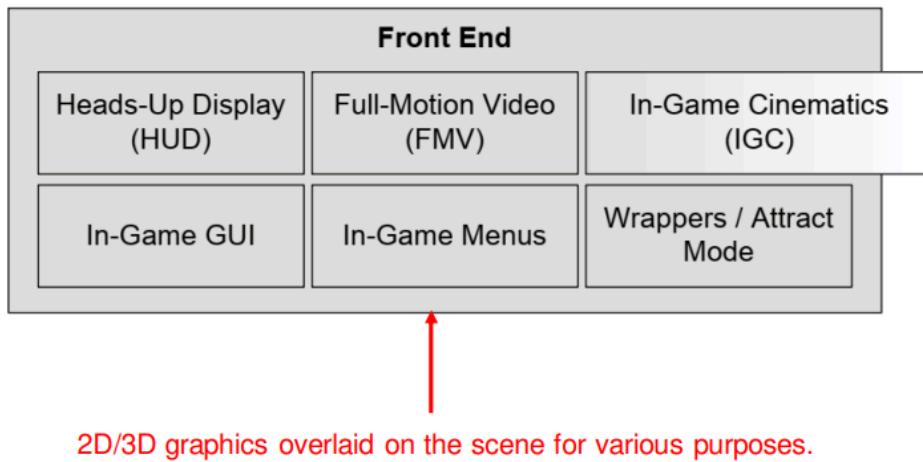


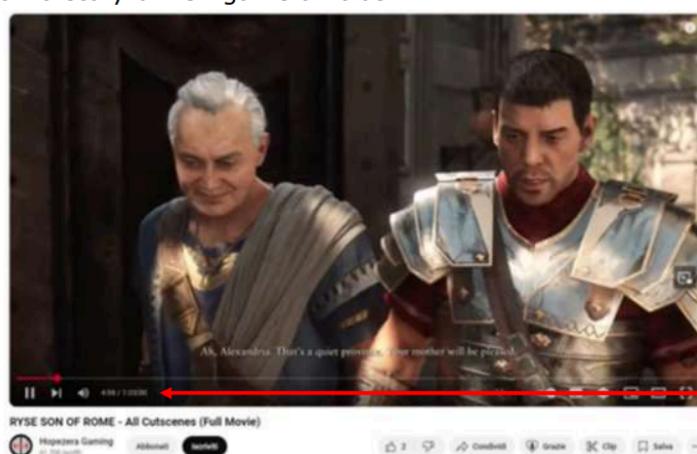
Lezione 6 26/03/2025

Continuo di game engines

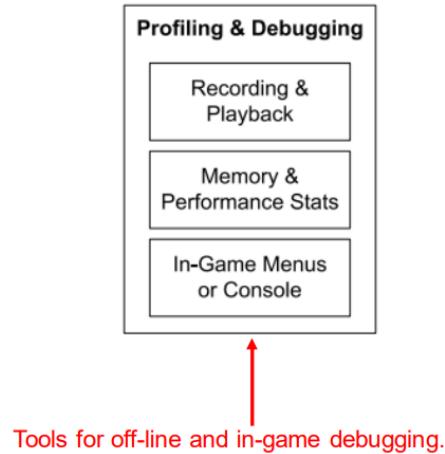


Abbiamo poi il modulo che gestisce tutti i **comandi/menu** che vengono mostrati a schermo durante il gioco, e anche tutte quelle informazioni in **overlay** che possono essere utili sia al giocatore durante la partita che allo sviluppatore per debug.

- ◆ In-game cinematic/ cut-scene / non-interactive sequences
 - Defines how a story-driven game unfolds



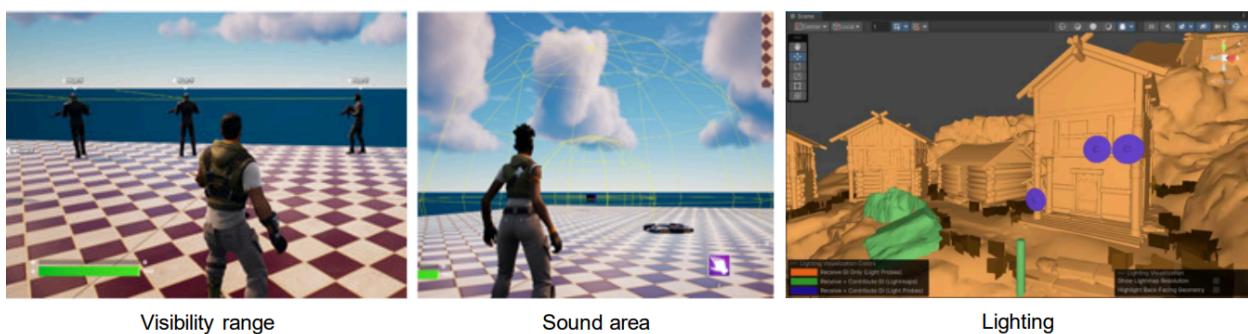
Abbiamo anche cinematiche, o le cutscenes, che al giorno d'oggi in realtà sono sequenze animate 3D, non sono pre-registerate.

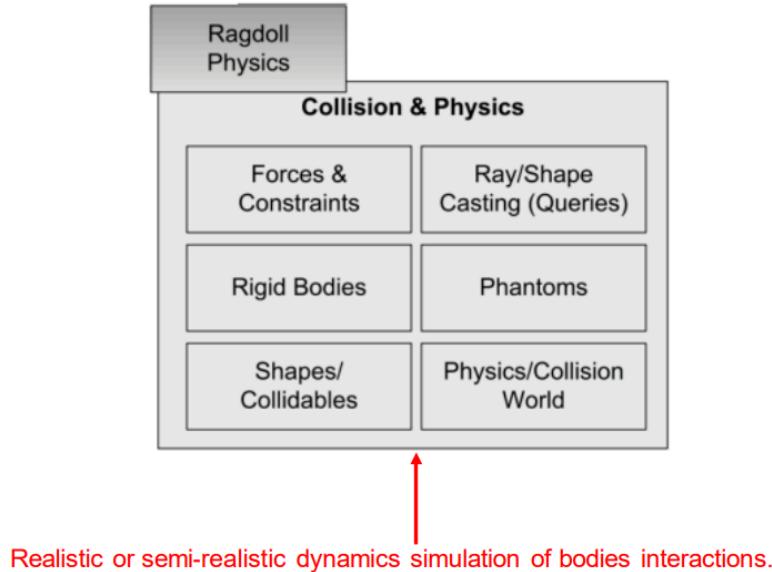


Poi abbiamo il modulo di **profiling e debugging**, che serve per la parte di sviluppo. Ci sono tool per il debug del codice e del gioco stesso.

Ci sono spesso dei menu nascosti nel gioco stesso, che possono mostrare informazioni utili di debug, come i tempi di calcolo, o dei menu che permettono di interagire con la partita attuale per abilitare/disabilitare degli eventi.

Può anche essere necessario controllare se alcune informazioni che non sono visibili, ma servono per attivare degli eventi, sono state inserite nel modo corretto.





Poi abbiamo tutta la parte della **fisica**, per creare un'esperienza realistica.

In generale un game engine deve affrontare 2 problemi di fisica:

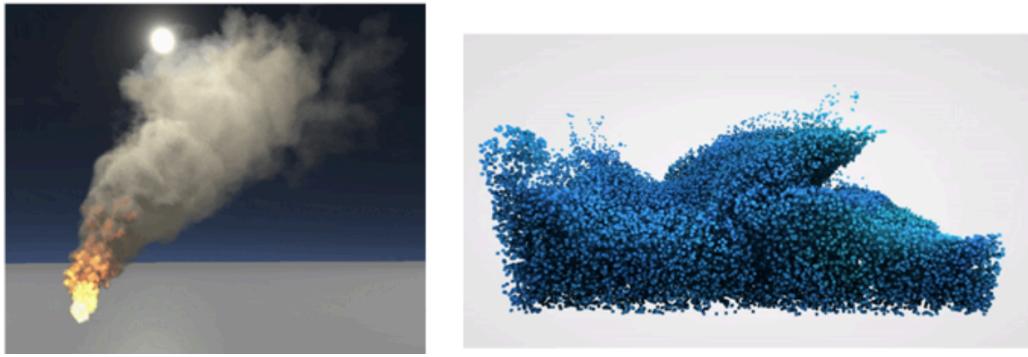
- **Come si comportano gli oggetti a delle forze esterne**, come qui se crolla il grattacielo come si comportano tutti i pezzi:



- Le **collisioni**, per non attraversare oggetti rigidi. Quindi bisogna verificare i contatti tra oggetti che non dovrebbero intersecarsi tra di loro, questo va fatto su tutti gli oggetti nella scena, sia statici che dinamici.

Questi sono problemi pesanti, infatti sono parallelizzati e vengono usati algoritmi di ottimizzazione, tramite librerie.

Se si vuole simulare qualcosa che non è un oggetto solido, come delle particelle:



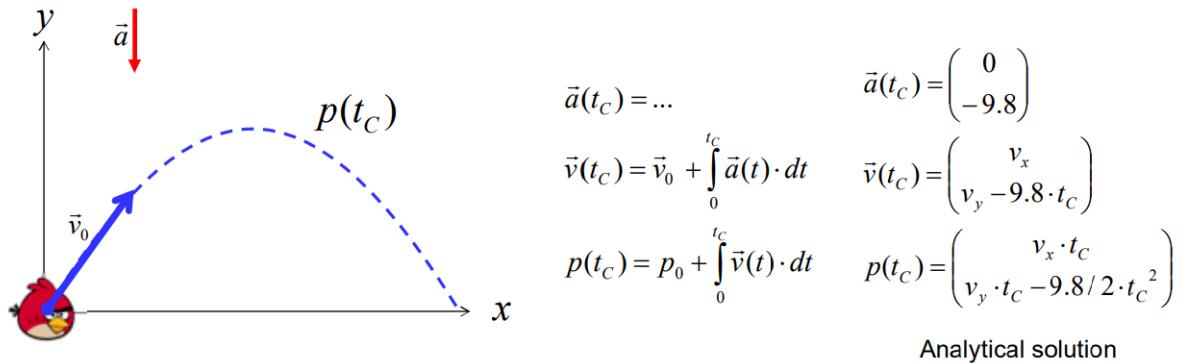
nei giochi si usano delle scorciatoie, si simulano le particelle in modo diverso, con degli oggetti discreti per dare l'illusione.

Qui ci interessano degli **oggetti rigidi grandi**:

dato un oggetto, se dovessimo modellare il suo comportamento, magari perchè si sta muovendo nella scena, dovremmo considerare la sua posizione, la sua velocità, accelerazione...

◆ Rigid Body

- A single, underformable, object obeying Newtonian laws of dynamics
- Dynamic as computation of position, velocity, acceleration,

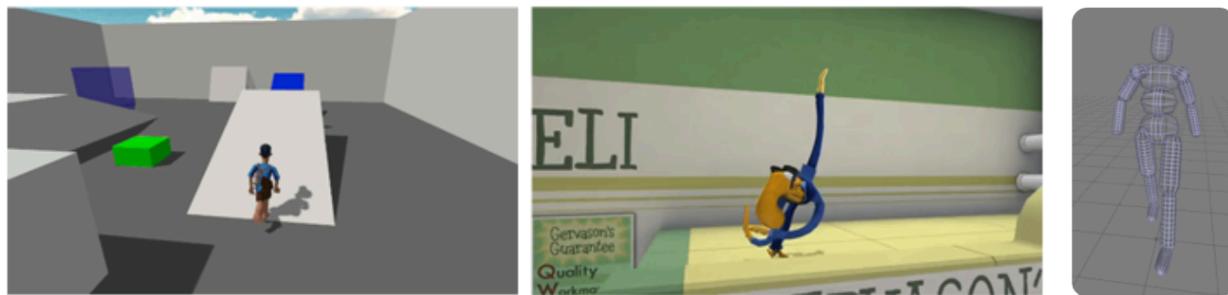


Abbiamo una serie di equazioni integrali a cui possiamo dare i parametri dell'oggetto per capire al tempo t dove si trova l'oggetto. Spesso però non risolviamo questi integrali, non calcoliamo la posizione al tempo t , ma piuttosto si va ad approssimare la posizione ad intervalli discreti (il prossimo frame). Abbiamo quindi un'approssimazione, quindi si rischia che l'errore si accumuli.

Poi abbiamo la gestione degli **oggetti grossi ma non monolitici**, per esempio se dobbiamo simulare il comportamento di un essere umano questo non è un oggetto rigido, ma possiamo simulare alcune sue parti come se fossero un oggetto rigido.

◆ Rag doll

- To simulate realistic physical reactions of a dead body, a rag doll system is used
- A rag doll consists of a **set of rigid bodies**, each representing body part
- The physics system calculates the positions and orientations of the rigid bodies.

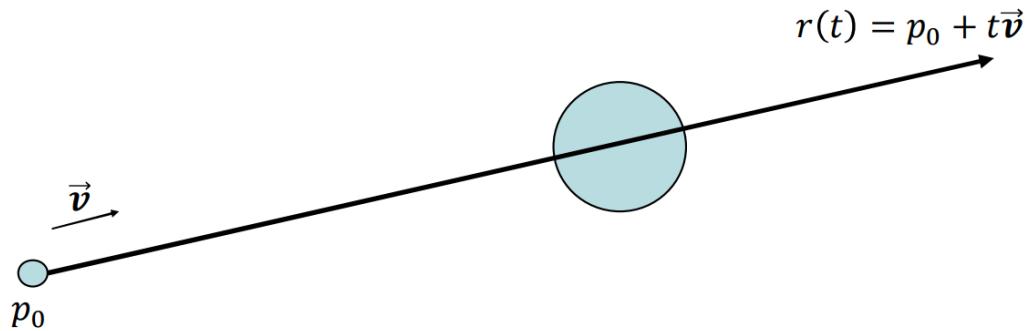


La **ragdoll** è quando si deve simulare un corpo morto. Si hanno dei problemi, per esempio in base a quando muore, bisogna calcolare la traiettoria che segue per raggiungere la terra, il modo in cui viene acciuffato, in base a cosa colpisce mentre cade...

Tipicamente si simulano parti del corpo come dei singoli oggetti singoli ma vincolati, e vedere come ciascuna parte rigida si comporta.

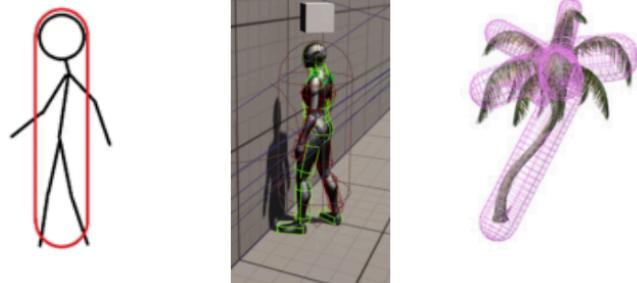
Questo è diverso dalla camminata, che può essere animata tramite script.

Abbiamo il problema dell'**analisi delle collisioni**, un oggetto va controllato istante per istante per vedere se collide con qualcosa. Questo spesso viene fatto tramite un "raggio" che segue la traiettoria dell'oggetto, e si controlla se interseca qualche oggetto.

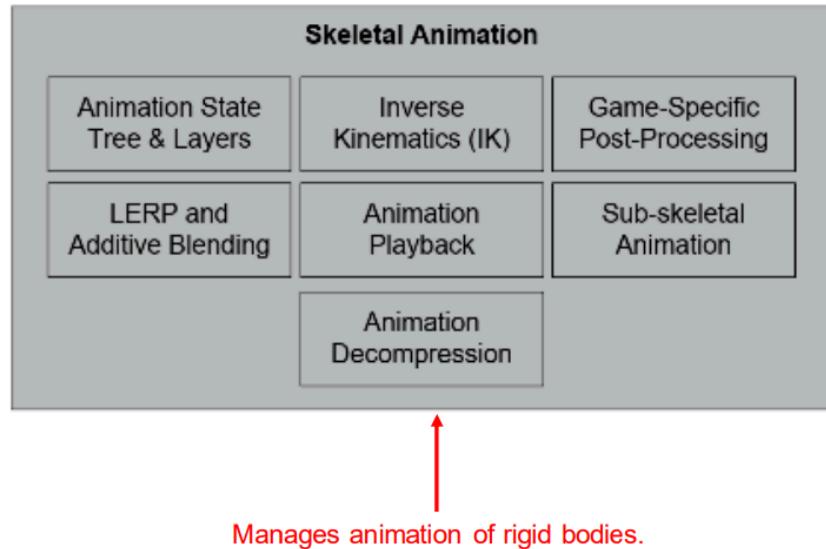


è facile fare il conto di questi raggi, restringendo l'area di ricerca di questi test. Questi raggi non si applicano alla geometria vera degli oggetti, ma si approssimano le geometrie:

- Approximate the shape of the objects with an, easy to detect collision, one
 - Spheres
 - Capsules
 - Axis Aligned (Bounding) Boxes
 - Generic Boxes
 - Cylinders
 - Convex polyhedron
 - Generic polyhedron
 - ...



Un altro caso di test di collisione sono i **triggers**. è una collisione con un ostacolo invisibile, questi trigger sono chiamati **phantoms**. Sono oggetti invisibili che vivono nella scena, e quando ricevono una collisione attivano un comportamento.

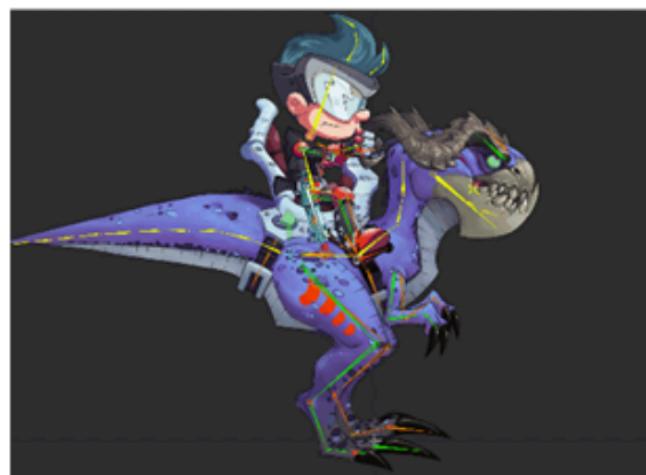


Un gioco deve avere **animazioni**. Ci sono diversi tipi di animazioni che vanno gestite.

Nei vecchi giochi, nel 2D, le animazioni sono una sequenza di immagini.

Ora si usa l'approccio dell'**animazione per parti**, si divide l'oggetto e ogni parte viene gestita come se fosse un'entità separata. Si ha quindi l'impressione che l'intero oggetto sia animato.

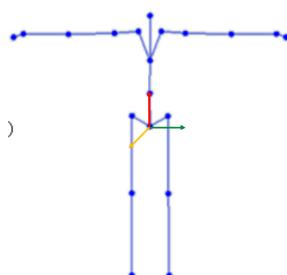
Questa animazione è fatta ruotando, spostando, traslando singole parti rigide (scheletro).



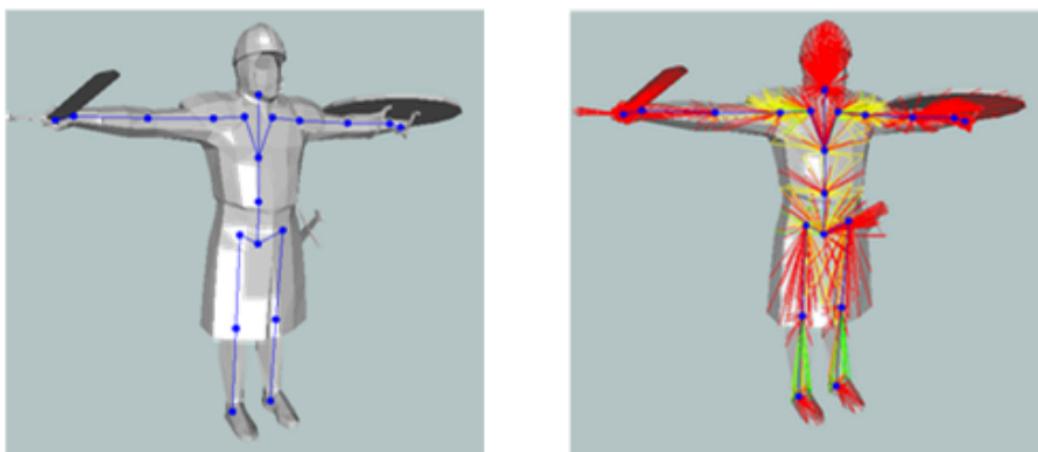
L'animazione si basa su questo concetto di scheletro: ho degli oggetti complessi che hanno diverse parti, e io cerco di animare le diverse parti separatamente. Ho bisogno una struttura dati che mi permetta di collegare le varie parti di modo che se ne sposto una, si porta indietro anche le altre parti collegate.

Questa animazione ha bisogno di diversi ingredienti:

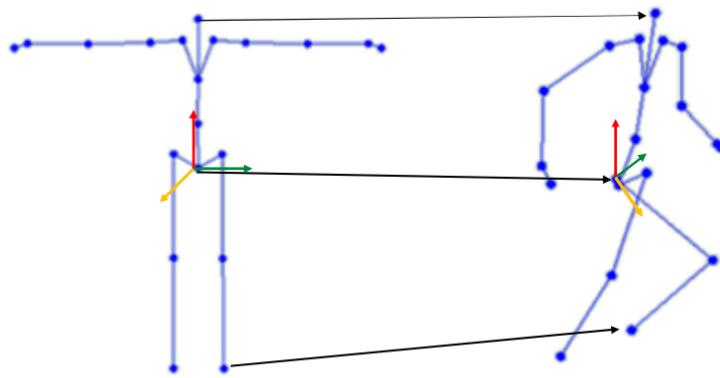
- uno scheletro: devo definire una struttura che mi dice che ci sono dei pezzi collegati ad altri pezzi, ad alto livello. è una struttura di supporto. La definizione dello scheletro si chiama **rigging**:
 - questa struttura ha un sistema di riferimento locale, quando si modellano queste strutture si ha la posa T, che è la posa di "riposo" standard.



- Poi si inserisce nella mesh che rappresenta l'oggetto 3D, collegando dei punti dello scheletro ai vertici più vicini. Di modo che per fare un'animazione possiamo muovere lo scheletro, che si porta indietro i vertici della mesh. La definizione di questi collegamenti si chiama **skinning**.

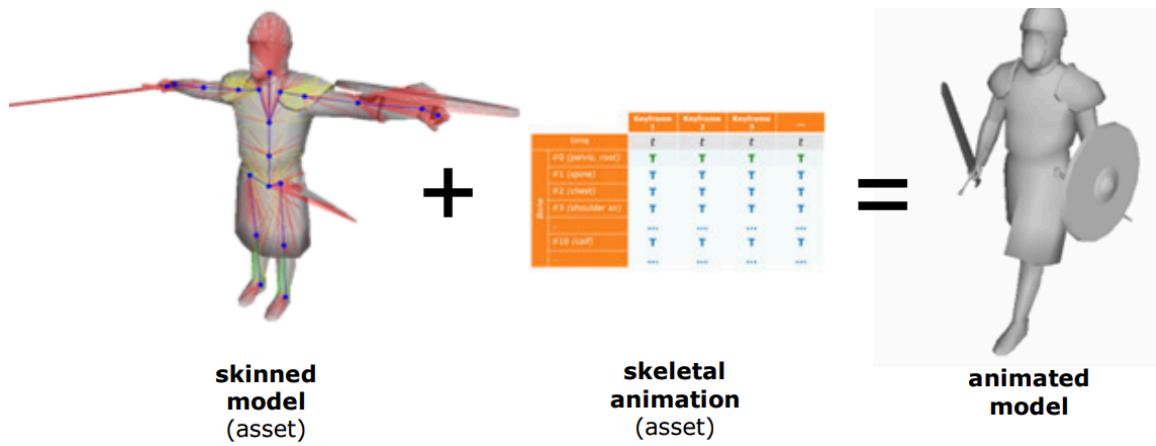


Queste due cose insieme definiscono l'**animazione**.



L'animazione è una sequenza di trasformazioni da applicare ai vertici dello scheletro, che a loro volta le applicano ai vertici della mesh.

Si può anche spostare l'animazione da un oggetto all'altro, se hanno uno scheletro compatibile.

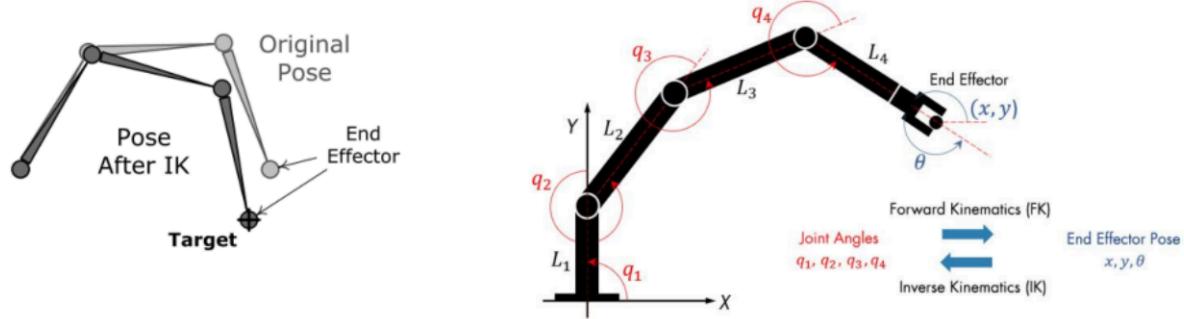


Quello che abbiamo visto fin ora è la **cinematica diretta**: dati questi punti di controllo e queste trasformazioni, ottengo le posizioni finali dei punti di controllo.

◆ Inverse kinematics



- The problem of obtaining joint angles from known coordinates of end effector
- Define the final position of an end effector (e.g. hand)
- Solve a minimization error problem to fit the configuration of joints (e.g. arm bones)



Invece la **cinematica inversa**: data la posizione finale che voglio raggiungere, che sequenza di animazioni mi serve per raggiungerla?

La cinematica inversa è un modo di correggere l'animazione standard, utile per esempio quando dobbiamo simulare una camminata in un terreno non planare, o se dobbiamo salire le scale.

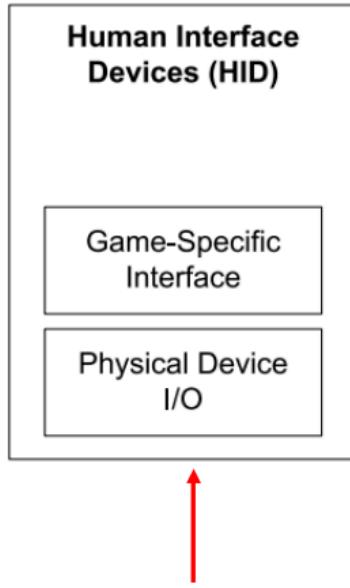
◆ Inverse Kinematics



- Useful to correct forward animation
- E.g. Walking on uneven terrain



Si usa quindi per capire se, prima di compiere il passo di salire il gradino, controlliamo se la posizione finale ha senso.



Process input from the player, obtained from various human interface devices (HIDs).

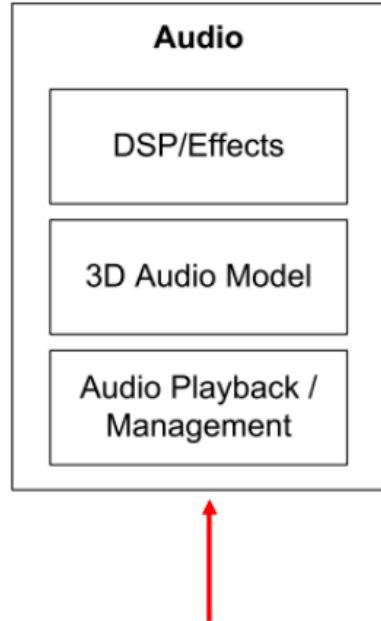
I dispositivi di input sono molti. La cosa importante è che questo modulo, che si deve interfacciare con il dispositivo, deve mascherare la parte di basso livello e quindi gestire tutti i dispositivi possibili.

Questi dispositivi possono comunicare in modi diversi:

- ◆ Game software reads and writes HID inputs and outputs in various ways
 - Polling
 - Explicitly querying the state of the device periodically
 - Reading hardware registers directly, reading a memory-mapped I/O port
 - Game pads and old-school joysticks
 - Interrupts
 - Electronic signal generated by the hardware
 - CPU temporarily suspend execution of the main program and an interrupt service routine (ISR)
 - Mouse, keyboards,...
 - Wireless
 - Communicate via Bluetooth protocol
 - Handled by a separate thread
 - Wiimote, Dualshock 3, Xbox360,...



I dispositivi di input possono anche dare feedback, come audio vibrazioni, feedback di forza...



Manages audio playback and sound effects with 3D audio rendering.

Asset audio, bisogna gestire anche il rendering 3D, l'audio deve rimbalzare sulle pareti, deve cambiare in base a come oriento la faccia...

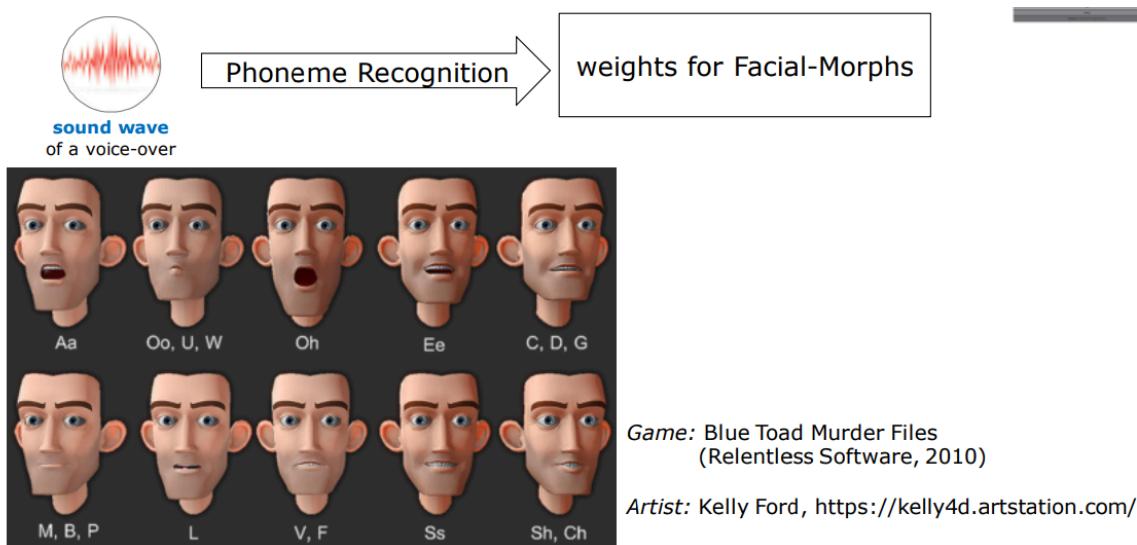
◆ Three basic phenomena:

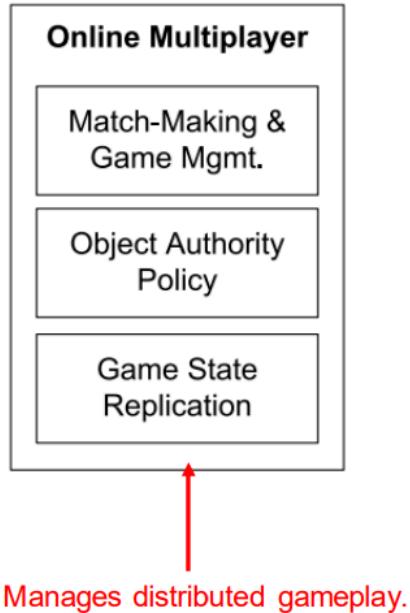
- Absorption
 - Some* energy of the sound-wave is lost (dissipated into heat)
- Reflection:
 - Some* part of the sound-wave bounces off (e.g.) walls
- Transmission:
 - Some* part of the sound-wave passes through solid objects

◆ Most common file formats

- .mp3
 - perceptual encoding
 - good balance between compression-ratio / quality
 - common for final releases / distributions
- .ogg (vorbis)
 - optimized for music
 - usually best quality for compressed
- .wav
 - uncompressed (PCM)
 - or, compressed (ADPCM)
 - common in production

Una parte legata all'audio e all'animazione è quella di sincronizzare il labbiale dei personaggi con l'audio effettivo





Una parte del game engine deve gestire il **networking**. Il problema è che tutti i giocatori devono vedere lo stesso stato della scena.

Bisogna risolvere molti problemi:

◆ Game networking

- What to communicate?
 - e.g.: complete statuses, status changes, inputs...
- How often ?
 - at which rate
- Over which protocol ?
 - TCP, UDP, ...
- Over which network architecture ?
 - Client/Sever, Peer-To-Peer
- How to deal with networking problems
 - latency ("lag") <== one main issue
 - limited bandwidth
 - connection loss / loss of packets

◆ Network protocols: UDP vs TCP

TCP sockets

- ◆ **Connection based**
 - ◆ Guaranteed reliable
 - ◆ Guaranteed ordered
 - ◆ Automatic breaking of data into packets
 - ◆ Flow control
 - ◆ Easy to use, feels like read and write data to a file

UDP sockets

- ◆ **What's a connection?**
 - ◆ No reliability
 - ◆ No ordering
 - ◆ Break your data yourself
 - ◆ No flow control
 - ◆ Hard.
Must detect and deal with problems yourself.

◆ Virtual connections over UDP

- add connection ID to packets
 - to filter out unrelated ones
- time out on prolonged silence (~ few secs)
 - declare "connection" dead
- add serial number to packets
 - to detect when one went missing / is out of order / is duplicate
- give ack back for received packets
 - optimize for lucky (& common) cases!
 - e.g. N (say 100) received msg == 1 ack (with bitmask) resend? only a few times, then give up (data expired)
- congestion avoidance: measure delivery time
 - tune send-rate (packets-per-sec) accordingly



Tipicamente i game engine hanno un protocollo custom basato su UDP.

◆ Choosing a protocol: a question of pacing

- fast paced game?
 - action games, FPS, ...
 - (sync every 20-100 msec)

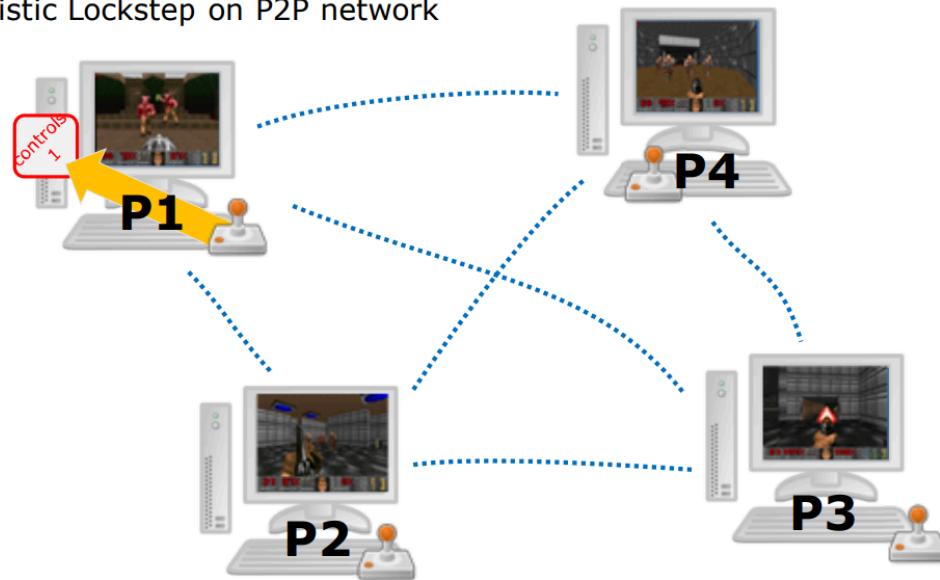
→ UDP almost necessary
(unless LAN only)
- slow paced game?
 - RTS, RPG...
 - (sync every ~500 msec)

→ can get away with TPC
- slower paced games?
 - MMORPGs, cards ...
 - (sync every few sec)

→ why not just HTTP
- traditional turn based ?
 - chess, checker
 - (sync every hour/day)

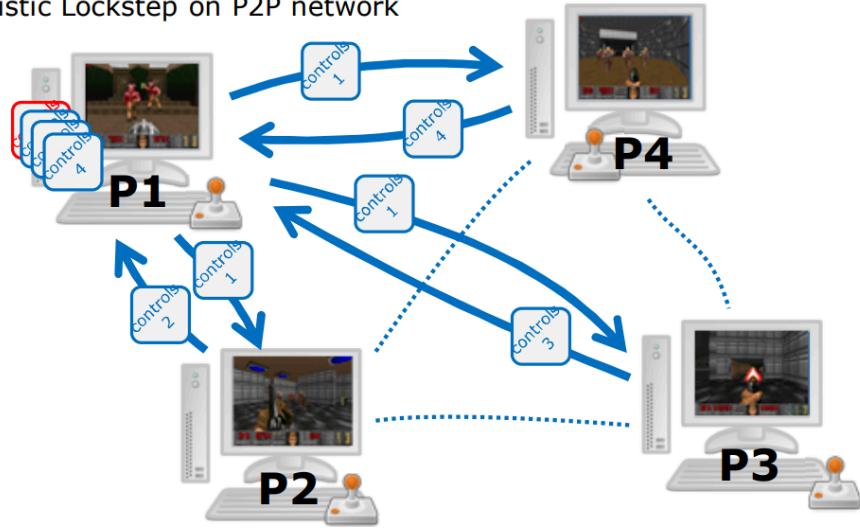
→ may as well use EMAIL

◆ Deterministic Lockstep on P2P network



Se io genero un'azione tramite il controller, questa azione non viene immediatamente eseguita, ma viene comunicata a tutti gli altri pc.

◆ Deterministic Lockstep on P2P network



La stessa cosa devono farla tutti gli altri pc. Prima di poter renderizzare, dobbiamo aspettare tutti i comandi da tutti i pc interconnessi in rete.

Poi, ciascun computer potrà renderizzare l'istante di gioco.

C'è quindi una separazione tra quando l'evento viene registrato a quando viene mostrato a schermo.

Il problema di questa modalità è che se un'informazione viene persa, si blocca tutto. L'altro problema è dato dal termine "deterministic", ovvero il pc deve prendere gli eventi, applicarli al gioco e generare l'istante di gioco successivo. Ma il problema è che diversi pc possono funzionare in modo diverso, deve essere fondamentale che soprattutto dal lato fisico se partiamo dallo stesso punto di partenza e applichiamo gli stessi comandi, il risultato deve essere lo stesso per tutti i giocatori.

◆ Limitations of deterministic Lockstep

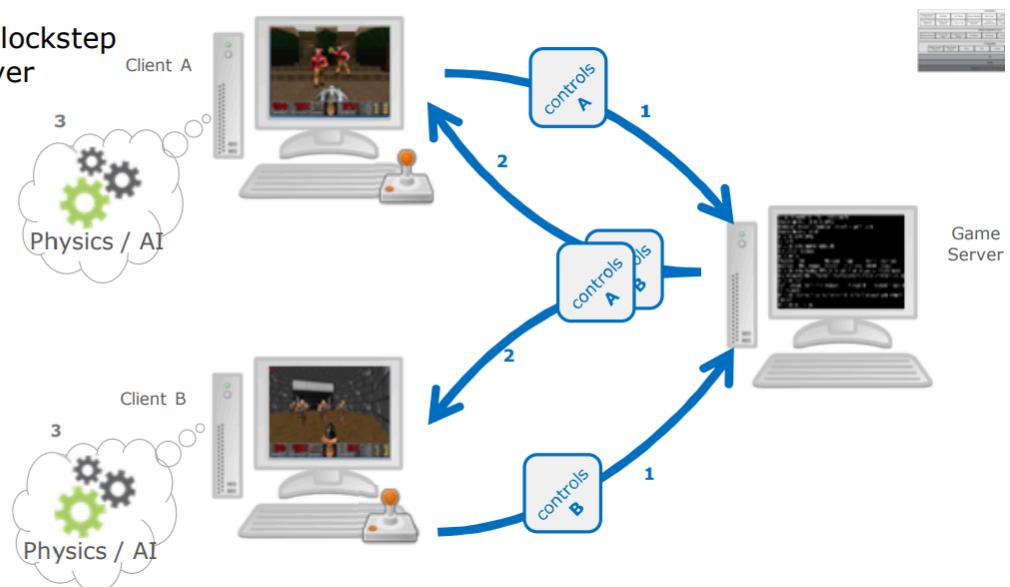
- Does not scale with number of players
- Responsiveness
- Input rate == delivery rate
- Delivery rate == as fast as the slowest connection
- If connection problems -> everybody freeze
- Assume complete determinism (can be problematic)

◆ Limitations of peer-to-peer

- Do not scale with number of players
- Every client needs to send/receive from all other clients

L'altro problema è che più i giocatori aumentano e più computazioni bisogna fare, quindi la complessità è quadratica.

◆ Deterministic lockstep on Client-Server



Un'altra soluzione potrebbe essere quella di appoggiarsi ad un server centrale. Tutti gli input vengono inviati al server che li colleziona e poi li spedisce, una volta raccolti, ai giocatori.

Il vantaggio è che la complessità delle comunicazioni è lineare, non dobbiamo più mandare tutto a tutti. Se vogliamo questo server può anche gestire i casi limite,

come se non ricevesse un dato da un giocatore. Però questo non risolve il problema del determinismo.

◆ Advantages

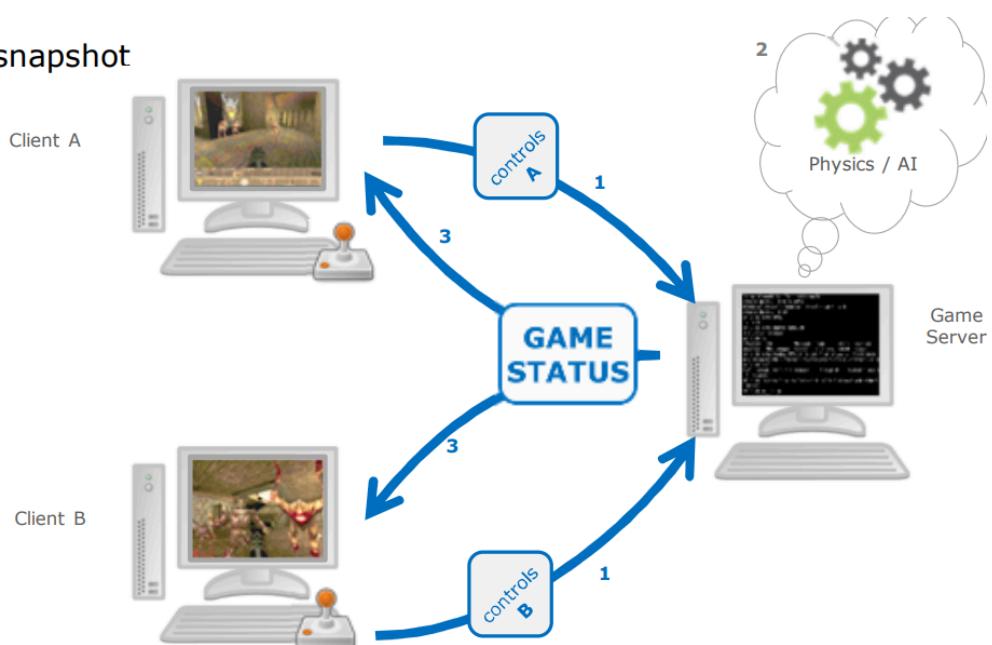
- Scalability: number of packets is linear (not quadratic)
- The server can now be made authoritative. E.g:
 - Packet loss from player 3?
Server makes up controls for player 3 (instead of waiting for them). Packet loss affects one player only



◆ Cost:

- Responsiveness: latency = $2 \times$ delivery time

◆ Game-status snapshot paradigm



Un'altra soluzione è quella di ridurre i client a semplici visualizzatori di stato, senza dover computare lo stato.

I client spediscono al server i comandi, e il server genera lo stato del gioco, riprovidendolo ai client che si limitano a mostrarlo. Questo sistema il determinismo.

Il problema è quale informazioni inviare, il server deve fare parecchio lavoro, bisogna cercare di inviare informazioni minimali.

◆ World “Snapshot” contains:



- Data needed for 3D rendering:(position-orientation of objects, plus anything else needed)
- All data needed for physic simulation

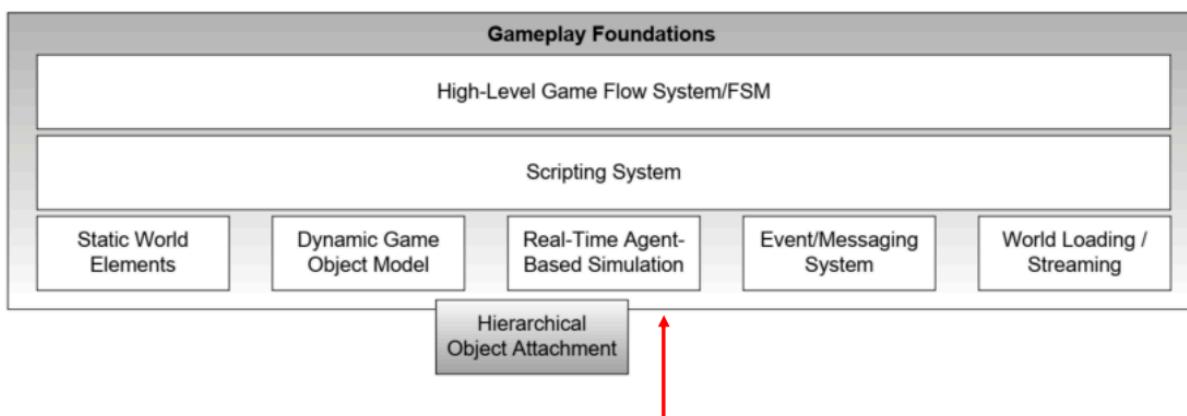
◆ Pros:

- Determinism: not a required assumption anymore
- Joining ongoing games becomes trivial
- Packet loss and slower connections bearable

◆ Cons:

- Packet size is a lot bigger! optimizations, to counter this:
 - compress world status
 - send only the portions of the status which changed or which interest a player
- Responsiveness: 2 x delivery time

Poi se un nuovo giocatore vuole entrare, non si può nella modalità peer to peer. Nella seconda modalità client-server si può fare parzialmente. Con lo snapshot si, il server gli manderà lo stato di quel momento.



Provides a suite of core facilities, upon which game specific logic can be implemented conveniently

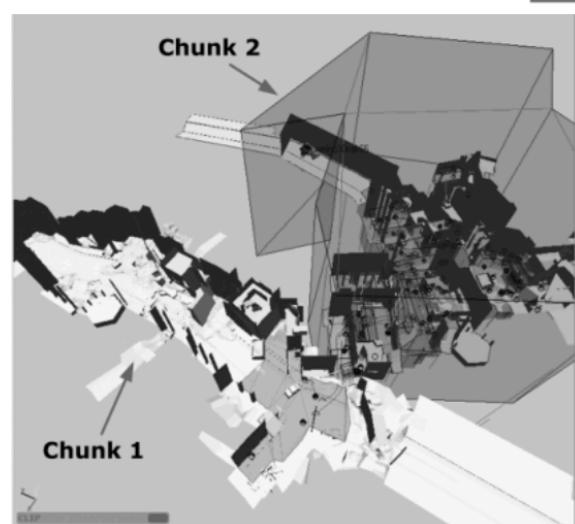
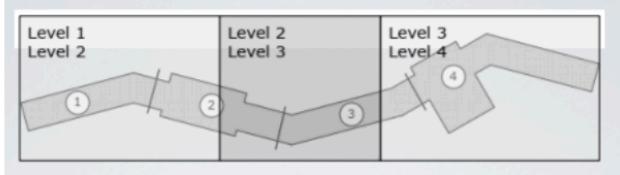
Questo blocco si deve occupare di come è strutturato il mondo, la partita (ci sono livelli, mondi, collegati...). Serve a definire in modo semplice tutta la **logica e meccanica di gioco**.

Troviamo la messaggistica per gli eventi (come si parlano gli oggetti), il sistema di scripting (un linguaggio che descrive come si svolge il gioco stesso, ovvero come succede quando accade un evento). Tutti quei giochi che ammettono il modding, sono basati su script, quindi si va a modificare lo script di un dato evento.

Tutti i game engine permettono editare il mondo e di associare a ciascun elemento delle proprietà come animazioni.

Un problema principale è come **gestire i livelli di gioco**, nei giochi passati i livelli erano completamente staccati. Quelli più recenti invece sono molto basati sulla continuità del gioco. Quindi bisogna caricare le parti di mondo nuove, e rimuovere quelle vecchie, questo è un sistema di streaming.

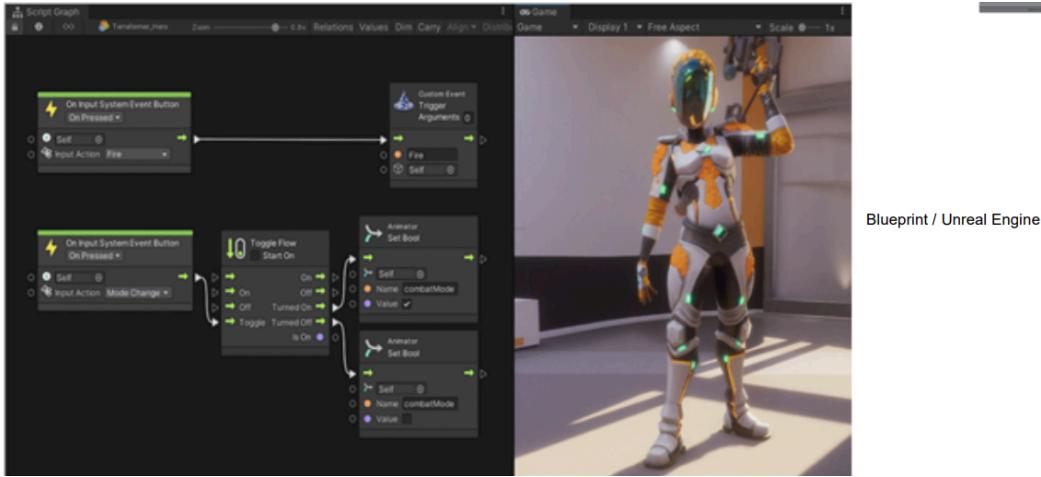
◆ World Chunks



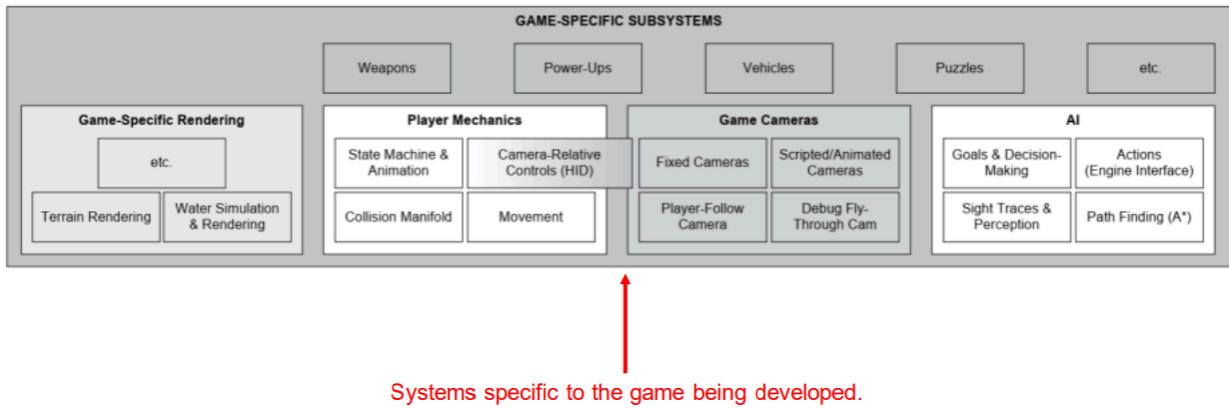
C'è anche un sistema di "airlock" dove per esempio il giocatore entra in una stanza, un'ascensore, e nel frattempo il mondo esterno viene sostituito.

Lo scripting è la parte più importante di questo modulo, si possono usare linguaggi standard.

Spesso questi script vengono anche creati tramite linguaggi visivi.



Nello scripting dobbiamo anche considerare la parte della storia del gioco.
Servono scripts per i dialoghi etc.



L'ultimo blocco è tutta la parte generica, c'è anche AI, vedremo.