



Serre dei Giardini Margherita
15 ottobre 2025

Game
Design

Design Modulare

Metodo e filosofia per sopravvivere a ogni scenario

Fabio Consorti



Ministero delle Imprese
e del Made in Italy





DEMON GIRL DIGITAL



Fabio Consorti

Video game developer and founder of **Demon Girl**. He have worked on two titles: **Unit 4** (PC, Xbox, NSwitch) and **Still Time** (PSVita, PS4). After gaining experience in the industry, he went independent and created **Demoniaca - Everlasting Night** (PC, PS4/5, Xbox X/S, NSwitch)



Federico Della Bona

Federico Della Bona is a talented indie developer who co-developed **Demoniaca - Everlasting Night** with Fabio. Lifelong friends and colleagues, they have worked together on many prototypes and smaller games, including *Camouman* and *Drag'N Dungeon*.



Fabio Bocci

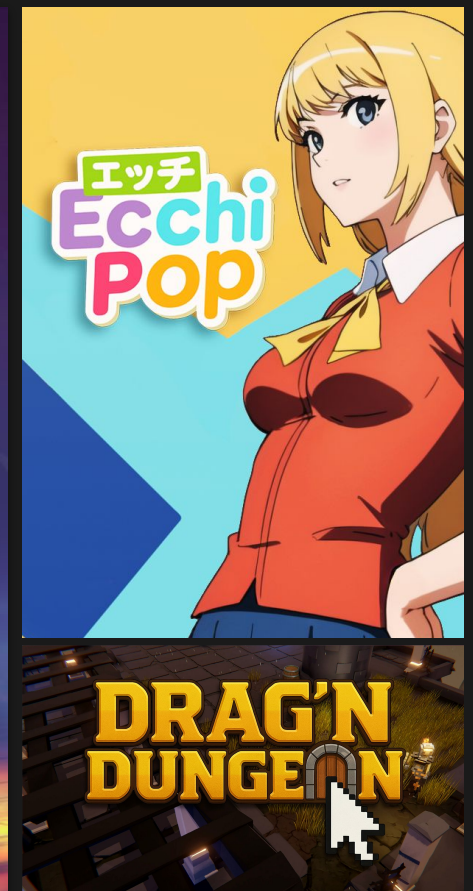
Video Game Developer with 5 years of experience

Worked for 3 years with an international team on an MMORPG, **Monumenta**, that is still active today.

I NOSTRI GIOCHI



In development



DEATH BLONDE - TRAILER

click to play



"Proun 19D" El Lissitzky, 1922

Fabio Consorti

DESIGN MODULARE

DEI SISTEMI DI GIOCO



INTRO

Un videogioco non è solo codice o arte: è un organismo che cresce, muta, e spesso... viene amputato o potenziato lungo la strada.

Pensare in modo **modulare** significa costruirlo perché possa **resistere al cambiamento**

"Il grande Drago Rosso e la donna vestita di sole" William Blake, 1805

Perché serve ai piccoli team

- Budget limitato
- Tempi stretti
- Persone che ricoprono più ruoli

→ Serve una struttura che non crolli quando qualcosa va tagliato o aggiunto.

Il design modulare ti permette di sopravvivere alle incertezze del prototipo.



"Il viandante sul mare di nebbia" Caspar David Friedrich, 1818



Filosofia modulare

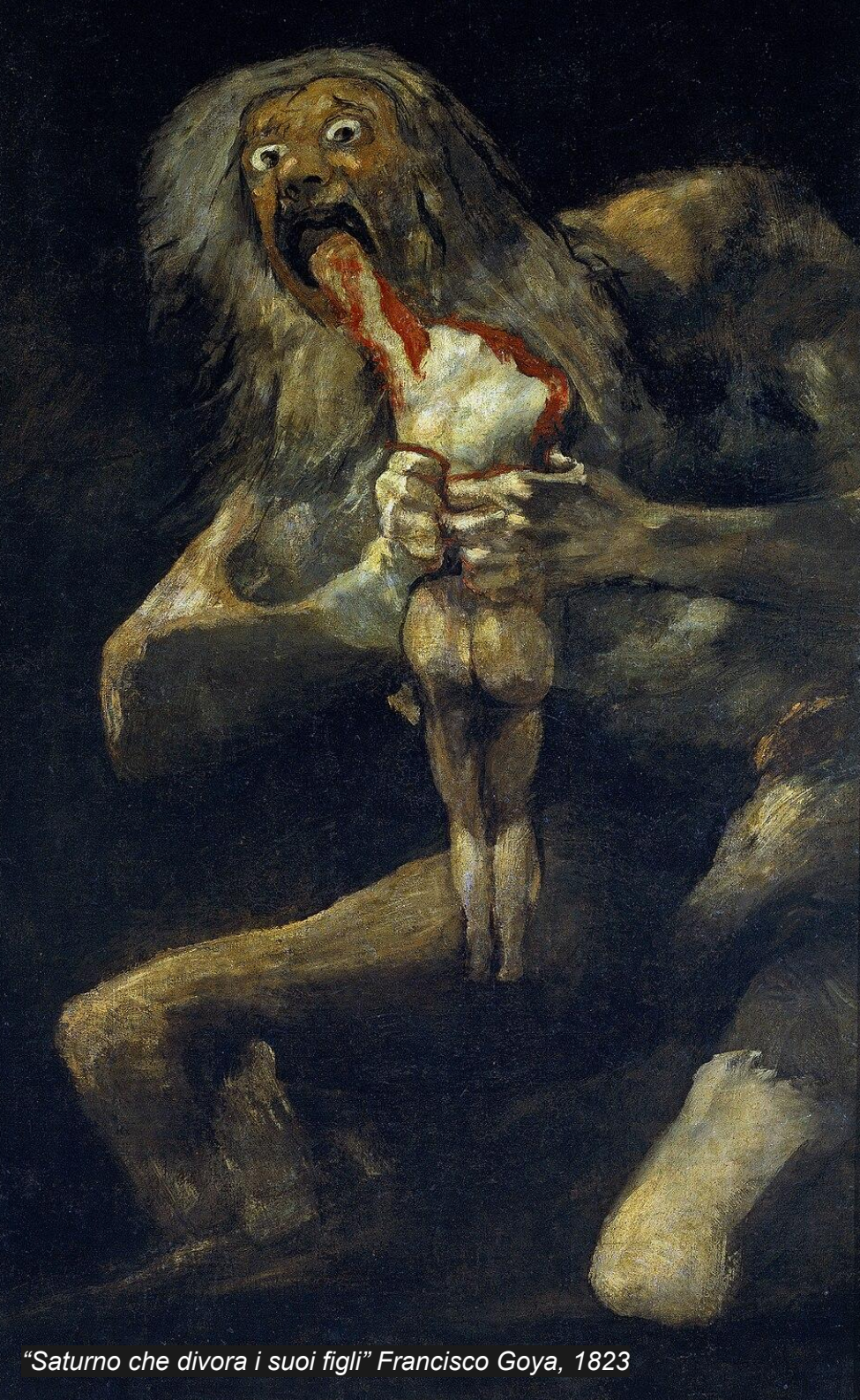
Sviluppare in modo modulare non è solo pratica: è una filosofia mentale.

Significa progettare pensando al cambiamento: confini chiari, responsabilità singola, contratti tra parti, decisioni reversibili. È scegliere la chiarezza alla furbizia, la coerenza di mondo all'accumulo caotico. Così ogni pezzo può vivere da solo, e l'insieme si adatta senza rompersi.

Il rischio del prototipo

Quando sviluppi un prototipo per proporlo a un publisher, non sai se il gioco **vivrà** o **morirà**.

Se il progetto è monolitico, ogni cambiamento diventa doloroso.
Se è modulare, può contrarsi o espandersi senza **perdere coerenza**.



"Saturno che divora i suoi figli" Francisco Goya, 1823



Concetto chiave

“Non costruire un castello: costruisci un villaggio che può diventare una grande città o un gruppo coeso di case”

Il sistema modulare nasce in scala, pronto a **crescere** o **ridursi**. Ogni modulo funziona da solo, ma insieme raccontano l'esperienza completa con coerenza.



Esempi pratici di modularità

- Armi e abilità come blocchi plug-in
- Nemici basati su comportamenti componibili
- Livelli generati da tiles e chunk riutilizzabili
- UI costruita per pannelli/layers indipendenti
- Utilizzo di prefabbricati e varianti

Vantaggi per il team

- Il codice è più leggibile
- I designer e gli artisti possono sperimentare senza paura
- Ogni membro capisce dove intervenire
- Facilita il lavoro anche a collaboratori esterni o temporanei
- Utilizzo stagno dell'IA generativa



"La scuola di Atene" Raffaello Sanzio, 1509

Il debito tecnico

Ogni scorciatoia accumula **debito tecnico**.

Il design modulare non lo elimina, ma lo rende visibile e gestibile.

Un modulo “sporco” non inquina il tutto: si può riscrivere o sostituire.

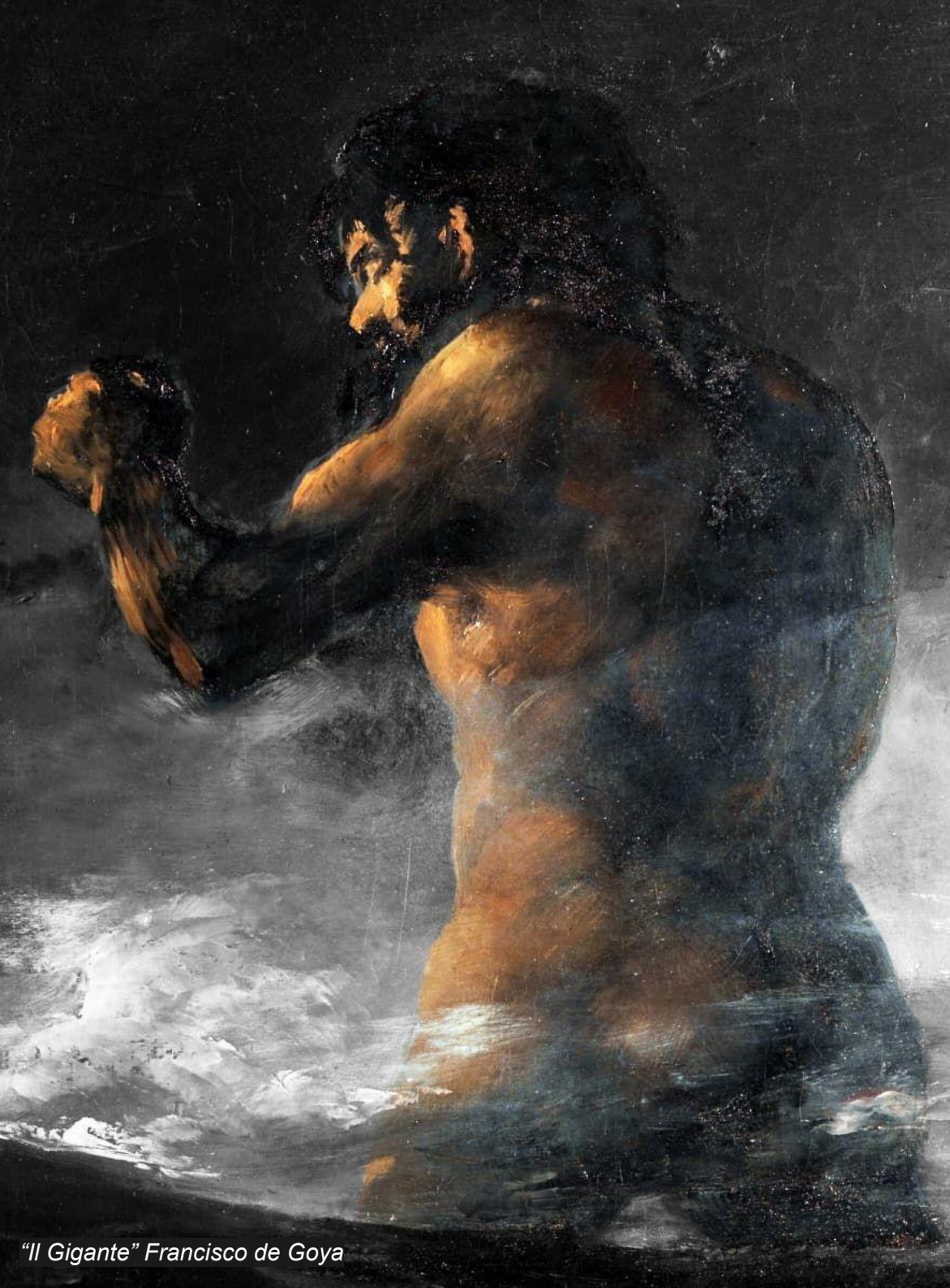


“L'isola dei morti” Arnold Böcklin

The painting 'Gli interni' by Vilhelm Hammershøi depicts a woman in a black dress with a white sash, seen from behind, standing in a room. She is positioned between a round wooden table in the foreground and a doorway in the background. The room features light-colored walls with decorative panels and a dark doorway. The lighting is soft and atmospheric, typical of Hammershøi's style.

Sistemi stagni e chiarezza

- Sai cosa entra/esce
- Puoi cambiare l'interno senza toccare l'esterno
- Un nuovo programmatore capisce la logica senza esplorare tutto
- Puoi esternalizzare il lavoro a terzi



Scalabilità: crescere o ridursi

Non puoi prevedere il futuro.

Se arriva un finanziamento, il
gioco si espande naturalmente.

Se non arriva, resta solido nella
versione minima.

In entrambi i casi, **non butti via
nulla.**



"La Città Ideale" Piero della Francesca

Filosofia produttiva

Pensa a ogni modulo come a un
“ecosistema autosufficiente”:

- Interagisce ma non dipende totalmente
- Può vivere da solo, ma funziona meglio in rete
- Documentato il “giusto” perchè esplicito, riutilizzabile, sostituibile



Ogni modulo è parte del tutto

Si può ampliare, spostare, restaurare

Protegge la coerenza interna

Si adatta al “terreno” (vincoli produttivi)

Un buon sistema non è rigido: è vivo.

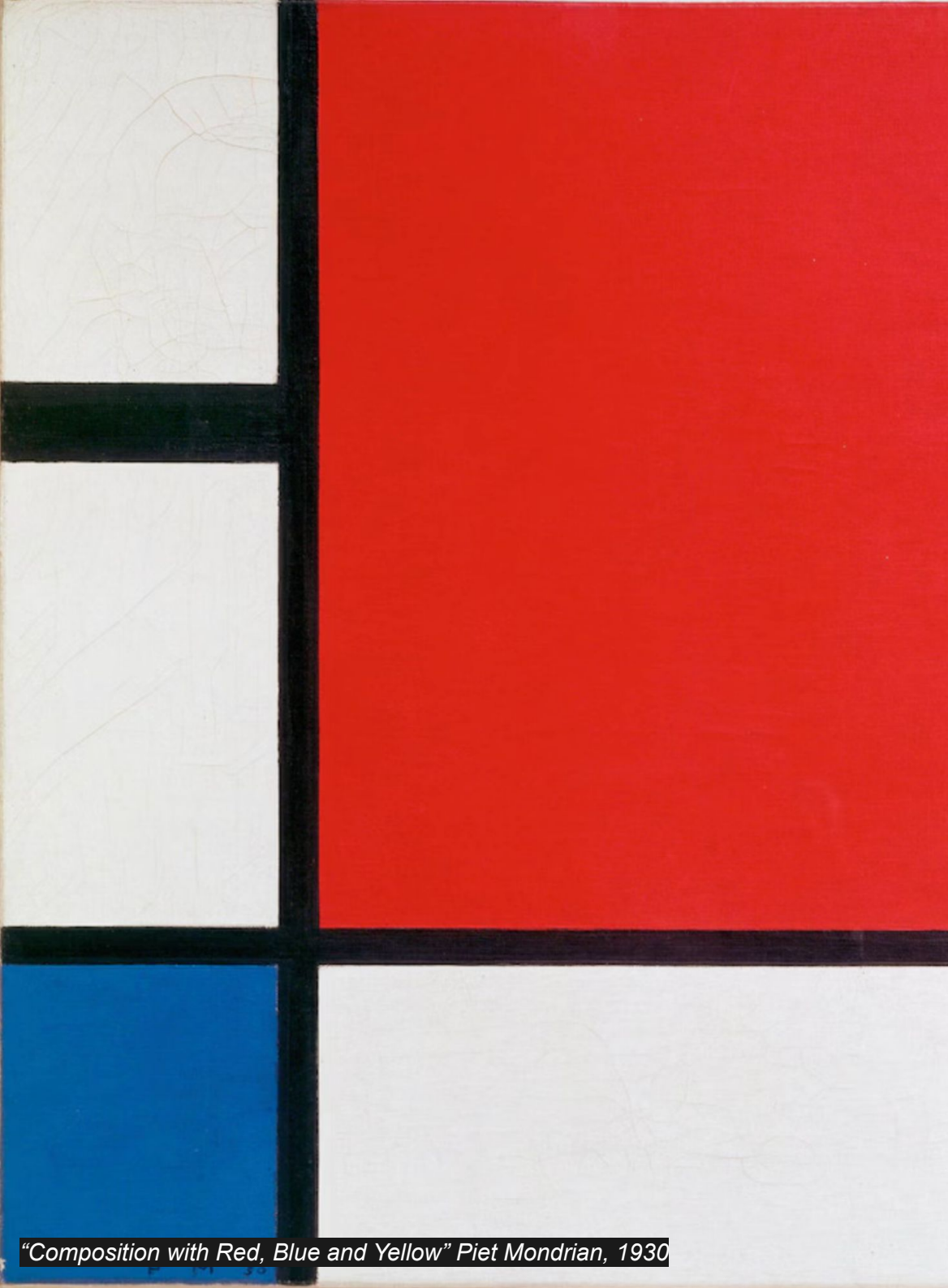
Errori comuni

- Unire troppi sistemi “a mano”
- Dipendenze invisibili
- Hack temporanei “che poi sistemiamo”
- Ambiguità tra moduli

→ Segnali che il progetto sta **perdendo forma**.



“Il sogno della ragione genera mostri” (Francisco Goya)



Cohesion First: un modulo, una ragione di esistere

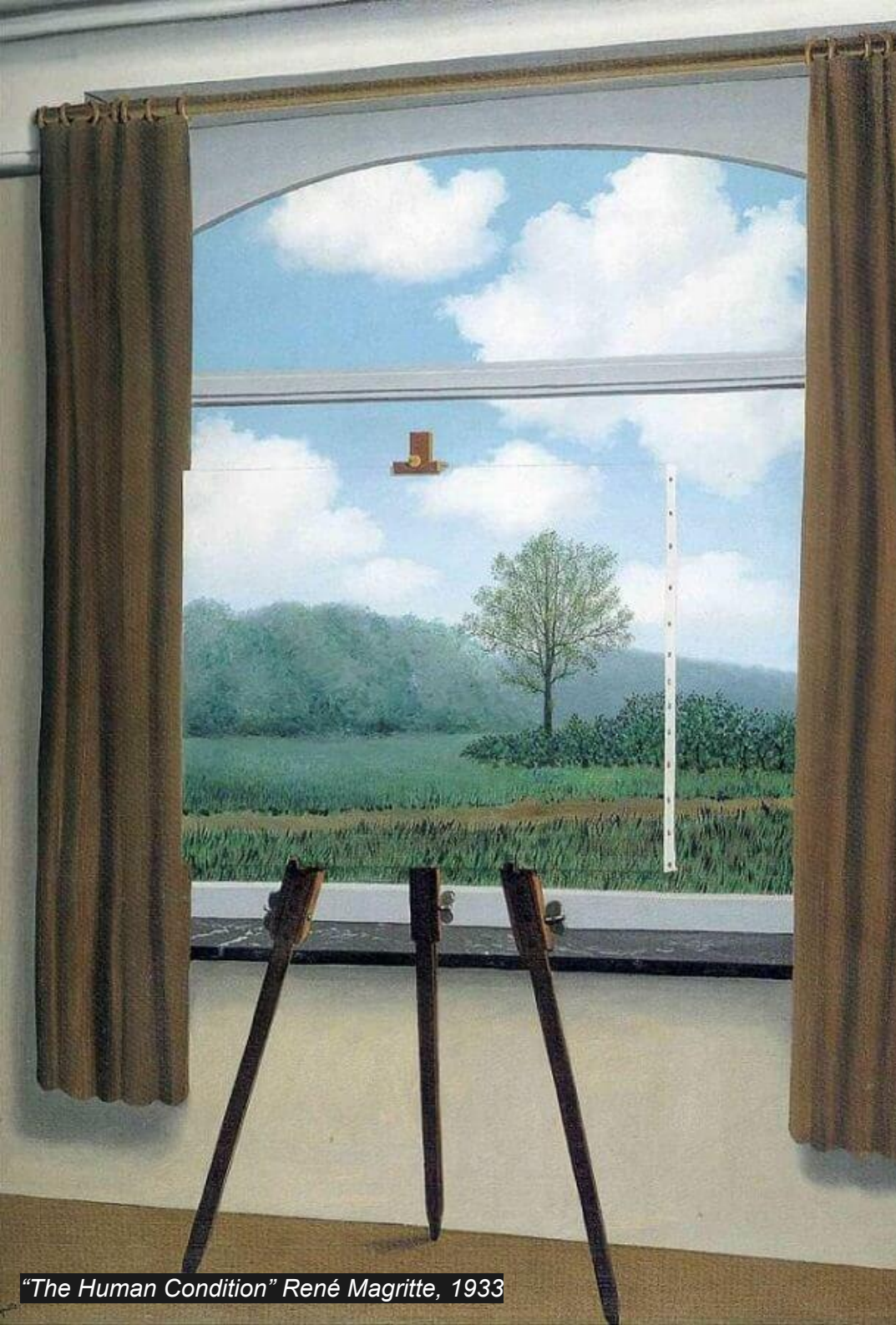
Un modulo appartiene allo stesso mondo degli altri, ma vive di una responsabilità precisa. Se il tuo modulo governa il movimento del personaggio, parla di input, velocità, attrito, salto e basta. I suoni non sono affar suo: li chiederà a chi di dovere.

Questo rende la logica **leggibile**, **testabile** e **sostituibile**. La domanda: riesci a descrivere il modulo in una sola frase senza “e anche”? Se la risposta è no stai mescolando dei comparti. Resisti alla tentazione: l'ordine oggi è libertà domani.

Tassonomia del mondo: cosa tenere insieme e cosa separare

Pensa al progetto come a un ecosistema ordinato.

- Gli Attori hanno i loro bisogni vitali
Movimento, Salute, Stato, Progressione
- i Sistemi definiscono le regole
Combattimento, IA, Progressione, Quest
- la Presentazione dà forma e sensazione
Audio, VFX, Animazione, Camera, UI
- l'Infrastruttura sostiene tutto
Salvataggi, Localizzazione, Piattaforma.



"The Human Condition" René Magritte, 1933

Tassonomia del mondo: la Regola

Accorpa solo ciò che condivide lo stesso dato centrale e lo stesso ritmo di vita:

due cose che cambiano sempre insieme possono convivere; ciò che ha tempi diversi o proprietari diversi va separato.

Così ogni pezzo resta comprensibile e non si trascina dietro dipendenze nascoste.



"Several Circles" Wassily Kandinsky, 1926

Accorpare o dividere: la decisione senza drammi

Quando sei nel dubbio, fatti tre domande.

Primo l'output è di natura diversa? Allora dividilo (il suono non è movimento).

Secondo questo pezzo ti servirà in altri attori o sistemi? Se sì, va isolato per essere riusato senza frizioni.

Terzo ha un ciclo di vita o un ritmo differente?

Se una cosa deve girare ogni frame e l'altra solo su evento, è un segnale di separazione. Se invece due parti cambiano sempre insieme, puoi accorparle senza paura.



Sky and Water I" M. C. Escher, 1938



Attrito

Un indizio che stai mescolando è quando, per testare un modulo ti servono troppi “manovre”.

I workarounds spesso sono il primo segnale di dispersione.

Significa che quel modulo è **troppo dipendente** dal vicinato.



Parlarsi senza mescolare: contratti chiari, canali puliti

I moduli **non devono conoscersi intimamente**: devono solo parlarsi bene. Il movimento emette un evento di atterraggio; l'audio lo ascolta e fa il suo mestiere; la camera reagisce; la UI si aggiorna. Nessuno infila le mani nei dati interni dell'altro se non c'è un reale motivo.

Questa **distanza salutare** evita dipendenze circolari e rende ogni pezzo sostituibile. Scrivi, per ogni modulo, un contratto: cosa entra, cosa esce, cosa garantisce e cosa succede quando qualcosa manca. È poco glamour, ma salva il progetto quando arrivano richieste a orologeria.

Esempio pratico di approccio

Osserva il quadro. Un sistema modulare funziona anche con un quadro perché è indipendente:

Ogni pedina è **muta**. Non conosce nulla dell'audio e non "fa partire" suoni.

- Un Audio Manager osserva la scena: vede cosa accade (es. "la pedina sta colpendo, ha una spada").
- Controllo: Quanti suoni ho attivi, il rumore generale della scena è tollerabile?
- Azione: approvato "colpo un con spada", riproduco un suono "colpo di spada".

Se tolgo l'Audio Manager o la Pedina, niente cambia.



"La battaglia di Alessandro a Issos" Albrecht Altdorfer, 1529



Esempio pratico di approccio

Possiamo applicare lo stesso principio anche ad il suono di un “click” di un bottone lato utente.

Il bottone è **muto**. Non conosce nulla dell’audio e non “fa partire” alcun suono.

- Monitoraggio: Un Audio Manager osserva le interazioni dell'utente.
- Controllo: Input in entrata, premuto un bottone, il bottone ha qualche eccezione?
- Azione: Approvato, riproduco il suono “bottone generico”.

Un sistema modulare può interagire con input diversi **senza interessarsi** della loro natura: un click del mouse è uguale ad un'interazione di attacco.



Quando un modulo è davvero “**FINITO**”

Finito, significa che il modulo fa una cosa sola, ha ingressi e uscite documentate, non pesca direttamente da comparti estranei (lo chiede a chi di dovere), può essere acceso o spento senza far crollare tutto o lasciare tracce (log puliti, piccoli overlay di debug, metriche)

Se per aggiungere un suono devi modificare il controllo, fermati: **stai comprimendo due mondi in uno**. Meglio un passo in più oggi, che un debito tecnico che ti presenterà il conto quando servirà scalare.

Auto- pubblicazione

Non possiamo prevedere con precisione le sorti del nostro prodotto, potresti trovarti in questa posizione: **ho un prototipo e non trovo nessuno, devo autopubblicarmi.**

La domanda è semplice e concreta: ho fatto qualcosa che riesco a pubblicare anche da solo?

Pensiero minimale: tengo solo ciò che serve a far capire e godere l'idea, tutto il resto fuori. Se resta in piedi senza aiuti esterni, allora è pubblicabile; altrimenti, va **ridotto o buttato via.**



"Abbazia nel querceto" Caspar David Friedrich



Trovo un investitore

Ho trovato un publisher/investitore: il mio progetto può espandersi per adattarsi alle esigenze di quest'ultimo?

Potrebbe non esserci più tempo per valutare se la struttura possa reggere l'aumento di scala senza snaturarsi.

Se i blocchi si aggiungono senza rifare la base, posso crescere quanto basta; se ogni aggiunta tira giù un pezzo, il sistema non è pronto per l'espansione e collassa.

Se me ne **accorgo a cose fatte**, potrei trovarmi presto in difficoltà.

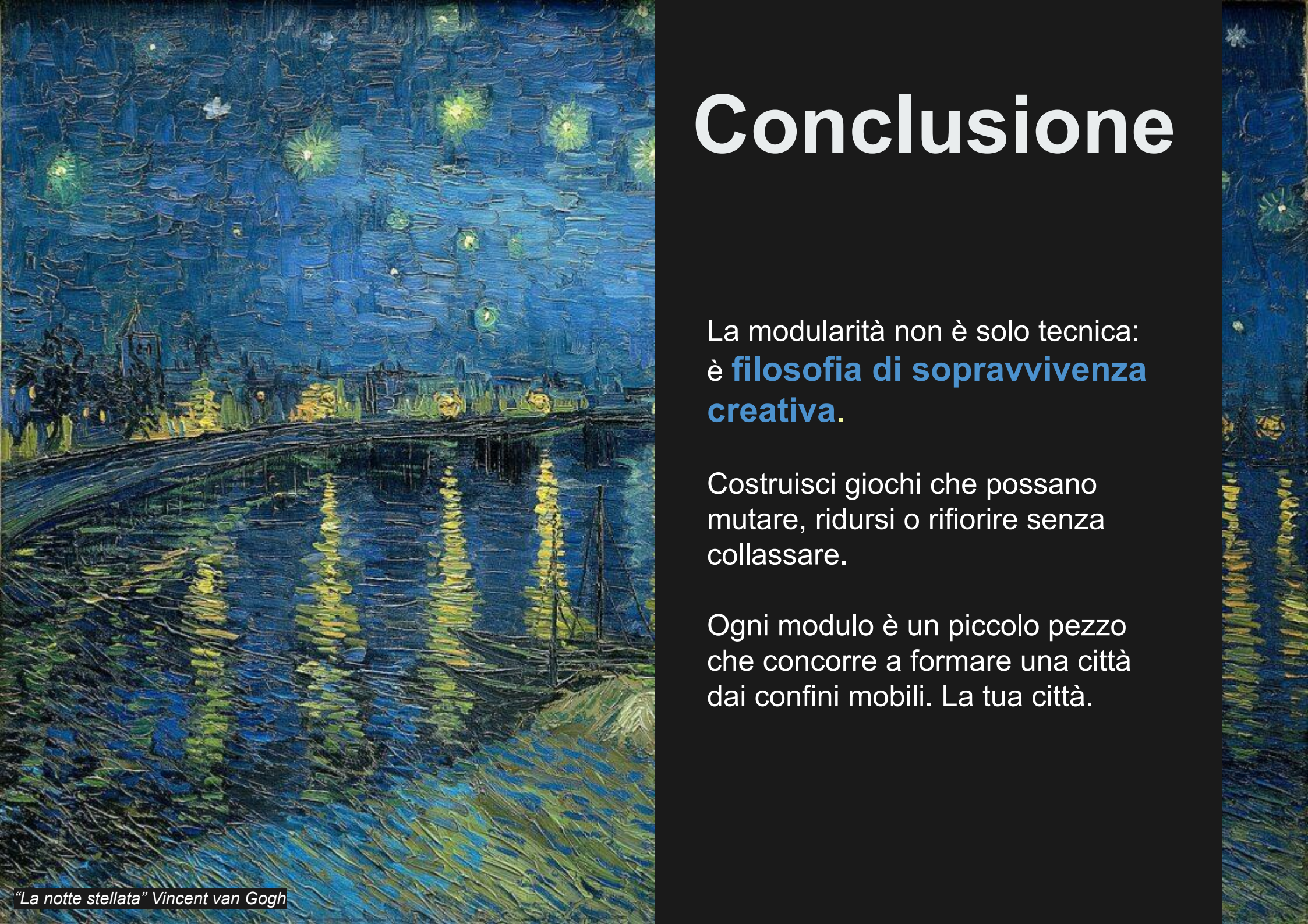


Post-lancio modulare

Una volta pubblicato, posso aggiungere facilmente qualcosa? Quante risorse servono davvero?

Se il gioco è modulare, **poche:** inserisco un pezzo, collego i punti, e vado avanti.

Questa differenza di costo e fatica può fare la differenza tra successo o **morte** del prodotto mentre e dopo l'uscita.

The background of the slide is a reproduction of the painting 'The Starry Night' by Vincent van Gogh. It depicts a night scene with a turbulent, star-filled sky in shades of blue and green, reflected in a dark body of water. A small town with a prominent church spire is visible on the horizon. The brushwork is thick and expressive, characteristic of Van Gogh's style.

Conclusione

La modularità non è solo tecnica:
è **filosofia di sopravvivenza
creativa**.

Costruisci giochi che possano
mutare, ridursi o rifiorire senza
collassare.

Ogni modulo è un piccolo pezzo
che concorre a formare una città
dai confini mobili. La tua città.



DESIGN MODULARE DEI SISTEMI DI GIOCO

"Snow Storm: Steam-Boat off a Harbour's Mouth" William Turner:

Tips and tricks

Un modulo = una frase. Se serve “e anche...”, va diviso.

Eventi > chiamate dirette. Parlatevi con segnali, non con campi interni.

Adapter ai confini. Mai conversioni dentro i moduli (UI↔Audio).

Default + eccezioni per tag. Comportamento comune, override puntuali.

Scene sandbox. Ogni modulo testabile da solo, con pochi “finti vicini”.

Feature Flags. Build diverse senza rimaneggiare codice; **kill-switch** pronto.

Scope Ladder (XS→XL). Cresci per pacchetti rilasciabili, non per riscritture.

Contratti chiari. Ingressi, Uscite, Garanzie, Guasti, Osservabilità.

Debito Controllato. Paga poco debito tecnico quando sali di scala (S→M→L).

Niente “manager di Dio”. Tanti manager piccoli, ognuno nel **suo** mondo.



Grazie per la partecipazione



Fabio Consorti

Personal

aki@akipage.com

Demon Girl Digital

info@demongirldigital.com

www.demongirldigital.com