

ESPECIFICACIÓN TÉCNICA INTEGRAL: SIMULADOR ELECTRONOVA

Versión: 2.0 (Visión Unificada - Con Decisiones Finales)

Estado: Especificación de Diseño Detallada

Fecha: 2024-05-24

1. VISIÓN GLOBAL Y OBJETIVOS

El Simulador ElectroNova es una plataforma de simulación empresarial (serious game) diseñada para la educación ejecutiva. Su objetivo es proporcionar un entorno inmersivo donde equipos de estudiantes gestionen una empresa de electrónica de consumo, tomando decisiones integrales y secuenciales que abarcan la cadena de valor completa: abastecimiento ético, manufactura con restricciones, logística multimodal, estrategia comercial segmentada, finanzas y gestión de riesgos, todo dentro de un mercado competitivo y dinámico.

Esta especificación fusiona de manera coherente la visión estratégica original, el desarrollo funcional de la versión 1.3 y el roadmap de expansión, resolviendo todas las desviaciones identificadas y estableciendo un plan de desarrollo detallado y factible.

2. MODELO DE NEGOCIO UNIFICADO: REGLAS Y PARÁMETROS

2.1. Entidades Fundamentales y sus Atributos

Entidad

Atributos Clave

Descripción y

Comportamiento

Empresa (Company)	<code>id</code> <code>name</code> <code>cash</code> <code>techLevel</code> <code>ethicsIndex</code> <code>productionQuota</code> <code>assignedCDP</code>	Entidad controlada por un equipo. Su <code>productionQuota</code> se recalcula cada ronda. El <code>assignedCDP</code> es siempre "Novaterra".
Producto (Product Line - 3)	<code>id</code> <code>name</code> ("Alta", "Media", "Básica"), <code>baseProductionCost</code> <code>rawMaterialFormula</code>	Cada gama tiene un costo base de manufactura y una fórmula fija de consumo de MP (ej: Gama Alta consume 70% Alfa, 30% Beta).
Materia Prima (Raw Material - 3)	<code>id</code> <code>name</code> ("Alfa", "Beta", "Omega"), <code>baseCost</code>	Insumo para producción. Su costo final depende del proveedor elegido.
Proveedor (Supplier)	<code>type</code> ("local", "imported"), <code>costMultiplier</code> <code>leadTime</code> <code>ethicsImpact</code>	Local: Lead Time 1 ronda, +\$2 de costo, +5 pts de ética por lote. Importado: Lead Time 2 rondas, costo base, impacto ético 0.
Plaza/Mercado (Market - 4)	<code>id</code> <code>name</code> ("Novaterra", "Solis", "Veridia", "Aurinea"), <code>demandPotential</code> <code>priceSensitivity</code>	Mercados con comportamientos definidos. Novaterra es la sede de la planta.

```
priceHardCap
```

```
qualityPreference
```

Lote de MP en Tránsito	<code>materialType</code> <code>supplier</code>	Entidad temporal creada al comprar MP.
	<code>units</code> <code>costPerUnit</code>	
	<code>roundsUntilArrival</code>	<code>roundsUntilArrival</code> decrece 1 por ronda.

Lote de PT en Transito	<code>productLine</code>	Entidad temporal creada al enviar logística.
	<code>destination</code> <code>units</code>	
	<code>costPerUnit</code>	
	<code>shipmentMethod</code>	
	<code>roundsUntilArrival</code>	

Lote de PT en Inventario	<code>productLine</code> <code>market</code>	Inventario vendible en una plaza.
	<code>units</code> <code>costPerUnit</code>	<code>ageInRounds</code> aumenta 1
	<code>ageInRounds</code>	por ronda sin venderse.

2.2. Reglas de Juego Críticas y Específicas

A. Capacidad de Producción Compartida y Asignación

- Regla: Existe una capacidad máxima agregada de la planta de 6,000 unidades por ronda.
- Mecánica: Al inicio de cada ronda, el sistema calcula: `Capacidad_Por_Empresa = 6,000 / Número_de_Empresas_Activas`. Una empresa está "activa" si no está en bancarrota y no ha cerrado.
- Validación: El sistema bloqueará el envío del formulario de decisiones si la suma de unidades programadas para producción en las 3 gamas excede la `productionQuota` de la empresa para esa ronda.
- Capacidad Inactiva: La capacidad no utilizada se pierde y no es transferible, acumulable o reasignable.

B. Sistema de Compras de Materia Prima y Proveedores

- Opciones: Por cada tipo de MP (Alfa, Beta, Omega), el jugador elige un proveedor por lote.
- Parámetros por Proveedor:

```
Lote costo = precio_base_MP * 1.2  
= 1 ronda impacto_ético = +5 puntos por lote comprado  
Importado costo = precio_base_MP  
lead_time = 2  
rondas impacto_ético = 0
```

- Flujo: Al confirmar la compra, se crea un Lote de MP en Tránsito con su roundsUntilArrival. Al llegar a 0, las unidades se suman al inventario de MP disponible.

C. Costo de Obsolescencia

- Regla: Todo lote de Producto Terminado en el inventario de una plaza con ageInRounds > 3 genera un costo financiero.
- Cálculo Específico: Costo_Obsolescencia_Lote = (Unidades_en_Lote * Costo_Unitario_Producción_del_Lote) * (Porcentaje_Obsolescencia / 100).
- Parámetro Administrable: El administrador define el Porcentaje_Obsolescencia al configurar la partida. Valor por defecto: 10%. Este campo será obsolescencePenaltyRate en la configuración de la partida (game settings).
- Aplicación: El costo se deduce del cash de la empresa durante el paso de Cierre Financiero de la ronda. Afecta directamente el Estado de Resultados.

2.3. Motor de Mercado Híbrido: ECPCIM con Elasticidad Integrada

Algoritmo Paso a Paso (por Plaza m y Gama p):

1. Calcular Demanda Total del Segmento (D_{total}):
$$D_{total}(m, p) = Demanda_Potencial_Base(m, p) * Factor_Evento_Aleatorio(m)$$
2. Calcular Score Competitivo (SC) para cada Empresa i:
$$SC(i, m, p) = w1 * Puntuación_Precio(i, m, p) + w2 * Puntuación_Calidad(i, p) + w3 * Puntuación_Marketing(i, m) + w4 * Puntuación_Disponibilidad(i, m, p) + w5 * Puntuación_Ética(i)$$

```

Puntuación_Precio = (Precio_Máximo_Aceptable(m) -
Precio_Empresa(i,m,p)) / Rango_de_Precios(normalizado e
inventario)

Puntuación_Calidad = f(Nivel_Tecnológico(i),
Calidad_MP_Utilizada(i,p))

Puntuación_Marketing = 1 + log10(Inversión_Marketing(i,m) + 1)
* 0.15

Puntuación_Disponibilidad = min(1,
Inventario_Disponible(i,m,p) / Demanda_Potencial_Individual)

Puntuación_Ética = (Índice_Ético(i) / 100) * w5

```

3. Asignar Cuota de Mercado Potencial:

```
Cuota_Potencial(i,m,p) = D_total(m,p) * (SC(i,m,p) /
Σ(SC(todas,m,p)))
```

4. Aplicar Elasticidad y Hard Cap por Empresa:

```
Demanda_Final(i,m,p) = Cuota_Potencial(i,m,p) *
(Precio_Refencia(m) / Precio_Empresa(i,m,p))^Sensibilidad(m)
Si Precio_Empresa(i,m,p) > Precio_HardCap(m) Entonces
Demanda_Final(i,m,p) = Demanda_Final(i,m,p) *
((Precio_HardCap(m)/Precio_Empresa(i,m,p))^4).
```

5. Validar Contra Inventario y Registrar Ventas:

```
Ventas_Reales(i,m,p) = min(Demanda_Final(i,m,p),
Inventario_Disponible(i,m,p))
Si Ventas_Reales < Proyección_Voluntaria_del_Jugador, aplicar
penalización de -0.075 al componente de fidelidad del SC en la siguiente ronda
para esa plaza.
```

3. PLAN DE DESARROLLO DETALLADO POR FASES

FASE 0: REFACTORIZACIÓN DE LA BASE Y MODELOS (Sprint 1-2)

Objetivo: Preparar la base de código v1.3 para soportar el nuevo modelo de datos sin romper la funcionalidad actual.

1. Tarea 0.1: Extender Esquemas de MongoDB.

```
Modifica CompanySchema para incluir techLevel, ethicsIndex
y productionQuota
```

- o Cambiar `rawMaterials`, `factoryStock`, `inTransit`, `inventory` de objetos simples a arrays de subdocumentos con los campos detallados en la Sección 4.1.

o Crear nuevas esquemas `ProductSchema`, `RawMaterialSchema`, `MarketSchema` y rellenarlos con datos maestros iniciales.

2. Tarea 0.2: Crear Servicio de Configuración de Partida.

o Endpoint `POST /api/games/:gameId/settings` que el usuario defina `totalCapacity`, `obsolescencePenaltyRate`, `initialCash`, `maxRounds`

3. Tarea 0.3: Actualizar Lógica de Decisión Existente.

- o Re-factorizar funciones de `processProduction`, `processLogistics` para iterar sobre arrays de productos.

FASE 1: NÚCLEO ESTRATÉGICO - v2.0 (Sprint 3-6)

Objetivo: Implementar el modelo de negocio unificado con las 3 reglas críticas.

1. Backend (Motor Principal):

- o T1.1. Servicio `CapacityService` con método `calculateAndAssignQuotas(gameRound)`
- o T1.2. Servicio `ProcurementService` con método `processPurchases(decisions)` creando lotes de MP en tránsito con proveedor y lead time
- o T1.3. Servicio `MarketEngineV2` que implementa el algoritmo híbrido paso a paso. Esto incluye algoritmos.
- o T1.4. Servicio `ObsolescenceService` con método `calculateAndApplyCosts(companies)` que itera sobre `inventory` y aplica la fórmula del 10% sobre el `unitCost`
- o T1.5. Nuevos endpoints `POST /api/decisions` para aceptar el formulario unificado, `GET /api/decisions/history` para el panel de estrategias.

2. Frontend (Interfaz Unificada):

- o T1.6 Componente `<DecisionFormV2>` con pestañas/accordion para Producción (con cuota visible), Compras MP (con selector de proveedor), Logística (por producto y destino), Precios & Marketing (matriz 4x3), Estrategia (I+D, estudios).
- o T1.7 Componente `<StrategyPanel>` Muestra tabla histórica, permite clonar decisiones de rondas anteriores en la ronda actual.

- T1.8: Panel de Admin extendido: Vista para configurar `GameSettings`
Tabla en tiempo real de `Ranking` (por utilidad neta acumulada), Modal para ver `Estrategias de Todos`

FASE 2: INTELIGENCIA FINANCIERA - v2.1 (Sprint 7-8)

Objetivo: Implementar contabilidad gerencial precisa.

1. T2.1: Diseñar esquema `FinancialStatementSchema` con subdocumentos para `incomeStatement` y `balanceSheet`.
2. T2.2: Servicio `AccountingService` que, en cada cierre de ronda:
 - Calcula Costo de Ventas (`COGS`) usando método FIFO sobre los lotes de inventario vendidos.
 - Calcula Utilidad Neta incorporando todos los costos nuevos (obsolescencia, premium por MP local, I+D).
 - Construye el Balance valorando inventarios a costo promedio.
3. T2.3: Componentes `<IncomeStatementReport>` y `<BalanceSheetReport>` en el dashboard del estudiante y del admin.

FASE 3: REALISMO Y ADAPTACIÓN - v2.2 (Sprint 9-10)

Objetivo: Introducir incertidumbre.

1. T3.1: Crear base de datos de `EventSchema` (`type`, `target`, `severity`, `duration`).
2. T3.2: Servicio `EventEngine` que, al iniciar una ronda, tiene un 15% de probabilidad de disparar un evento (ej: "Incremento del 200% en costo logístico terrestre por 2 rondas").
3. T3.3: Componente `<NewsFeed>` que muestra noticias en el dashboard.

FASE 4: DIFERENCIACIÓN Y ÉTICA PROFUNDA - v2.3 (Sprint 11-12)

Objetivo: Activar decisiones estratégicas a largo plazo.

1. T4.1: Mecánica de I+D: Endpoint POST `/api/decisions/rnd`. La inversión se acumula; al alcanzar un umbral, `techLevel` aumenta +1.

2. T4.2: Conectar `ethicsIndex` al motor de mercado. El componente `Puntuación_Etica` en el SC solo será mayor que 0 para plazas "sensibles a la reputación" como Veridia.
3. T4.3: Integrar `ethicsIndex` y `techLevel` en la fórmula final del Winner Scorecard (WSC): $WSC = 0.4 * Utilidad_Acumulada + 0.3 * Participación_Mercado + 0.2 * Ética + 0.1 * TechLevel$.

FASE 5: COMPETENCIA DINÁMICA - v3.0 (Sprint 13+)

Objetivo: Implementar competencia adaptativa.

1. T5.1: Diseñar `BotProfileSchema` (agresivo, conservador, imitador).
2. T5.2: Servicio `BotAIService` que, para cada bot, analice decisiones de jugadores humanos en la ronda anterior y genere sus propias decisiones (ej. bot "imitador" baja un 5% su precio si un humano líder bajó el suyo).
3. T5.3: Integrar bots como `companies` en el motor de mercado.

4. ARQUITECTURA TÉCNICA Y ESQUEMAS

4.1. Esquemas de Base de Datos Críticos (Extracto)

`javascript`

```
// GameSettings Schema (Configuración por el Admin)
const GameSettingsSchema = new mongoose.Schema({
  totalProductionCapacity: { type: Number, default: 6000, min: 1000 },
  obsolescencePenaltyRate: { type: Number, default: 10, min: 0, max: 100 }, // Porcentaje
  initialCompanyCash: { type: mongoose.Decimal128, default: 500000.00 },
  maxRounds: { type: Number, default: 10 },
  rawMaterialSuppliers: {
    local: { costMultiplier: { type: Number, default: 1.2 }, ethicsBonus: { type: Number, default: 5 } },
    imported: { costMultiplier: { type: Number, default: 1.0 }, ethicsBonus: { type: Number, default: 0 } }
  }
});
```

```
// Subdocumento para Lotes de Inventario (en CompanySchema)
const inventoryLotSchema = {
  productLineId: { type: mongoose.Schema.Types.ObjectId, ref: 'Product', required: true },
  market: { type: String, enum: ['Novaterra', 'Solís', 'Veridia', 'Aurínea'], required: true },
  units: { type: Number, required: true, min: 0 },
  unitCost: { type: mongoose.Decimal128, required: true }, // Costo total de producción por unidad
  ageInRounds: { type: Number, default: 0, min: 0 } // Incrementa cada ronda que no se vende
};


```

4.2. Flujo de Procesamiento de Ronda (Pseudocódigo)

javascript

```
async function processGameRound(gameId, roundNumber) {
  // 1. ASIGNAR CAPACIDAD
  const activeCompanies = await Company.find({ gameId, isBankrupt: false });
  const quotaPerCompany = GAME_SETTINGS.totalProductionCapacity /
    activeCompanies.length;
  activeCompanies.forEach(c => { c.productionQuota = quotaPerCompany;
  c.save(); });

  // 2. PROCESAR LLEGADAS (MP y PT)
  await processArrivals(activeCompanies); // Disminuye roundsUntilArrival,
  mueve a inventario.

  // 3. VALIDAR Y EJECUTAR DECISIONES (en orden estricto)
  const allDecisions = await Decision.find({ gameId, round: roundNumber })
    .populate('companyId');
  for (const decision of allDecisions) {
    await validateProductionQuota(decision,
    decision.companyId.productionQuota);
    await processProcurement(decision); // Crea lotes MP en tránsito
    await processProduction(decision); // Consume MP, crea PT en factoryStock
    await processLogistics(decision); // Mueve PT de factoryStock a inTransit
  }
}
```

```

// 4. EJECUTAR MOTOR DE MERCADO POR PLAZA Y GAMA
const markets = await Market.find({});
const products = await Product.find({});
for (const market of markets) {
  for (const product of products) {
    const marketResult = await MarketEngineV2.calculateSales(market,
product, activeCompanies);
    await recordSales(marketResult); // Actualiza inventario, calcula
ingresos
  }
}

// 5. APLICAR OBSOLESCENCIA Y CIERRE FINANCIERO
for (const company of activeCompanies) {
  const obsolescenceCost = await
ObsolescenceService.calculateForCompany(company,
GAME_SETTINGS.obsolescencePenaltyRate);
  await AccountingService.closeRound(company, roundNumber,
obsolescenceCost);
  company.cash -= obsolescenceCost;
  await company.save();
}

// 6. NOTIFICAR CAMBIOS VÍA WEBSOCKET
io.to(gameId).emit('roundProcessed', { round: roundNumber });
}

```

5. INTERFACES DE USUARIO PRINCIPALES

5.1. Formulario de Decisiones del Estudiante (Vista Unificada)

- Sección 1: Producción
 - a Entrada numérica para cada Gama (Alta, Media, Básica).
 - b Indicador en tiempo real: "Cuota: 1,500 / Usado: 900 / Restante: 600".
- Sección 2: Compras de MP

- Tabla 3x2 (Filas: Alfa, Beta, Omega. Columnas: Unidades, Proveedor [Dropdown Local/Importado]).
- Tooltip: "Local: +20% costo, +5 ética, llega en 1 ronda. Importado: costo base, llega en 2".
- Sección 3: Logística
 - Tabla 3x4x2 (Productos x Plazas x [Unidades a Enviar, Método (Dropdown Aéreo/Terrestre)])
- Sección 4: Estrategia Comercial
 - Matriz de Precios 4x3 (Plazas x Gamas). Input numérico por celda.
 - Cuatro inputs de Marketing (uno por plaza).
 - Input único de Inversión en I+D.
- Sección 5: Herramientas
 - Checkbox "Comprar Estudio de Mercado" (\$15,000).
 - Campo opcional "Proyección de Ventas Totales".

5.2. Panel de Administración del Docente

- Pestaña 1: Configuración. Formulario para `GameSettings` (capacidad total, tasa de obsolescencia, etc.).
- Pestaña 2: Monitoreo. Tabla en tiempo real con columnas: Equipo, Cash, Útil. Neta (Ronda), Útil. Acumulada, Ética, Tech Level. Con botón "Ver Estrategia" por fila.
- Pestaña 3: Control. Botones "Procesar Ronda", "Pausar Juego", "Forzar Finalización", "Reiniciar Ronda" (con confirmación).
- Pestaña 4: Reportes. Gráficos de tendencia (Utilidad por equipo, Participación de mercado por plaza).

6. CONSIDERACIONES FINALES Y DEPENDENCIAS

1. Riesgo Técnico Principal: La implementación del `MarketEngineV2` (T1.3) es la más compleja. Debe desarrollarse con pruebas unitarias exhaustivas para cada paso del algoritmo, utilizando datos de mock.
2. Orden de Implementación: Es crítico seguir la secuencia de fases. No se puede implementar la Fase 2 (Finanzas) sin que la Fase 1 esté completa, ya que los servicios de contabilidad dependen de los nuevos modelos de datos y costos.

3. Configuración por Admin: Todos los valores numéricos mencionados (costos base, pesos del SC, porcentaje de eventos) deben ser parámetros en `GameSettingsSchema`, no constantes en el código, para máxima flexibilidad pedagógica.
4. Migración de Datos: Para partidas existentes de la v1.3, se necesitará un script de migración que asigne un `productLine` único ("Estándar") a todos los registros de inventario y decisiones antiguas, y calcule un `ethicsIndex` inicial.

Esta especificación proporciona una hoja de ruta técnica completa y detallada para transformar el simulador ElectroNova en la herramienta educativa de alto impacto que se concibió originalmente, manteniendo la viabilidad del desarrollo incremental.