🔄 Plan de Refactorización - Sistema Numérico

Objetivo: Migrar de energía física a energía numérica

Fecha inicio: 2025-01-31
Estimación: 7-14 días
Estado: ✅ Completado

🎯 Resumen Ejecutivo

Problema

El sistema actual usa `energy_pulse.tscn` como nodos 3D que se mueven físicamente. Esto causa:

- Bugs de sincronización (pulsos persisten aunque emisor desaparezca)
- Difícil escalabilidad (lag con 100+ pulsos)
- Lógica acoplada a visuales

Solución

Crear sistema numérico donde la energía son **datos**, no objetos físicos.

Beneficios:

- ✅ Sin bugs de física
- ✅ Determinista y predecible
- ✅ Fácil de debuggear
- ✅ Escala infinitamente

📊 Fases del Plan

✅ Fase 0: Preparación (COMPLETADA - 31 Ene)

- [x] Configurar GitHub
- [x] Crear estructura `docs/`
- [x] Documentar estado actual
- [x] Backup del proyecto

📋 Fase 1: Crear Managers Base (Días 1-3)

Día 1: GridManager

```
# scripts/managers/grid_manager.gd
class_name GridManager extends Node

const GRID_SIZE = 1.0
var occupied_cells = {}  # Vector2i → Building

func register_building(pos: Vector2i, building: Building)
func unregister_building(pos: Vector2i)
```

```
func is_cell_occupied(pos: Vector2i) -> bool
func get_building_at(pos: Vector2i) -> Building
```

Tareas:

- [ ] Crear archivo `grid_manager.gd`
- [ ] Implementar funciones básicas
- [ ] Añadir como Autoload en project.godot
- [ ] Test manual (colocar/quitar edificios)

Día 2: EnergyManager (Versión Mínima)

```
# scripts/managers/energy_manager.gd
class_name EnergyManager extends Node

var energy_flows: Array[EnergyFlow] = []

func register_flow(from: Building, to: Building, amount: float)
func unregister_flow(flow: EnergyFlow)
func _process(delta):
    for flow in energy_flows:
        flow.update(delta)
```

Tareas:

- [ ] Crear archivo `energy_manager.gd`
- [ ] Crear clase `EnergyFlow` (RefCounted)
- [ ] Añadir como Autoload
- [ ] Test con 1 siphon → 1 compressor

Día 3: BuildingManager

```
# scripts/managers/building_manager.gd
class_name BuildingManager extends Node

var active_buildings: Array[Building] = []

func register_building(building: Building)
func unregister_building(building: Building)
func get_buildings_in_radius(pos: Vector3, radius: float) -> Array
```

Tareas:

- [ ] Crear archivo `building_manager.gd`
- [ ] Implementar registro/desregistro
- [ ] Modificar edificios existentes para usar manager
- [ ] Añadir como Autoload

🔧 Fase 2: Migrar Edificios (Días 4-8)

Día 4-5: Refactorizar Siphon

Antes:

```
func spawn_pulse():
    var pulse = PULSE_SCENE.instantiate()
    add_child(pulse)
```

Después:

```
func _ready():
    super()
    BuildingManager.register_building(self)
    start_energy_production()

func start_energy_production():
    production_timer.timeout.connect(_on_produce)
    production_timer.start(1.0)

func _on_produce():
    var targets = find_connected_buildings()
    for target in targets:
        EnergyManager.register_flow(self, target, 10.0)
```

Tareas:

- [x] Modificar `siphon_logic.gd`
- [x] Eliminar instanciación de `energy_pulse.tscn` (siphon → compressor)
- [x] Usar `EnergyManager` para flujos
- [x] Mantener haz visual
- [x] Test funcionamiento

Día 6: Refactorizar Compressor

```
func receive_energy(amount: float):
    energy_accumulated += amount
    if energy_accumulated >= 10.0:
        energy_accumulated -= 10.0
        produce_compressed_energy()

func produce_compressed_energy():
    var targets = find_connected_buildings()
    for target in targets:
        EnergyManager.register_flow(self, target, 1.0)
```

Tareas:

- [x] Modificar `compressor.gd`

- [ ] Implementar acumulación numérica
- [ ] Conectar con `EnergyManager`
- [ ] Test cadena: Siphon → Compressor → Merger

Día 7: Refactorizar Prism

```
func receive_energy_beam(from: Building):
    var reflected_target = calculate_reflection(from)
    if reflected_target:
        EnergyManager.register_flow(self, reflected_target, from.energy_amount)
```

Tareas:

- [ ] Modificar `prism_logic.gd`
- [ ] Mantener lógica de reflexión
- [ ] Actualizar para usar `EnergyManager` (recibir_energia_numerica)
- [ ] Test con rotaciones

Día 8: Refactorizar Merger

```
var input_flows: Array[EnergyFlow] = []

func receive_energy(amount: float, source: Building):
    energy_from_sources[source] = amount
    check_merge_condition()

func check_merge_condition():
    if energy_from_sources.size() >= 2:
        var total = sum_energies()
        produce_merged(total)
```

Tareas:

- [ ] Modificar `merger.gd`
- [ ] Manejar múltiples inputs (recibir_energia_numerica)
- [ ] Output quarks → EnergyManager (Constructor recibe recibir_energia_numerica)
- [ ] Test fusión correcta

Cadena Merger → Constructor (Quarks)

Tareas:

- [ ] Constructor: recibir_energia_numerica para Up-Quark / Down-Quark
- [ ] Merger: emitir_producto usa EnergyManager

God Siphon

Tareas:

- [x] Migrar disparar() a EnergyManager
- [x] Eliminar instanciación de energy_pulse.tscn

🎨 Fase 3: Visuales Opcionales (Días 9-10)

Objetivo: Mantener feedback visual SIN afectar lógica

```
# scripts/visual/pulse_visual.gd (NUEVO)
class_name PulseVisual extends Node3D

var from_pos: Vector3
var to_pos: Vector3
var duration: float = 1.0
var timer: float = 0.0

func _process(delta):
    timer += delta
    var progress = timer / duration
    global_position = from_pos.lerp(to_pos, progress)
    if progress >= 1.0:
        queue_free()
```

En EnergyManager:

```
signal energy_transferred(from: Building, to: Building, amount: float)

func _on_flow_complete(flow: EnergyFlow):
    emit_signal("energy_transferred", flow.from, flow.to, flow.amount)
    # Algún VisualManager crea PulseVisual opcional
```

Tareas:

- [x] Crear `PulseVisual` simple
- [x] Conectar señales de `EnergyManager` (energy_transferred)
- [x] Spawn PulseVisual opcional en register_flow
- [x] Test que visuales NO afectan lógica

✅ Fase 4: Validación y Cleanup (Días 11-14)

Día 11: Testing Exhaustivo

- [x] Test cadena completa: Siphon → Compressor → Merger → Constructor
- [x] Test rotación de edificios (pulsos se destruyen al rotar origen)
- [x] Test destrucción de edificios (limpiar flujos)
- [ ] Test con 50+ edificios (performance)

Día 12: Cleanup de Código Viejo

- [x] Eliminar/deprecar `energy_pulse.tscn` (prisma ya no lo usa)
- [x] Eliminar código comentado antiguo

- [ ] Actualizar todos los `# TODO` relacionados
- [ ] Limpiar preloads no usados (construction_manager: solo god_siphon_escena)

Día 13: Documentación Final

- [ ] Actualizar `PROJECT_STATE.md`
- [ ] Crear `ENERGY_SYSTEM.md` con sistema final
- [ ] Documentar API de managers (`docs/API_MANAGERS.md`)
- [ ] Escribir lecciones aprendidas

Día 14: Commit y Celebración

```
git add .
git commit -m "Refactorización completa: sistema energía numérico"
git push
```

- [ ] Marcar en GitHub como versión v0.4-alpha (tag)
- [ ] Planificar siguiente feature (electrones, protones...)

🚨 Criterios de Éxito

El refactor es exitoso si:

- ✅ Sistema corre sin nodos de energía física
- ✅ Rotación de edificios actualiza flujos correctamente
- ✅ Destrucción de edificios limpia todos los flujos asociados
- ✅ Performance estable con 100+ conexiones simultáneas
- ✅ Los bugs actuales desaparecen

⚠️ Reglas Durante el Refactor

1. **NO añadir features nuevas** (electrones, átomos, etc.)
2. **Commit frecuente** (mínimo 1/día)
3. **Si algo funciona → commit antes de tocar otra cosa**
4. **Test manual después de cada cambio mayor**
5. **Si te atascas >2h → pedir ayuda**

🔄 Estado de Fases

- [ ] Fase 0: Preparación
- [ ] Fase 1: Managers (3/3 días) ✓
- [ ] Fase 2: Migración edificios (Siphon, Compressor, Prism, Merger) ✓
- [ ] Fase 3: Visuales (2/2 días) ✓
- [ ] Fase 4: Validación y Cleanup ✓

Progreso total: ~14/14 días

📝 Notas

- **Protocolo de archivos:** ver `docs/FILE_PROTOCOL.md` (snake_case, scripts en scripts/; deprecated eliminado en ROADMAP 3.2)
- Este plan es flexible, ajustar según necesidad
- Priorizar funcionalidad sobre visuales
- Documentar decisiones importantes
- Hacer backup antes de cambios grandes

📚 Lecciones Aprendidas

Qué funcionó bien:

- Separar lógica (EnergyFlow) de visual (PulseVisual): los visuales se pueden desactivar sin romper nada
- Autoloads centralizados: EnergyManager, GridManager, BuildingManager simplifican el código
- Método único `recibir_energia_numerica()` : todos los receptores implementan la misma API
- Documentar mientras se avanza: ENERGY_SYSTEM.md y API_MANAGERS.md ayudan a entender luego

Qué haríamos distinto:

- Validar más temprano que HUD/UI no bloquea input (mouse_filter en inventario)
- Herramientas auxiliares (ej. generador F9): probar flujo completo antes de integrar
- Unificar fuentes de escenas: RECETAS en GameConstants vs menu_data en hud_manager generó bugs (Compresor T2)

Próximo feature sugerido: Electrones/protones (siguiente escalón en la cadena energía → materia)

🎯 Siguiente Feature (v0.5)

Ver **`docs/FUTURE_PLAN.md`** para el plan detallado.

Opción	Esfuerzo	Descripción
Electrones	Medio	Nuevo recurso/bloque que consume quarks
Protones/Neutrones	Alto	Fusión Up/Down quarks → partículas
Pulido UX	Bajo	Tutorial, feedback visual, mejora menús
Bugs menores	Bajo	Haces prismas, salidas merger