

Engineering Gameplay Systems

Dare ordine alla complessità attraverso sistemi di gameplay modulari

Davide Mercurio



nacón
STUDIO
MILAN



Ministero delle Imprese
e del Made in Italy



ENGINEERING GAMEPLAY SYSTEMS

Dare ordine alla complessità attraverso sistemi di gameplay modulari

Davide Mercurio

Lead Gameplay Programmer @ Nacon Studio Milan

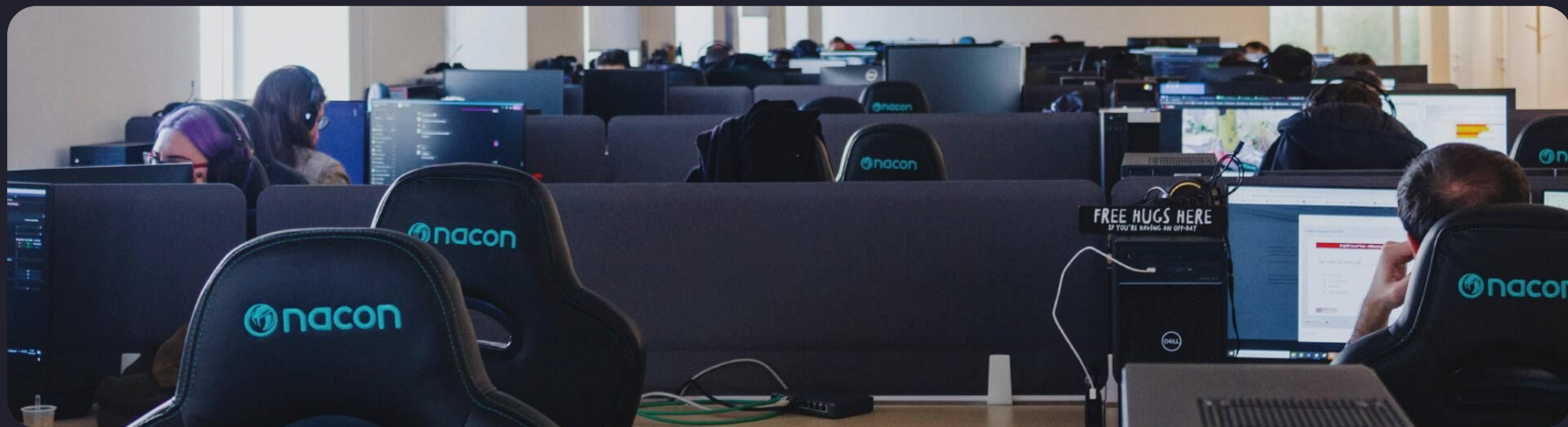
CHI SONO

- **Lead Gameplay Programmer** in **Nacon Studio Milan**
- **10 anni** nel settore Game Dev
- **7 titoli** pubblicati su PC e Consoles



CHI SIAMO

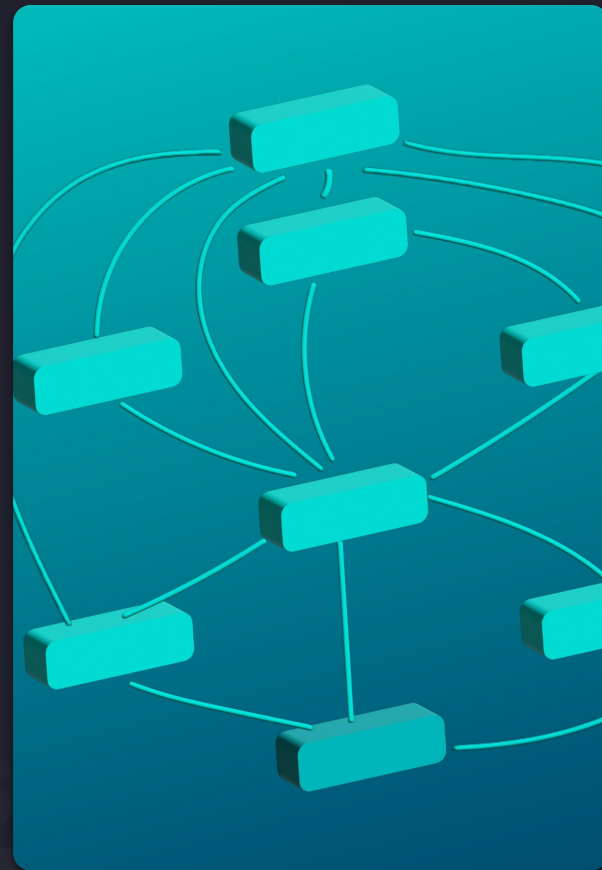
Nacon Studio Milan è parte del gruppo francese **NACON**, specializzato nella **produzione** e **pubblicazione** di videogiochi e **accessori gaming**, con una presenza globale in oltre **100 paesi**.



IL PROBLEMA

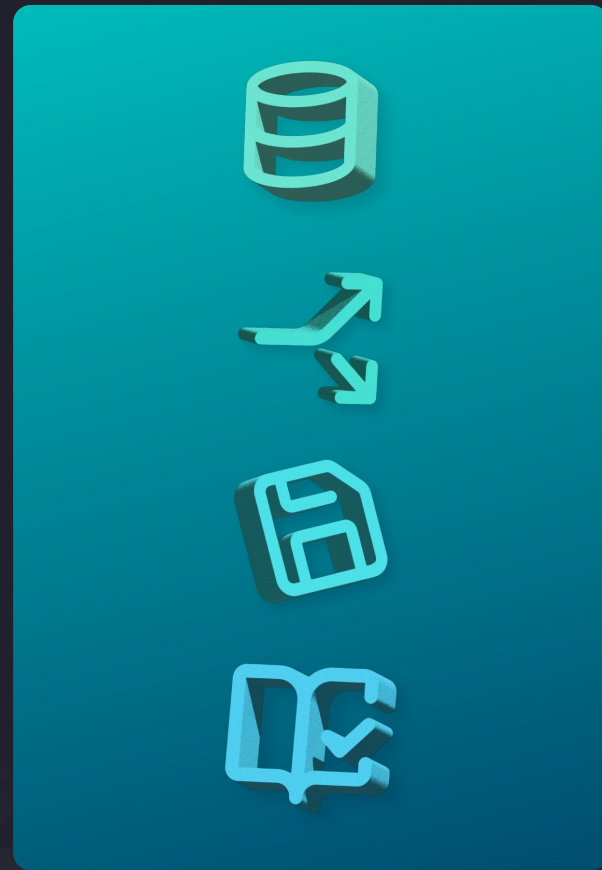
Quando i sistemi crescono, il caos cresce con loro

- In un progetto complesso i sistemi di gameplay si **moltiplicano**
- Ogni sistema tende ad avere le proprie **variabili, eventi e logiche**
- Col tempo si creano **duplicazioni, dipendenze circolari e incoerenze**
- Aggiornare o estendere diventa **difficile e rischioso**



AGENDA

- **Collection**
- **Condition**
- **Save System**
- **Conclusioni**



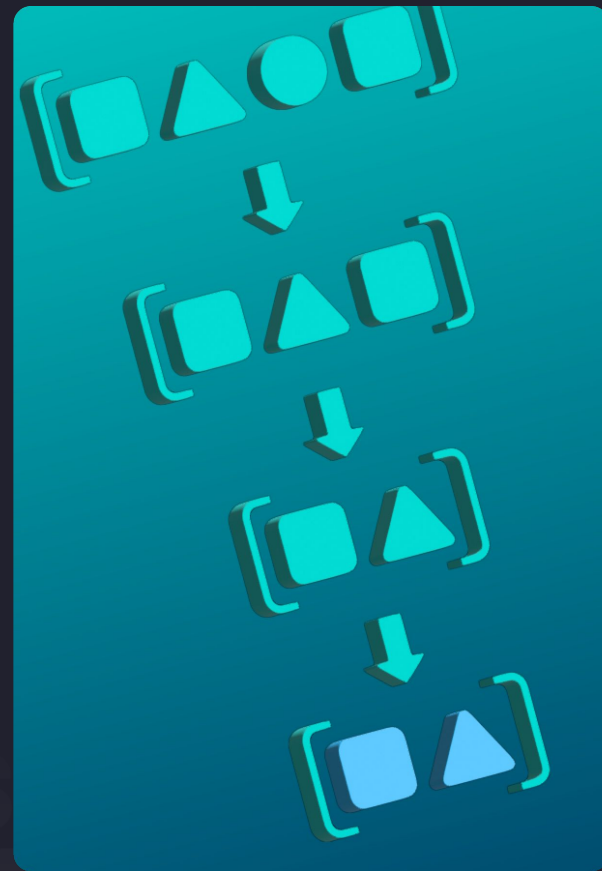
COLLECTION



COS'È UNA COLLECTION

Condividere dati tra sistemi diversi

- Un framework di **dati condivisi**
- Gestisce **token** e **valori di stato**
- Riduce la **duplicazione** e favorisce la **coerenza** tra sistemi
- È la **base** su cui poggiano molti sistemi di Gameplay



ARCHITETTURA BASE

Come sono gestiti i dati

- Ogni dato ha un **tag**, un **valore** e la sua **variazione**
- Il container gestisce i **dati** e gli **eventi** di cambiamento
- Eventi tramite **tag specifici** o **gerarchici**
- La **Collection** fornisce un'API chiara per **accedere** e **aggiornare** i valori
- Il component collega la **Collection** ad un **Actor**

[Actor]

└─ UGPCollectionComponent

└─ UGPCollection

└─ FGPCollectibleContainer

└─ FGPCollectibleEntry

COMPONENT E REGISTRY

Gestire più collection nel mondo di gioco

- **Punto di ingresso** della Collection nel **mondo**
- Registrazione automatica nel **Registry globale**
- Gestione **centralizzata** di tutte le Collection attive
- Collection diverse per **contesti differenti**
- Accesso **indipendente**
- Struttura **modulare** e **scalabile**

[GameInstance]

└─ UGPCollectionRegistrySubsystem

└─ Collection.Registry.Global

└─ Collection.Registry.Quests

└─ Collection.Registry.World

EVENTI E REATTIVITÀ

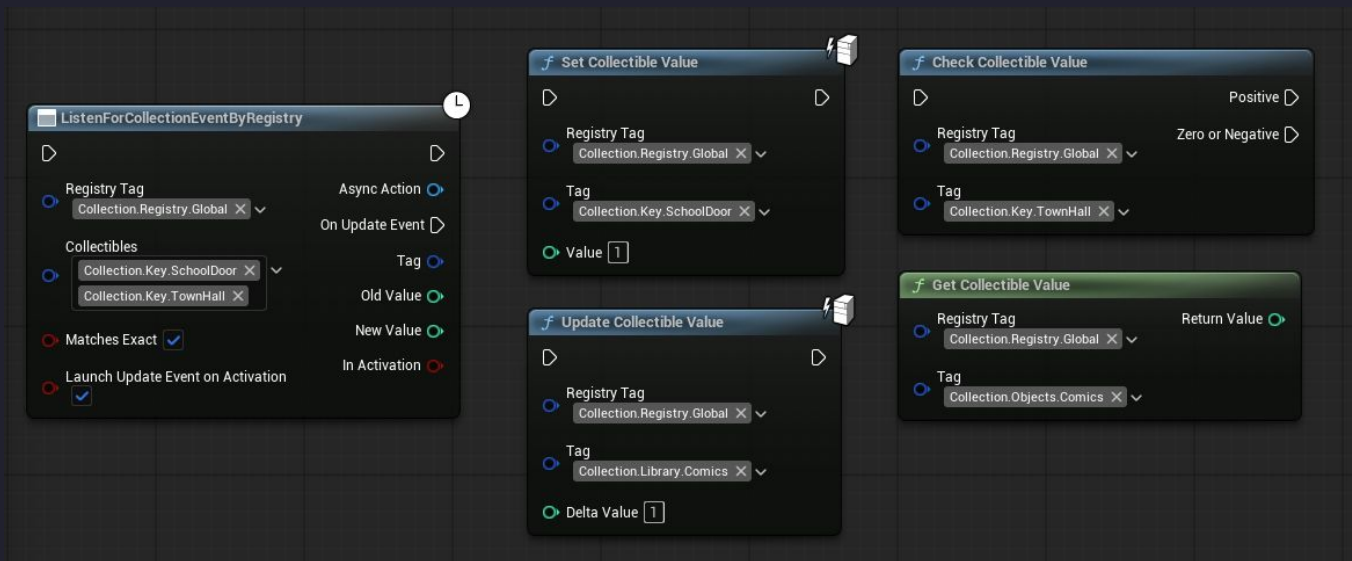
Rendere i dati reattivi

- Eventi generati ad ogni **variazione** di valore
- Sottoscrizione a **tag specifici** o intere **gerarchie** (es. *Quest.Token* o *Quest.Token.X.Y.Z*)
- Notifiche ordinate grazie al **batching**

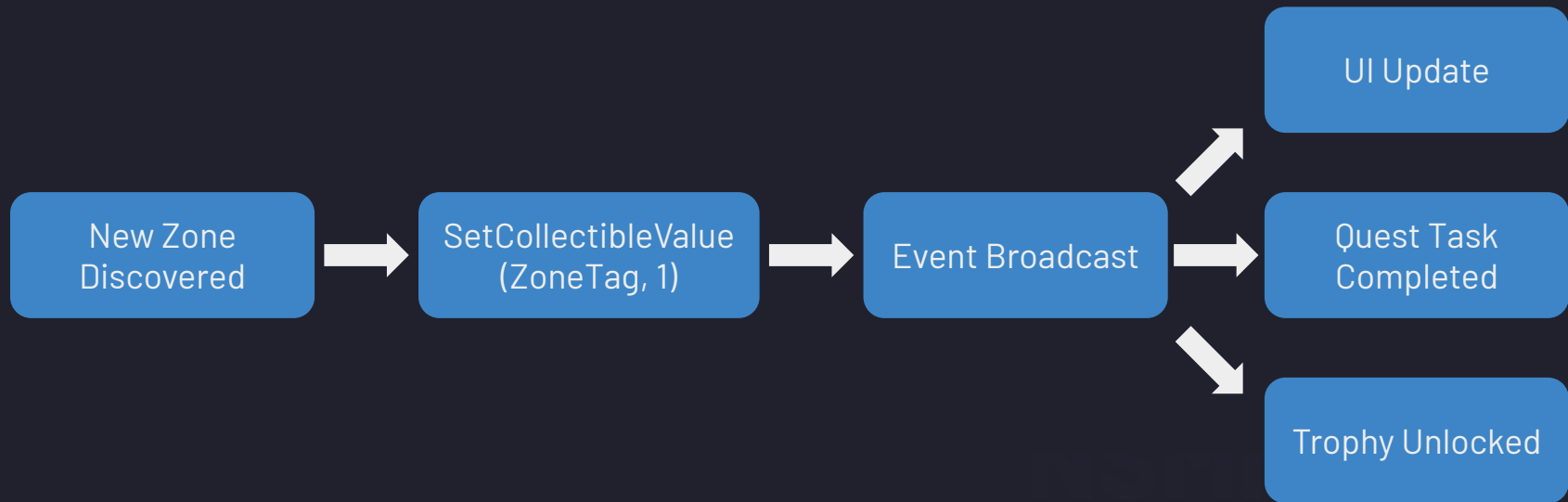
```
Collection->RegisterEvent (TAG_Token, bMatchesExact, this,  
  
[] (const FGameplayTag& Tag, int32 OldVal, int32 NewVal) {  
  
    UE_LOG(LogTemp, Log, TEXT("%s changed from %d to %d"), *Tag.ToString(), OldVal,  
NewVal);  
  
});
```

BLUEPRINTS

Alla portata di tutti



CASO D'USO PRATICO

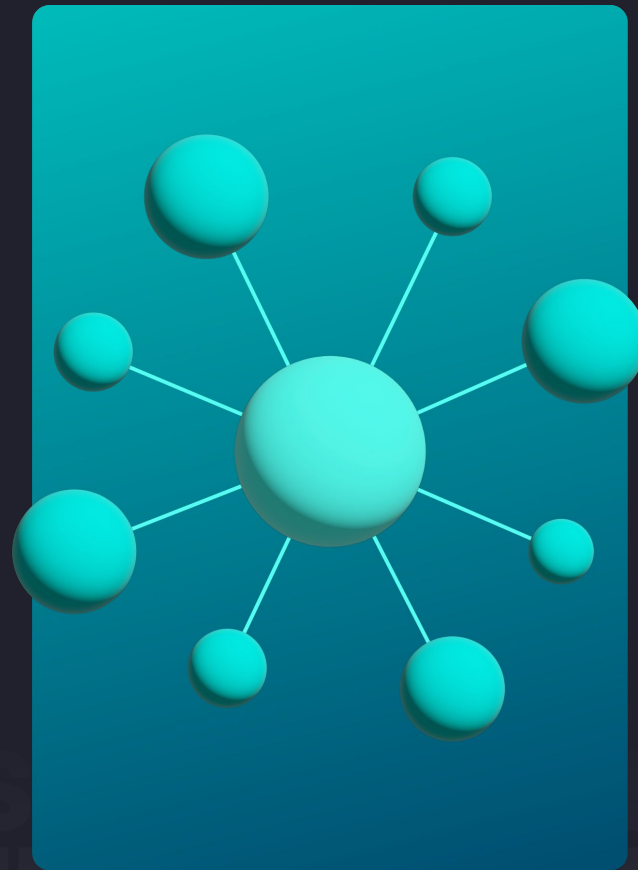


DEBUG TOOL

Registry	Global ▾	Collectible Tag	None ▾	0	Set Value
Tag					
Value					
OldValue					
▶ Recently Added					
Collection.Tutorial	Read Tome		1		0
Collection.Tutorial	Regain Energy		1		0
Collection.Tutorial	Find Food		1		0
Collection.Streamer	Complete Addictive		1		0
Collection.Streamer	Complete Addictive		1		0
Collection.Tutorial	Learn Tagging		1		0
Collection.Crafting.Recipe	Animal Compass		1		0
Collection.Crafting.Recipe	Animal Map		1		0
Collection.Crafting.Recipe	Animal Medicine		1		0
Collection.Crafting.Recipe	Animal Language		1		0
Collection.Crafting.Recipe	Animal Taming		1		0
Collection.Crafting.Recipe	BSG		1		0
Collection.Crafting.Recipe	Buildings		1		0
Collection.Crafting.Recipe	Read Tome		1		0
Collection.Crafting.Recipe	Complete Addictive		1		0

BENEFICI

- **Centralizzazione**
- **Modularità**
- **Serializzazione**
- **Reattività e trasparenza**



CONDITION



CONDITION: IDEA E SCOPO

Verità booleane a supporto del gameplay

- Ogni condition risponde a una **domanda booleana** sul mondo di gioco
- Permette di **reagire ai cambiamenti** senza creare dipendenze dirette
- È **pluggable**: può leggere dati da sistemi diversi
- Mantiene la logica **separata** dal comportamento del gameplay



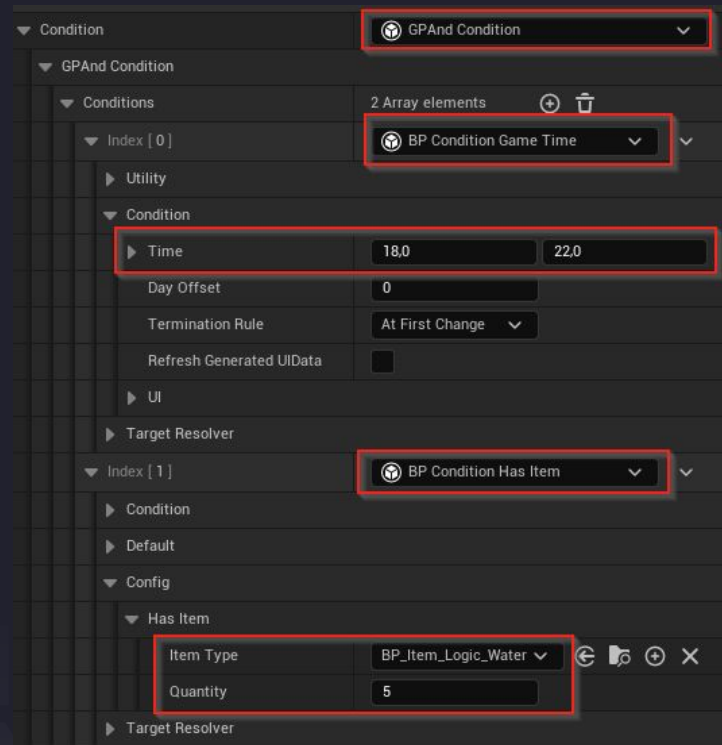
ESEMPI

- **HasItem:** il giocatore ha l'oggetto X nel suo equipaggiamento?
- **IsInVehicle:** Il giocatore sta guidando un veicolo?
- **GameTime:** È l'ora X? È tra le HH:MM e le HH:MM?
- **Weather:** Piove? È Sereno? C'è Nebbia?
- **AIAwareness:** Ci sono AI in stato "Searching/Alerted" entro un determinato raggio?
- **CombatState:** Il giocatore è in stato di fight? Ha finito di combattere da almeno N secondi?
- **CollectibleValue:** Che valore ha il token X nella global collection?

ATOMICITÀ E COMPOSIZIONE (AND / OR / NOT)

Dalle atomiche alle composite: logica modulare

- **Atomic Condition:** incapsula una singola regola su una singola fonte (es. "HasItem")
- **Composite Condition:** combina condition figlie:
 - **AND** → vero se tutte sono vere
 - **OR** → vero se almeno una è vera
 - **NOT** → inverte la verità di una figlia
- **Propagazione:** la composite si iscrive ai cambiamenti delle figlie e ri-valuta solo quando serve
- **Testing:** ogni atomic è testabile isolatamente; le composite sono prevedibili e leggibili



UN PO' DI CODICE

Definizione della classe UCondition

```
UCLASS( Abstract, Blueprintable, BlueprintType, DefaultToInstanced, EditInlineNew )  
class TSC_API UGCondition : public UObject
```

Esempio d'uso

```
UPROPERTY( EditAnywhere, BlueprintReadOnly, Instanced )  
TObjectPtr<UCondition> Condition;  
  
Condition->Start( [] (bool bVal)  
{  
    // Condition state changed  
});  
Condition->Stop();  
  
const bool bConditionVal = Condition->Evaluate();
```

UN PO' DI CODICE PT. 2

Esempi di conditions

```
bool UHasCollectibleCondition::Evaluate() const
{
    return UCollectionBPFunctionLibrary::GetValue(RegistryTag, CollectibleTag) == Value;
}

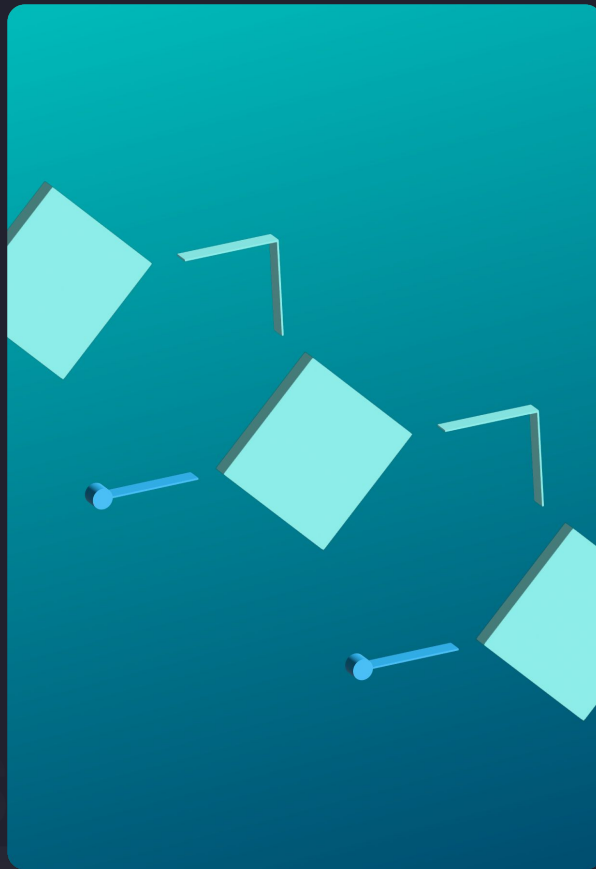
bool UIsInVehicleCondition::Evaluate() const
{
    return PlayerPawn->IsA(AVehicleBase::StaticClass());
}

bool UGameTimeCondition::Evaluate() const
{
    const float Hour = TimeOfDayProvider->GetCurrentHour();
    return Hour >= StartHour && Hour <= EndHour;
}
```

BEST PRACTICES

Come creare una buona condition

- **Singola responsabilità:** una condizione, una regola
- **Data-driven:** rendi configurabile tramite l'esposizione di parametri (es. Ora del giorno, tipo di item, etc)
- **Event-first:** preferisci delegati/eventi; fallback al polling quando non c'è alternativa
- **Lifecycle chiaro:** registra/deregistra dai sistemi collegati
- **No side effects:** le Condition non eseguono logiche di gameplay



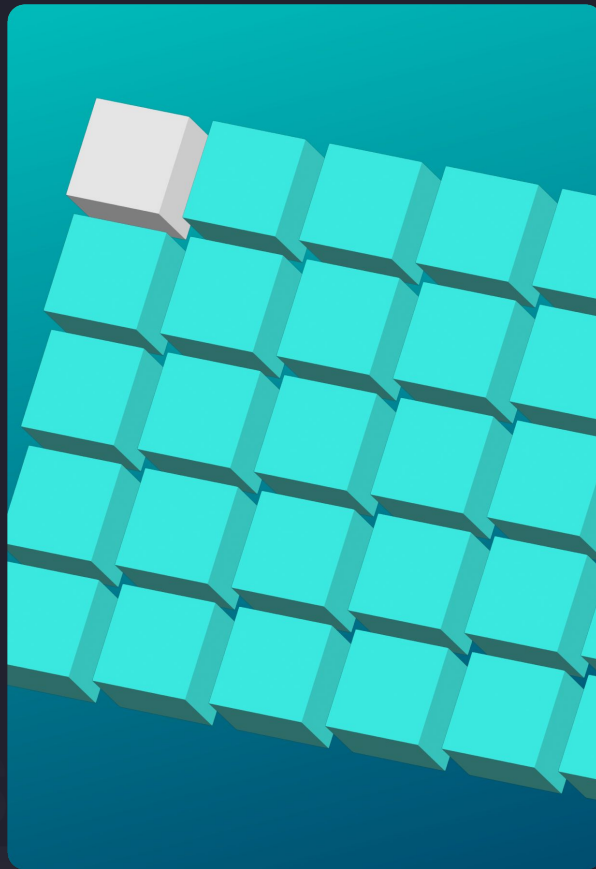
SAVE SYSTEM



UN SAVE SYSTEM GENERICO

Invece di tanti salvataggi ad hoc

- In un progetto complesso, **centinaia di sistemi e attori** devono salvare il proprio stato
- Serve un **sistema scalabile e automatico**, dove ogni oggetto dichiara *cosa* salvare e l'infrastruttura decide *come*
- L'obiettivo è mantenere **ordine, coerenza e flessibilità** nel tempo



I DATI SALVATI: FNSMSaveGameData

Il livello più basso: dove vengono serializzati i dati

- **Data** contiene i byte serializzati dei campi marcati **UPROPERTY(SaveGame)**
- È il “livello fisico” del salvataggio: un buffer binario compatto e indipendente dal tipo

```
USTRUCT ()  
struct NSMSAVESYSTEM_API FNSMSaveGameData  
{  
    GENERATED_BODY ()  
  
    TArray<uint8> Data;  
};
```

OBJECTS E SUBOBJECTS: FNSMSaveGameRecord

Il record di un singolo oggetto (e dei suoi componenti)

- **FNSMSaveGameRecord** rappresenta uno **UObject** salvato
- **Data**: i byte dell'oggetto principale
- **SubobjectsData** : i suoi componenti o sub-object (come UActorComponent), ciascuno con un proprio buffer
- Ogni record è **indipendente** e può essere ricostruito da solo

```
USTRUCT()  
struct NSMSAVESYSTEM_API FNSMSaveGameRecord  
{  
    GENERATED_BODY()  
  
    FNSMSaveGameRecordData Data;  
  
    TMap<FName, FNSMSaveGameRecordData> SubobjectsData;  
};
```

UNSMSaveGame

La mappa dei record: come viene organizzato un salvataggio

- Tutto il gioco viene salvato in una **TMap: Chiave - Record**
- La chiave (FName) è l'**identificativo dell'oggetto** (può essere un nome o GUID)
- La funzione **Serialize()** si occupa di:
 - **Scrivere/Leggere** la mappa completa (NamedRecords)
 - **Comprimerla/Decomprimerla** per ridurre lo spazio su disco

```
UCLASS()
class NSMSAVESYSTEM_API UNSMSaveGame : public
ULocalPlayerSaveGame
{
    GENERATED_BODY()

public:
    virtual void Serialize(FArchive& Ar) override;

private:
    TMap<FName, FNSMSaveGameRecord> NamedRecords;
};
```

SaveData E LoadData

Come serializzare/deserializzare le proprietà SaveGame di uno UObject

```
void UNSMSaveGameStatics::SaveData(UObject* Object,
TArray<uint8>& Data)
{
    FMemoryWriter MemoryWriter(Data, true, true);
    FNSMSaveGameArchive Ar(MemoryWriter);

    Object->Serialize(Ar);
}
```

```
void UNSMSaveGameStatics::LoadData(UObject* Object,
const TArray<uint8>& Data)
{
    FMemoryReader MemoryReader(Data, true);
    FNSMSaveGameArchive Ar(MemoryReader);

    Object->Serialize(Ar);
}
```

```
struct NSMSAVESYSTEM_API FNSMSaveGameArchive : public
FObjectAndNameAsStringProxyArchive
{
    FNSMSaveGameArchive(FArchive& InInnerArchive) :
FObjectAndNameAsStringProxyArchive(InInnerArchive, false)
    {
        ArIsSaveGame = true;
    }
};
```

```
UCLASS()
class UMySavedObject : UObject
{
    GENERATED_BODY()

    UPROPERTY(SaveGame)
    int MySavedVariable;
};
```

▼ Advanced	
Config Variable	☐
Transient	☐
SaveGame	☒
Advanced Display	☐
Deprecated	☐

CONCLUSIONI

Costruisci sistemi che generano soluzioni.

Progetta strutture pensate per risolvere problemi ricorrenti, non per chiudere casi singoli.

Fai comunicare i sistemi, non li mescolare.

I sistemi di Gameplay restano indipendenti, ma condividono dati e comunicano in modo chiaro e controllato.

Progetta per il futuro, non per oggi.

Il progetto, il team e le esigenze evolvono: un'architettura sana è quella che sa adattarsi nel tempo.



THE END

Grazie per l'attenzione! Ci sono domande?

CONTACT ME



[linkedin.com/in/d-mercurio](https://www.linkedin.com/in/d-mercurio)

JOIN US



naconstudiomilan.com/careers