MVCGameEngine — REFACTOR_V1

Informe unificado: situación base, problemas de fronteras y soluciones

Estado: REFACTOR_V1 · Documento de arquitectura consolidado listo para GitHub

---

ÍNDICE

ABSTRACT

MVCGameEngine es un motor educativo y modular que permite crear arcades muy distintos sin tocar el core MVC. El core proporciona infraestructura (tiempo, física, eventos, ejecución), mientras que una serie de módulos base configurables (World*, LevelGenerator, IAGenerator, ActionsGenerator) permiten definir la experiencia de juego.

El engine funciona correctamente, pero presenta problemas estructurales que dificultan:

• entender dónde modificar un parámetro para que tenga efecto,
• evitar overrides silenciosos entre módulos,
• y ofrecer una buena experiencia en las primeras horas de uso.

Este informe:

• fija el estado real de partida (qué es cada módulo hoy),
• identifica las roturas de frontera que causan confusión,
• y propone soluciones conceptuales que:
• aclaran responsabilidades,
• reducen parámetros dispersos,
• mantienen el core como infraestructura,
• y permiten una alta variación de arcades sin tocarlo.

---

2. INFORME DE SITUACIÓN BASE

2.1 Filosofía general del engine

• Arquitectura MVC, sin game loop global.
• Cada DynamicBody ejecuta su propio tick en su propio hilo.
• El core:
• calcula física,
• detecta eventos,
• ejecuta acciones,
• mantiene coherencia temporal (dt).
• Los módulos base están fuera del core y son intercambiables para crear juegos distintos.

2.2 Módulos base existentes y su rol actual

World* (WorldDefinition + Providers + Assets)

Define:

• qué elementos existen,
• qué assets se usan,
• cómo se ven los objetos.

Incluye:

• definición de items,
• armas,
• emitters,
• fondos.

Actualmente mezcla:

• definición visual,
• parámetros físicos,
• y parte de la lógica de gameplay.

LevelGenerator

• Crea la escena inicial.
• Coloca elementos “estáticos”.
• Hoy aplica el nivel en el constructor.

Objetivo gamer explícito:

• gestionar escenas estáticas y cambios de nivel.

IAGenerator

• Genera la parte dinámica del mundo.
• Decide:
• qué aparece,
• cuándo aparece,

• con qué ritmo.

Actualmente:

• pisa tamaños y masas,
• hace bootstrap del player,
• y se ve afectado indirectamente por la semántica de MOVE.

ActionsGenerator

• Recibe eventos ricos.
• Decide qué acciones se disparan.
• Es el ruleset del juego.

Actualmente sufre confusión porque:

• algunas acciones parecen necesarias para que el mundo avance.

Core (Model + Controller)

Infraestructura pura:

• física,
• tiempo,
• eventos,
• ejecución de acciones.

Implementa correctamente:

• tick por body,
• commit de física,
• NO_MOVE con timestamp correcto.

Pero con semántica difícil de entender desde fuera.

2.3 Estado del Main

El Main actual:

• contiene muchos parámetros de diseño (tamaños, masas, delays),
• actúa como pseudo–archivo de configuración del juego,
• y expone al usuario demasiadas decisiones demasiado pronto.

Esto es un síntoma, no el problema raíz.

---

3. PROBLEMAS DE FRONTERAS IDENTIFICADOS

3.1 Doble fuente de verdad

• Tamaños y masas definidos en World* y recalculados en IA.
• Resultado: cambiar un valor no siempre tiene efecto.

3.2 Variabilidad visual limitada

- Pocos assets → miles de instancias.
- Sin rangos bien definidos, el arcade se vuelve repetitivo.

3.3 Movimiento acoplado a acciones

- El commit de movimiento depende implícitamente de acciones (MOVE).
- Spawn o ticks sin eventos pueden congelar bodies.
- El comportamiento es correcto, pero opaco.

3.4 NO_MOVE malinterpretado

- Conceptualmente podría confundirse con “pausar el tiempo”.
- En realidad congela posición pero actualiza timestamp (correcto).

3.5 IA y Rules lidiando con infraestructura

IA y ActionsGenerator no deberían:

- preocuparse de dt,
- ni de timestamps,
- ni de “mantener el mundo en marcha”.

3.6 Main sobrecargado

- Muchos parámetros acaban en el punto de entrada.
- El usuario no sabe qué es esencial y qué es detalle.

---

4. PROPUESTAS DE SOLUCIÓN (CONSOLIDADAS)

4.1 World*: definición clara + variación explícita

4.1.1 Separar ItemDTO y PrototypeItemDTO

ItemDTO (determinista)

- Objetos del nivel:
- tamaño fijo,
- posición fija,
- apariencia fija.
- Consumidos por LevelGenerator.

PrototypeItemDTO (generativo)

- Objetos spawnables:
- asteroides,
- debris,
- powerups dinámicos.
- Contienen:
- rangos de tamaño, orientación, rotación,

• referencia a asset,
• política de masa.
• Consumidos por IAGenerator.

4.1.2 Rangos como contrato, no como valor final

• World*:
• no fija tamaños finales,
• fija dominios válidos.
• IA:
• muestrea dentro de esos rangos.

Una única fuente de verdad.

4.1.3 Densidad definida en assets

Cada asset puede definir su densidad (material).

Ejemplo: asteroides férreos más densos.

La masa se calcula siempre como:

```
mass = density * size^p * massScale
```

Beneficios:

• coherencia física,
• variedad real,
• ningún minMass/maxMass en IA ni en el Main.

4.2 LevelGenerator: escena estática y progresión

• Instala ItemDTO.
• Gestiona cambios de nivel.

No:

• genera dinámicos,
• define ritmo,
• toca reglas.

4.3 IAGenerator: dinámica pura

• Decide cuándo y cuántos.
• Instancia prototipos.
• No redefine apariencia ni física base.

IAConfig se reduce a:

• spawn rates,
• delays,

- patrones.

IA no lidia con timestamps ni movimiento base.

4.4 ActionsGenerator: reglas, no infraestructura

- Las acciones expresan consecuencias:
- spawn,
- die,
- overrides explícitos.

No son responsables de:

- hacer avanzar el mundo,
- mantener bodies activos.

4.5 Core: commit obligatorio y explícito

4.5.1 Contrato definitivo del tick por body

En cada tick:

1. Se calcula dt
2. Física propone nuevos valores
3. Se detectan eventos
4. Rules generan acciones
5. El core resuelve una **MovementDirective**:
6. DEFAULT_COMMIT
7. FREEZE (NO_MOVE)
8. OVERRIDE
9. Siempre se commitea
10. Siempre se actualiza el timestamp
11. El tiempo nunca se congela

4.5.2 NO_MOVE correcto y centralizado

- NO_MOVE:
- congela posición/velocidad,
- actualiza timestamp.

IA y rules no se ocupan de esto. El core es el único responsable.

4.5.3 MOVE desaparece como acción pública

- MOVE deja de ser acción necesaria.
- El commit es infraestructura.

En el futuro:

- el “movimiento” se modela como evento o métrica (energía, desgaste).

4.6 Simplificación radical del Main

Gracias a:

- prototipos,
- densidad en assets,
- masa derivada,
- contratos claros,

el Main:

- deja de contener diseño,
- solo orquesta módulos.

Esto mejora enormemente el onboarding.

---

5. GRID DE CROSS-REFERENCE

Problema	Solución
Tamaños pisados entre módulos	PrototypeItemDTO con rangos
Juego repetitivo	Rangos + variación continua
Masas arbitrarias	Densidad en assets + masa derivada
Main sobrecargado	Diseño movido a World
Bodies se congelan sin MOVE	Commit obligatorio
Spawn congela emisores	Movimiento independiente de acciones
Rebotes inconsistentes	MovementDirective
NO_MOVE confuso	Semántica clara + core
IA tocando infraestructura	Commit y timestamp solo en core
Reglas difíciles de razonar	Acciones = consecuencias

---

6. CONCLUSIÓN

MVCGameEngine no necesitaba “más features”, sino fronteras claras.

Las soluciones propuestas:

- respetan el diseño original,
- mantienen el core como infraestructura,
- permiten una gran variación de arcades,
- y convierten un comportamiento correcto pero opaco en un sistema correcto y explicable.

El resultado práctico es clave:

Cuando un desarrollador quiere cambiar algo, sabe exactamente dónde hacerlo.

Eso es lo que transforma un motor funcional en un motor realmente usable.