:</span>
                      <div className="font-bold">{mission.turns} turns</div>
                    </div>
                    <div>
                      <span className="text-gray-400">Prestige:</span>
                      <div className="font-bold text-purple-400">+{mission.gains.prestige}
</div>
                    </div>
                    <div>
                      <span className="text-gray-400">Resources:</span>
                      <div className="font-bold">
                        +{mission.gains.scrap} scrap, +{mission.gains.energy} energy, +
{mission.gains.data} data
                      </div>
                    </div>
                  </div>
                </div>
              ))}
            </div>
          )}
        </div>
      </div>
    )}

    {/* Inventory Phase */}
    {gamePhase === 'inventory' && (
      <div className="space-y-6">
        <div className="flex justify-between items-center">
          <h2 className="text-2xl">Equipment & Inventory</h2>
          <button
            onClick={goBackFromInventory}
            className="bg-gray-600 hover:bg-gray-700 px-4 py-2 rounded-lg"
          >
            {previousPhase === 'run' ? 'Back to Mission' : 'Back to Menu'}
          </button>
        </div>

        {/* Equipped Cards */}
        <div className="bg-gray-800 rounded-lg p-6">
          <h3 className="text-xl mb-4">Equipment Loadout</h3>

          {Object.entries(ship.equipmentSlots).map(([slotType, maxSlots]) => {
            const equipped = equippedCards[slotType] || [];
            const SlotIcon = cardTypes[slotType]?.icon || Star;

            return (
              <div key={slotType} className="mb-6">
                <h4 className="text-lg mb-2 flex items-center gap-2">
                  <SlotIcon className="w-5 h-5" />
                  {cardTypes[slotType]?.name} Slots ({equipped.length}/{maxSlots})
                </h4>
```

```
                <div className="grid grid-cols-2 md:grid-cols-4 gap-3">
                  {Array.from({ length: maxSlots }).map((_, slotIndex) => {
                    const card = equipped[slotIndex];

                    return (
                      <div key={slotIndex} className="relative">
                        {card ? (
                          <div className={`${rarities[card.rarity].color} p-3 rounded-lg text-
center relative`}>
                            <SlotIcon className="mx-auto mb-1" size={16} />
                            <div className="text-xs font-bold">{rarities[card.rarity].name}</
div>
                            <div className="text-xs">{cardTypes[card.type]?.name}</div>
                            <div className="text-xs mt-1">+{card.equipPower}</div>
                            <button
                              onClick={() => unequipCard(card.id)}
                              className="absolute top-1 right-1 bg-red-600 hover:bg-red-700
text-white text-xs px-1 py-0.5 rounded"
                            >
                              ×
                            </button>
                          </div>
                        ) : (
                          <div className="bg-gray-700 border-2 border-dashed border-
gray-600 p-3 rounded-lg flex items-center justify-center min-h-20">
                            <span className="text-gray-500 text-xs">Empty</span>
                          </div>
                        )}
                      </div>
                    );
                  })}
                </div>
              </div>
            );
          })}
        </div>

        {/* Available Cards */}
        <div className="bg-gray-800 rounded-lg p-6">
          <h3 className="text-xl mb-4">Available Cards ({inventory.filter(c => !
c.isEquipped).length})</h3>

          {inventory.filter(c => !c.isEquipped).length === 0 ? (
            <div className="text-center text-gray-400 py-8">
              No cards available. Open some card packs!
            </div>
          ) : (
            <div className="space-y-6 max-h-96 overflow-y-auto">
              {Object.entries(cardTypes).map(([cardType, cardInfo]) => {
                const cardsOfType = inventory
                  .filter(card => !card.isEquipped && card.type === cardType)
                  .sort((a, b) => {
                    // Sort by rarity (legendary first, common last)
                    const rarityOrder = { legendary: 5, epic: 4, rare: 3, uncommon: 2,
common: 1 };
                    return rarityOrder[b.rarity] - rarityOrder[a.rarity];
                  });

                if (cardsOfType.length === 0) return null;

                const CardIcon = cardInfo.icon || Star;
                const equippedOfType = equippedCards[cardType] || [];
                const maxSlots = ship.equipmentSlots[cardType] || 0;

                return (
                  <div key={cardType} className="border-t border-gray-700 pt-4
first:border-t-0 first:pt-0">
                    <h4 className="text-lg mb-3 flex items-center gap-2">
                      <CardIcon className="w-5 h-5" />
                      {cardInfo.name}s ({cardsOfType.length})
                      <span className="text-sm text-gray-400">
                        - {equippedOfType.length}/{maxSlots} equipped
                      </span>
                    </h4>

                    <div className="grid grid-cols-2 md:grid-cols-4 lg:grid-cols-6 gap-3">
                      {cardsOfType.map(card => {
                        const canEquip = equippedOfType.length < maxSlots;

                        return (
                          <div
                            key={card.id}
                            className={`${rarities[card.rarity].color} p-3 rounded-lg text-center
text-xs relative`}
                          >
                            <CardIcon className="mx-auto mb-1" size={16} />
                            <div className="font-bold">{rarities[card.rarity].name}</div>
                            <div>{cardInfo.name}</div>
                            <div className="mt-1">
                              <div>Equip: +{card.equipPower}</div>
                              <div>Use: +{card.consumePower}</div>
                            </div>
                <div className="mt-2 space-y-1">
                 {canEquip ? (
                  <button
                   onClick={() => equipCard(card.id)}
                   className="w-full bg-blue-600 hover:bg-blue-700 text-white
text-xs px-2 py-1 rounded"
                  >
                   Equip
                  </button>
                 ) : (
                  <div className="text-xs text-gray-400 mb-1">Slots full</div>
                 )}

                 <button
                  onClick={() => consumeCard(card.id)}
                  className="w-full bg-green-600 hover:bg-green-700 text-white
text-xs px-2 py-1 rounded"
                 >
                  Use
                 </button>
                </div>

                <div className="text-xs mt-1 text-gray-200">
                 <div>⚙️ {cardInfo.equipEffect}</div>
                 <div>💊 {cardInfo.consumeEffect}</div>
                </div>
               </div>
              );
             })}
            </div>
           </div>
          );
         })}
        </div>
       )}
      </div>
     </div>
    )}

    {/* Card Packs Phase */}
    {gamePhase === 'packs' && (
     <div className="space-y-6">
      <div className="flex justify-between items-center">
       <h2 className="text-2xl">Card Packs</h2>
       <button
        onClick={() => setGamePhase('menu')}
        className="bg-gray-600 hover:bg-gray-700 px-4 py-2 rounded-lg"
       >
        Back to Menu
       </button>
      </div>

      <div className="bg-gray-800 rounded-lg p-6 text-center">
       <div className="text-6xl mb-4">📦</div>
       <h3 className="text-xl mb-4">Standard Pack</h3>
       <p className="mb-4">5 random cards - 50 Credits</p>
       <button
        onClick={openPack}
        disabled={credits < 50}
        className="bg-blue-600 hover:bg-blue-700 disabled:bg-gray-600 px-6 py-3
rounded-lg text-lg transition-colors"
       >
        Open Pack
       </button>
      </div>

      <div className="bg-gray-800 rounded-lg p-6">
       <h3 className="text-xl mb-4">Recent Cards ({inventory.length})</h3>
       <div className="grid grid-cols-2 md:grid-cols-5 gap-2 max-h-60 overflow-y-
auto">
        {inventory.slice(-20).map(card => {
         const CardIcon = cardTypes[card.type]?.icon || Star;
         return (
          <div
           key={card.id}
           className={`${rarities[card.rarity].color} p-3 rounded-lg text-center text-xs
relative`}
          >
           <CardIcon className="mx-auto mb-1" size={16} />
           <div className="font-bold">{rarities[card.rarity].name}</div>
           <div>{cardTypes[card.type]?.name}</div>
           <div className="mt-1">
            <div>Equip: +{card.equipPower}</div>
            <div>Use: +{card.consumePower}</div>
           </div>
           {card.isEquipped && (
            <div className="absolute top-1 right-1 bg-blue-600 text-white text-xs
px-1 py-0.5 rounded">
             ⚙️
            </div>
           )}
          </div>
         );
        })}
```

```
          </div>
        </div>
      </div>
    ))}

    {/* Upgrade Phase */}
    {gamePhase === 'upgrade' && (
      <div className="space-y-6">
        <div className="flex justify-between items-center">
          <h2 className="text-2xl">Upgrades</h2>
          <button
            onClick={() => setGamePhase('menu')}
            className="bg-gray-600 hover:bg-gray-700 px-4 py-2 rounded-lg"
          >
            Back to Menu
          </button>
        </div>

        <div className="grid grid-cols-1 md:grid-cols-2 gap-6">
          <div className="bg-gray-800 rounded-lg p-6">
            <h3 className="text-xl mb-4">Skill Upgrades (Prestige Points)</h3>
            <div className="space-y-3">
              {Object.entries(skills).map(([skill, level]) => (
                <div key={skill} className="flex justify-between items-center">
                  <span className="capitalize">{skill} (Level {level})</span>
                  <button
                    onClick={() => upgradeSkill(skill)}
                    disabled={prestigePoints < level * 10}
                    className="bg-purple-600 hover:bg-purple-700 disabled:bg-gray-600
px-3 py-1 rounded text-sm"
                  >
                    Upgrade ({level * 10} PP)
                  </button>
                </div>
              ))}
            </div>
          </div>

          <div className="bg-gray-800 rounded-lg p-6">
            <h3 className="text-xl mb-4">Ship Upgrades</h3>
            <div className="space-y-3">
              <div>Current: {ship.name} (Level {ship.level})</div>
              <div className="text-sm text-gray-400">
                Next upgrade: +1 all stats
                {(ship.level + 1) % 2 === 0 && ' +1 weapon slot'}
                {(ship.level + 1) % 3 === 0 && ' +1 scanner slot'}
                {(ship.level + 1) % 4 === 0 && ' +1 engine slot'}
```

```
                  {(ship.level + 1) % 3 === 1 && ' +1 habitat slot'}
                  {(ship.level + 1) % 5 === 0 && ' +1 shield slot'}
                </div>
                <div className="text-sm text-gray-400">
                  Cost: {ship.level * 20} Scrap, {ship.level * 10} Energy
                </div>
                <button
                  onClick={upgradeShip}
                  disabled={scrap < ship.level * 20 || energy < ship.level * 10}
                  className="bg-blue-600 hover:bg-blue-700 disabled:bg-gray-600 px-4
py-2 rounded"
                >
                  Upgrade Ship
                </button>
              </div>
            </div>
          </div>
        </div>
      )}
    </div>
  </div>
  );
};

export default SpaceCardGame;
```