Chapter 1

Introduzione

Il seguente documento è stato realizzato con lo scopo di fornire un punto di partenza per tutti coloro che intendono approcciarsi alla programmazione.

La tecnica che si è scelto di utilizzare è quella di insegnare a programmare mediante la realizzazione di un videogioco, confidando di riuscire a stimolare la curiosità e la fantasia dei lettori tramite vicende e personaggi immaginari. Domande del tipo "Come posso far volare questo personaggio?", "Come posso fare in modo che possa usare una spada per difendersi e combattere?", "Come si realizza una mappa del mondo?", "Come posso implementare un inventario?" oppure ancora "Come faccio a far si che i nemici attacchino il giocatore?", sono quelle che porteranno a volare verso nuovi orizzonti.

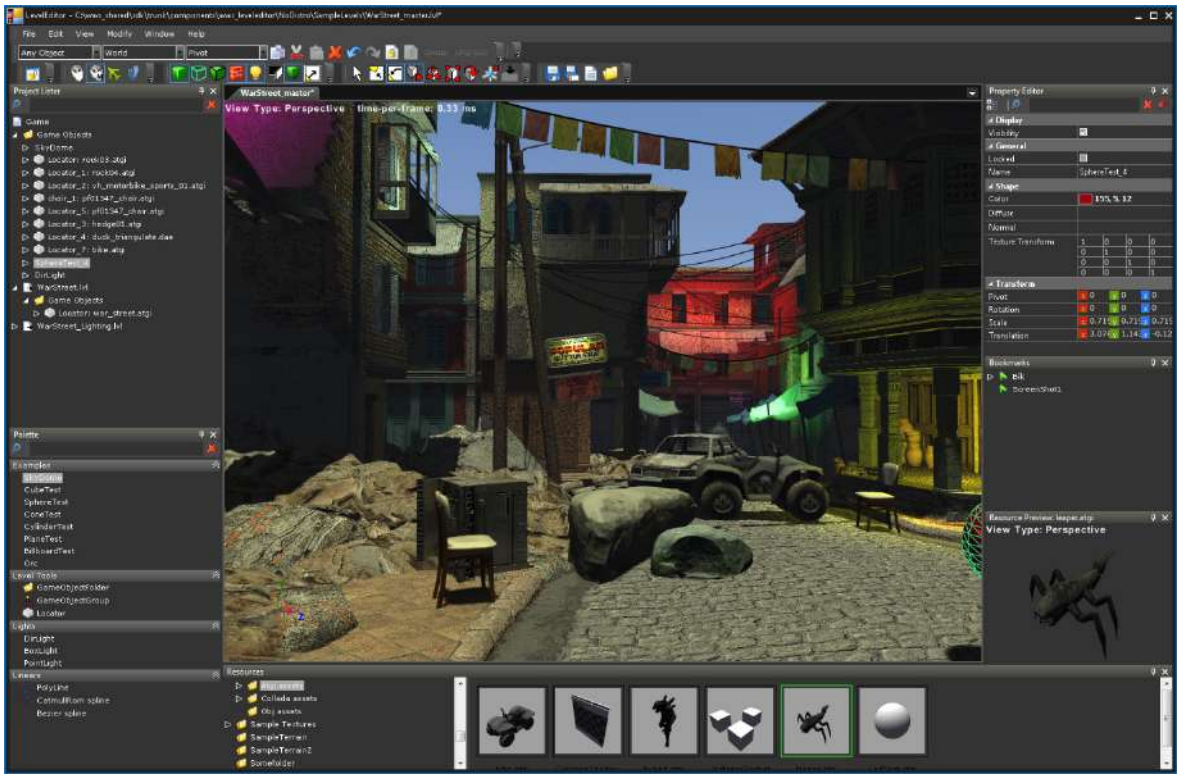
Il seguente capitolo si pone come obiettivo quello di rispondere ad alcune domande che ci si pone quotidianamente quando si parla di videogiochi; sfatare miti; dissipare dubbi ed eliminare false certezze; trovare soluzioni a problemi comuni e, inoltre, illustrare le motivazioni che si celano dietro la scelta della tecnologia utilizzata in questo documento.

1.1.1 Godot

In questo paragrafo, vedremo le motivazioni che ci hanno portati a scegliere Godot come strumento per la realizzazione del videogioco di questo documento; cos'è un game engine; perché non si sono scelte altre opzioni come ad esempio Unity e, infine, le differenze che ci sono tra Game Engines e Game Framework.

1.1.1.1 Che cos'è un Game Engine

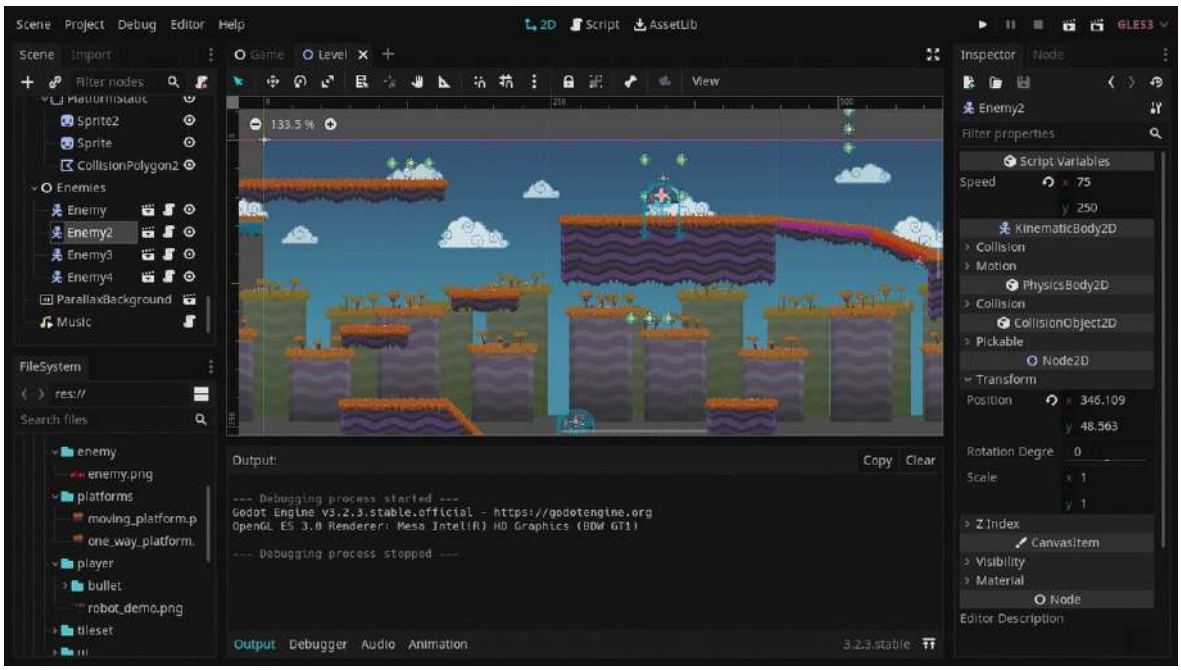
Un *Game Engine* è un framework ideato principalmente per lo sviluppo di videogiochi e, in generale, include librerie e supporti per lo sviluppo, come, ad esempio, i *Level Editor*.



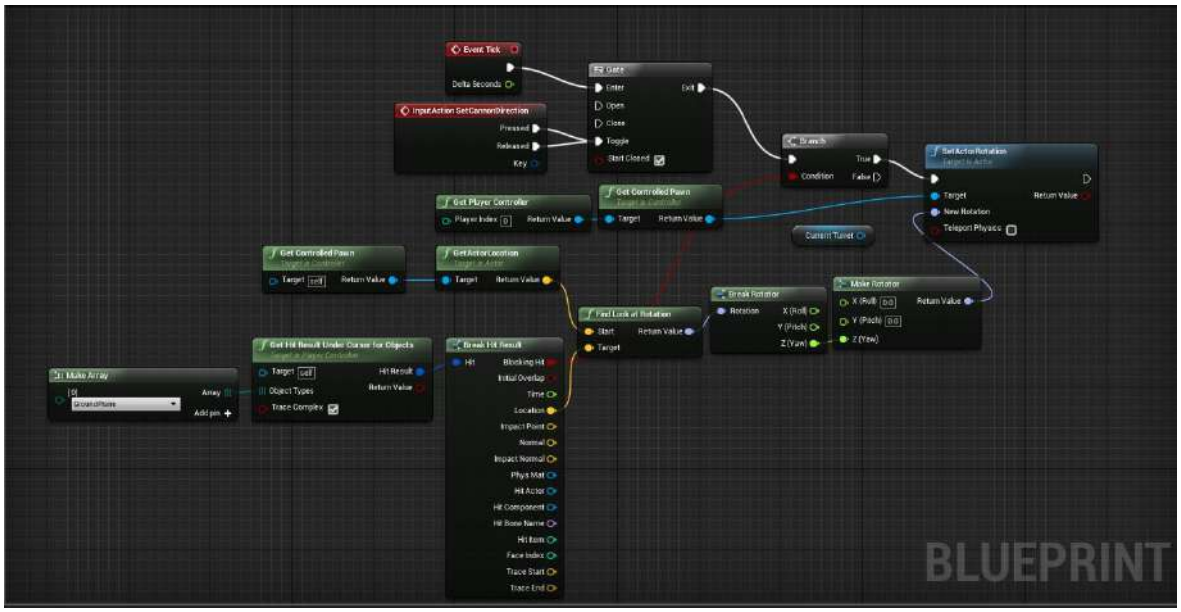
[1]

Le funzionalità core tipicamente fornite da un game engine possono includere: un rendering engine per grafica 2D o 3D, un phisics engine (o collision detection), suoni, scripting, animazioni, intelligenza artificiale, networking, streaming, gestione della memoria, threading, localization support, scene graph e supporto video per cinematiche.

[1] Fonte: Sony LevelEditor



Nella maggior parte dei casi, i game engine forniscono una suite di strumenti di sviluppo "visivi" (ovvero una GUI), i quali permettono una realizzazione di videogiochi rapida e semplificata, mediante un approccio data-driven. Tale approccio permette ad alcuni engine di diminuire notevolmente l'ammontare di conoscenza di programmazione richiesta allo sviluppatore e, di conseguenza, di rendere la percentuale di programmazione di codice necessaria per lo sviluppo di videogiochi quasi pari allo 0%. In altre parole, con alcuni game engine anche chi non conosce nessun linguaggio di programmazione e/o ha zero esperienza nel campo può realizzare un videogioco senza dover mai mettere mano a del codice.



1.1.1.2 Game Engine vs Game Framework

Un *Game Engine* è un insieme di strumenti e meccanismi già predisposti (ovvero già presenti all'interno del programma), per la creazione di (un certo tipo di) videogioco.

Un *Framework* è essenzialmente una codebase che gestisce alcuni aspetti importanti di un videogioco, come, solitamente, l'interfacciamento hardware e l'input, ma non è fornito di un insieme predeterminato di regole e/o di una interfaccia grafica.

Ad esempio, l'Unreal Engine è uno shooter engine; al suo interno vi sono già delle regole preimpostate per il movimento, l'input, la fisica, la mappa, lo spawning, ecc.... Inoltre, è già fornito di un map editor e di un asset editor che possono essere utilizzati per soppiantare queste regole con delle proprie, in modo tale da poter realizzare un videogioco proprio come lo si desidera. Questo però non vuole dire che l'Unreal Engine non possa essere utilizzato per sviluppare altre tipologie di videogiochi diverse da quella degli *FPS* (*First-Person Shooter*), ma che semplicemente è stato già ottimizzato con delle regole che permettano uno sviluppo veloce di meccaniche shooter games con modelli e textures dettagliate. La realizzazione delle altre categorie di videogiochi, difatti, si basano proprio sullo giocare con queste ottimizzazioni (quindi, un gioco in cui si è scelto di muoversi ma non di sparare potrebbe diventare, ad esempio, uno *sports game*). È dunque possibile armeggiare con le funzionalità interne di un Game Engine per fare qualcosa di totalmente diverso dalla tipologia di gioco per cui l'engine stesso è stato pensato (come ad esempio un puzzle game). Tuttavia, bisogna tenere a mente che ciò non equivale a sfruttarne a pieno le sue potenzialità (poiché per gli sviluppatori utilizzare un Game Engine sarebbe come avere una buona fetta di game development già fatta).

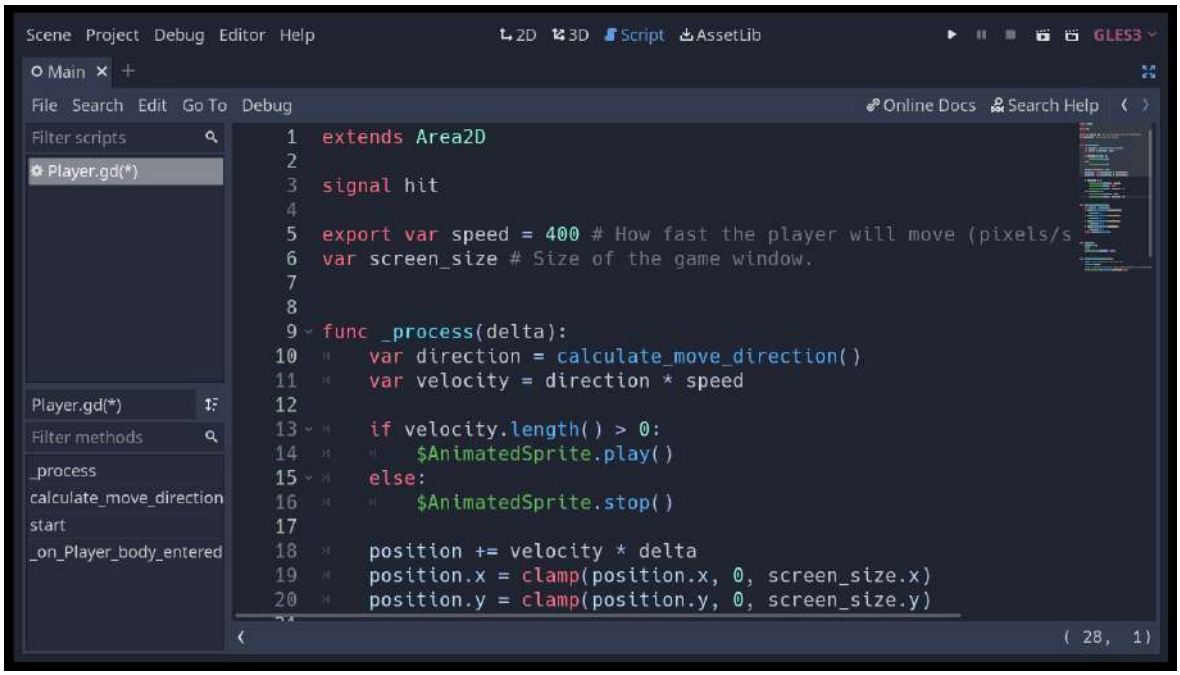
I Frameworks, invece, non sono altro che insiemi di codice che permettono di arrivare "subito" all'aspetto pratico del game making. Un buon Framework tende a unificare vari aspetti di un videogioco, vale a dire: input controllato, grafica, suono, persistenza, e asset management. I Frameworks non sono predisposti per la realizzazione di una certa tipologia di videogioco, così come lo sono invece i Game Engines, ma le loro capacità potrebbero essere adatte per diverse di loro (un framework 2D non riuscirà mai a creare un FPS).

Riassumendo, mentre un Game Engine è un pacchetto completo in cui tutto il lavoro pesante è già stato fatto e lo sviluppatore può semplicemente concentrarsi sul game development vero e proprio mediante l'utilizzo di level/scene editors, strumenti per l'importazione e la gestione di assets (modelli, textures, suoni, sprites, ecc...), un sistema di animazione e un linguaggio di scripting o una API per la logica del videogioco; un Framework è qualcosa che astrae i dettagli della programmazione più a basso livello e si occupa della creazione della OS-level window in cui verrà eseguito il videogioco, di fornire un metodo diretto per disegnare oggetti su tale finestra, di gestire i vari dispositivi di input e di pensare a un sistema per il controllo del suono. Nient'altro. Tutto il resto bisogna programmarselo da sé.

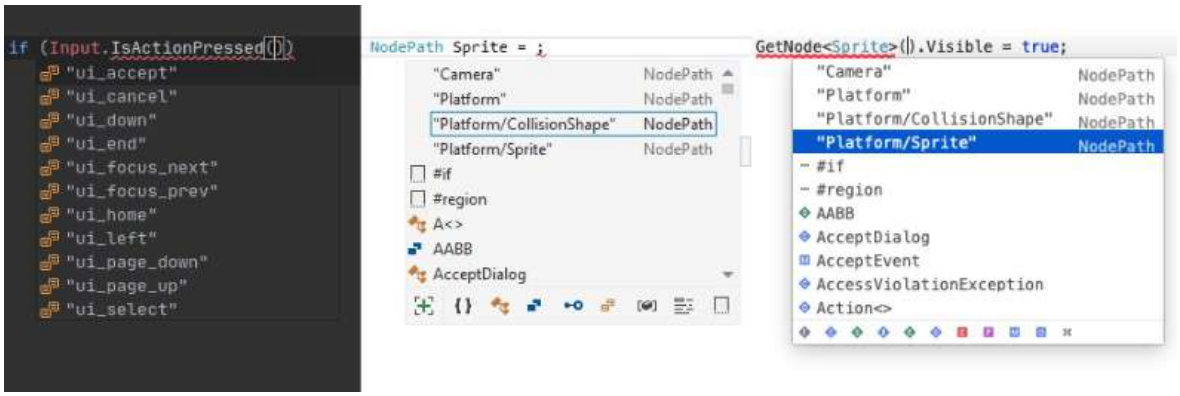
1.1.1.3 Cos'è Godot

Godot è un game engine che offre quattro linguaggi di programmazione, che possono anche essere mischiati tra loro: GDScript, C# e, tramite GDExtension technology, C e C++:

- GDScript è perfetto per chi ha poche esperienze di programmazione: è stato pensato appositamente per Godot e per soddisfare i bisogni degli sviluppatori; ha una sintassi leggera e diretta e, inoltre, tra tutti gli altri linguaggi è in assoluto quello che riesce a comunicare più da "vicino" con il game engine



- C# è uno dei linguaggi preferiti dai game developers e offre un buon trade-off tra performance e semplicità di utilizzo, anche se, tuttavia, bisogna stare attenti al suo garbage collector.



- C e C++ non sono consigliati come linguaggi da utilizzare insieme a Godot. In primo luogo perché possono essere implementati solamente tramite estensioni e, in secondo luogo, perché possono portare a enormi limitazioni durante lo sviluppo; limitazioni così gravi che alcune volte si è addirittura costretti a dover riscrivere l'intero videogioco in un diverso linguaggio di programmazione, semplicemente per poter implementare una determinata meccanica di gioco.

1.1.1.4 Perché imparare Godot?

Godot è gratis, open source ed è, inoltre, la scelta più consigliata per chi non ha mai programmato un videogioco poiché migliora notevolmente la qualità degli sviluppatori che lo utilizzano e aiuta a comprendere concetti di computazione e/o risolvere problemi in GDscript che magari, di solito, non si riescono a implementare facilmente in C#, C o C++.

Pro:

- è in continuo aggiornamento
- è completamente gratuito
- lo scripting è molto più solido rispetto a tanti altri engine

Contro:

- ha una community ristretta
- manca di funzionalità avanzate presenti invece in altri engine come, ad esempio, Unreal Engine

1.1.1.5 Perché non utilizzare Unity?

Unity è il game engine più conosciuto in assoluto, chiunque, anche chi non ha mai provato a programmare un videogioco, nella sua vita avrà sentito nominare la parola "Unity" almeno una volta.

Fino a qualche mese fa (dalla scrittura di questo documento) Unity era considerato il salvatore dell'industria dei videogiochi: era relativamente facile da usare e forniva un engine/framework per la realizzazione di diverse tipologie di videogioco. Nei primi anni 2000 era addirittura superiore a engines utilizzati da team di sviluppo giapponesi.

Ma se davvero è così allora perché non utilizzarlo?

Nonostante Unity vanti molte figure promettenti tra i suoi ranghi, che gli permettono di esibire alcune features strabilianti, il problema sussiste quando questi personaggi lasciano l'azienda e le loro features finisco per diventare obsolete o, addirittura, per cadere a pezzi. Come se non bastasse, l'engine acquista inoltre quei prodotti considerati "competitivi" che poi con il tempo finiscono per non essere più supportati, quando il creatore originale lascia l'azienda. La goccia che ha fatto traboccare il vaso fu però un annuncio[2], ufficiale, i cui punti chiave sono di seguito riportati ed elencati:

- Unity Personal (gratuito) non può essere utilizzato offline
- un qualsiasi livello di licenza Unity richiede ora agli sviluppatori di pagare una tariffa fissa (Runtime fee) di alcuni centesimi per ogni singolo download (ciò significa che se un utente disinstalla un gioco e lo reinstalla, viene contato 2 volte)
- Unity plus, pro ed enterprise presentano ora un costo di iscrizione e molti sviluppatori professionisti sono costretti ad utilizzarli
- la somma in denaro da pagare dipende dal livello di licenza Unity e dai guadagni scaturiti dai videogiochi sviluppati con l'engine

Il punto dunque è questo: Unity sta già facendo pagare una licenza, e adesso chiederà anche una tassa fissa a installazione. Che senso ha allora acquistare una licenza se poi c'è anche dell'altro da pagare?

Leggendo questa prima parte verrebbe da dire che, in realtà, Unity ha da poco cambiato nuovamente i loro terms and conditions. Si, è vero, in questo articolo[3], in cui sommariamente si dice che:

- Unity Personal rimarrà gratuito
- non vi sarà una Runtime fee per i giochi sviluppati con Unity Personal
- Il tetto di Unity Personal di avere un guadagno di $100,000 negli ultimi 12 mesi sarà incrementato a $200,000 e verrà anche rimosso l'obbligo di utilizzo dello splash screen "Made with Unity"
- nessun videogioco con meno di $1 milioni di ricavi su 12 mesi sarà soggetto alla Runtime fee

[2]https://blog.unity.com/news/plan-pricing-and-packaging-updates?ref=insertcredit.com
[3]https://blog.unity.com/news/open-letter-on-runtime-fee

- le nuove politiche di prezzo verranno applicate a partire dalla prossima versione LTS di Unity, che uscirà nel 2024 o oltre (progetti e/o videogiochi che usciranno prima e/o che sono già usciti non saranno soggetti a tasse a meno che non vengano aggiornati con la nuova versione di Unity)

- per i videogiochi soggetti alla Runtime fee è stata data la scelta di o 2.5% dei guadagni oppure il totale ricavato a partire dal numero di nuove interazioni di persone con il gioco. Entrambi i numeri dovranno tuttavia essere riportati personalmente a partire dai dati disponibili e, inoltre, verrà richiesta sempre la somma minore.

La domanda dunque sorge spontanea: ora che si fa? Si continua a programmare videogiochi in Unity?

Molto probabilmente no. I nuovi cambi nei prezzi, che tra l'altro sono il frutto delle azioni di una community unita e non prese direttamente da Unity, citando Rami Ismail su X, "*ci terranno al sicuro fino a quando non capiremo a cosa passare*". Unity può, e cambierà le sue regole, in qualsiasi momento converrà a loro; non vi è alcuna sicurezza che Unity rispetterà le promesse prese. Questo è, al momento, il pensiero della community. La fiducia è stata tradita, molti sviluppatori stanno già passando a prodotti open source.

Per concludere, Unity non risentirà gli effetti delle loro azioni nell'immediato futuro, ci vorrà sicuramente qualche anno, tuttavia, è per questi motivi che riteniamo che, se stiamo cercando di sviluppare videogiochi, e stiamo iniziando adesso, con uno sguardo rivolto al futuro, Unity non sia la scelta giusta per il nostro scopo, semplicemente perché, come già detto in precedenza, molto probabilmente tra qualche anno non sarà più utilizzato.

1.1.1.6 Perché non utilizzare GameMaker?

GameMaker è un engine mirato e ideato principalmente per la realizzazione di videogiochi 2D, permettendo un utilizzo out-of-the-box di raster graphics, vector graphics, e 2D skeletal animations, insieme a tutto un pacchetto di librerie standard per la creazione di grafica e figure 2D.

Allora perché non utilizzare GameMaker? GameMaker ha un suo linguaggio di programmazione, il "*GameMaker Language*" (*GML*), particolarmente adatto per i non-programmatori e per la funzionalità di "drag and drop". GML è un linguaggio estremamente lento, ogni forma di data-processing è davvero ridotta all'osso; le sue potenzialità non sono per nulla vicine a quelle di un buon linguaggio di programmazione come C++, Java o C# e, inoltre, è estremamente progettato male per quanto riguarda lo scripting modulare. È proprio per questi motivi che, all'interno della community di game development, GML è visto più come "un giocattolo che si dà a chi non ha davvero mai visto una linea di codice".

Un'ulteriore problematica di GameMaker è il troppo overhead o, in altre parole, l'engine che utilizza è estremamente pesante; ciò è dovuto al fatto che deve essere utilizzato per supportare una vasta gamma di videogiochi e, soprattutto, perché deve supportare eventi "drag e drop".

Per concludere, a causa delle varie limitazioni del linguaggio e al fatto che l'engine sia fin troppo facile da utilizzare, non è una buona idea basare la realizzazione del videogioco su GameMaker. In linea di massima, è sì vero che con GML si possa fare

qualsiasi cosa ma, tuttavia, si andrebbe di sicuro incontro a molte più difficoltà rispetto ad altri linguaggi di programmazione.

1.2 Obiettivi

Questo documento si pone come obiettivo il voler incrementare le capacità di programmazione degli studenti mediante la realizzazione di un videogame 2D della tipologia RPG (Role-playing game). I lettori impareranno difatti a programmare videogiochi in Godot e Javascript utilizzando come basi di partenza per le varie fasi dell'apprendimento proprio i due videogiochi creati nel corso di questo documento.

Si è scelto di sviluppare due videogiochi in modo tale da non "annoiare" il lettore con lo stesso videogioco in entrambe le metodologie di sviluppo (Godot e Javascript). Per realizzazione si intende uno sviluppo completamente da zero: sprites dei personaggi, design dei livelli, algoritmi per il dialogue/combat system, e molte altre feature rientrano difatti tra le competenze che un programmatore di videogiochi deve possedere e, di conseguenza, sono quindi materiale didattico da dover elargire.

1.2.1 Perché si è scelto un videogioco 2D RPG

Gli RPG 2D sono un ottimo punto di partenza per chi vuole iniziare a programmare videogiochi, oltre che a essere una grande sfida per chi vuole accrescere le proprie abilità di programmatore, per chi vuole fare esperienza o, ancora, per chi vuole approcciarsi in qualcosa di totalmente nuovo.

Il 2D è da preferire al 3D poiché non presenta una logica, una fisica e altre meccaniche troppo complesse per i nostri obiettivi.

Si ha scelto la categoria RPG perché è una delle poche tipologie che racchiude in sé la maggior parte delle caratteristiche presenti in tutte le altre. Se si sa programmare un RPG, allora si saprà certamente programmare una qualsiasi altra categoria di videogioco, come, ad esempio, un puzzle-game; questo perché un RPG può avere, al suo interno, puzzle da completare, battaglie a turni, minigame, partite a tempo, ecc... Difatti, qualsiasi tipologia di videogioco che si riesca a pensare può essere benissimo inclusa all'interno di un RPG sotto forma di "quest" (obiettivo) o qualsiasi altra meccanica.

1.3 Videogame language

Come prima cosa bisogna chiarire alcuni concetti dell'ambito videoludico che spesso si usano impropriamente, ovvero senza conoscerne il significato, e/o per sentito dire, senza avere una chiara immagine in mente di cosa si stia parlando e/o a cui ci si sta riferendo.

1.3.1 2D vs 3D

Uno degli aspetti più comuni e discussi dell'ambito videoludico è la grande differenza che vi è tra il 2D e il 3D.

Qualche anno fa, i primi videogiochi erano tutti in 2D, ovvero senza profondità. La mancanza di profondità era un grave "difetto" del 2D, è dunque per questo che si ricorrevano a effetti ottici in oggetti come scale, palazzi, strade o qualsiasi altra entità che necessitasse di una propria profondità per essere riconosciuta. Pensiamoci bene, l'oggetto che noi chiamiamo "scale" o "scalinata" viene inconsciamente associato al pensiero di "salire su" o "scendere giù"; dunque, nel momento in cui nessuna delle due avviene, ci sentiamo traditi dal nostro ricordo dell'oggetto e iniziamo a dubitare sul fatto che quello che ci troviamo davanti sia davvero una "scala". È qui che entra in

gioco la prospettiva: la giusta angolazione può far sembrare che si stia effettivamente salendo o scendendo. Lo stesso concetto si può applicare ai muri: un muro non è "muro" se il nostro giocatore riesce ad attraversarlo!

Al giorno d'oggi, invece, la maggior parte dei videogiochi sono tutti in 3D, e quindi con profondità, e, addirittura, ci si sta muovendo sempre più nella direzione della realtà aumentata. È qui che sorgono domande del tipo: che cosa si intende effettivamente per 2D? Perché alcuni videogiochi continuano ad avere questa grafica? Quali sono le differenze principali tra i videogiochi 2D e quelli 3D?

1.3.1.1 Cosa sono i videogiochi 2D?

I videogiochi 2D (dall'inglese "2 Dimensions", ovvero "2 Dimensioni") sono "piatti", formati da sprites ("disegni") che possono muoversi attraverso lo schermo solo in alto, in basso, a destra o a sinistra. Generalmente, questo tipo di movimento limitato rende i videogiochi 2D molto più accessibili rispetto a quelli 3D.

Se ci si è mai posti la domanda su come rendere i videogichi 3D più accessibili, buttare un occhio su alcuni videogiochi 2D può essere fonte di ispirazione: chiunque può giocare un videogiochi 2D di Pokémon, ma la maggior parte delle persone trova difficilissimo anche solo comprendere i comandi di un gioco come Dark Souls 3.

1.3.1.2 Cosa sono i videogiochi 3D?

I videogiochi 3D (dall'inglese "3 Dimensions", ovvero "3 Dimensioni") consentono il movimento su un piano tridimensionale, permettendo così al giocatore di esplorare la mappa a 360° e proseguire per una qualsiasi direzione che si preferisce. È proprio questo difatti il grande punto forte dei videogiochi 3D: la libertà nell'esplorazione.

Sotto il punto di vista di uno sviluppatore, la creazione di un videogiochi 3D è molto più complessa rispetto a quella di un videogioco 2D. Anche se si tratta solo di una dimensione in più, bisogna comunque considerare aspetti come camera systems, textures, modelli, luci e ombre, oltre che riempire qualsiasi nicchia o pertugio di un mondo virtuale completamente esplorabile.

1.3.1.3 Quali sono i vantaggi che i videogiochi 2D possiedono rispetti a quelli 3D?

Una volta chiarite le maggiori differenze esistenti tra i videogiochi 2D e quelli 3D passiamo ora ad elencare alcuni dei vantaggi un po' più tecnici che i primi possiedono rispetto ai secondi:

1. Tempo di caricamento ridotto e dimensione dei file minori: i videogiochi 2D consumano molto meno spazio su disco e caricano più rapidamente; queste caratteristiche li rendono dunque i candidati perfetti per videogiochi mirati ai dispositivi mobili e sistemi low-end

2. Focus sul gameplay: i videogiochi 2D prioritizzano le meccaniche di gameplay rispetto a grafiche elaborate, enfatizzando così lòa creatività e l'innovazione nel game design

3. Stile artistico unico: i videogiochi 2D offrono ai designers una maggiore flessibilità nel creare delle estetiche visibilmente più uniche e distintive, consentendo così agli artisti di sperimentare con diversi approcci stilistici

Sono proprio questi vantaggi, insieme a tanti altri come il fattore "nostalgia", a contribuire nella popolarità sempre più crescente dei videogiochi 2D.

1.3.1.4 Quali sono le differenze di gameplay tra i videogiochi 2D e quelli 3D?

Le differenze che vi sono tra il gameplay dei videogiochi 2D e quello dei videogiochi 3D sono significative e impattano sulla player experience in diversi modi:

1. Movimento: come detto in precedenza, nei videogiochi 2D il movimento è tipicamente ristretto al semplice muoversi a sinistra, destra, sopra e sotto, con movimenti diagonali occasionali. Dall'altro lato, i videogiochi 3D offrono movimenti più fluidi e tridimensionali, consentendo al giocatore azioni come saltare, scalare, nuotare ed eseguire una vasta gamma di azioni complesse.

2. Controlli: generalmente, i videogiochi 2D hanno control schemes più semplici dovuti alla loro natura bidimensionale, spesso includendo movimenti base come sinistra, destra, salto e attacco. In contrasto, i videogiochi 3D richiedono controlli più complessi che permettano di controllare il proprio personaggio all'interno di uno spazio tridimensionale, il che può essere molto difficile per i giocatori novizi.

3. Meccaniche di gioco ("Gameplay Mechanics"): i videogiochi 2D spesso sono incentrati sul platforming, il puzzle-solving e azioni a scorrimento laterale, enfatizzando molto sul timing preciso e sulla percezione spaziale. I videogiochi 3D, invece, offrono più libertà di movimento e di esplorazione, consentendo ai giocatori di interagire con gli oggetti del mondo virtuale in un modo più dinamico.

4. Immersione: i videogiochi 3D hanno l'abilità di creare un'esperienza molto più immersiva per i giocatori, grazie alla profondità aggiunta e al realismo dell'ambiente. La grafica dettagliata e le tecniche di animazione avanzate dei videogiochi 3D possono difatti far sentire i giocatori come se fossero davvero all'interno del mondo virtuale, soprattutto se si parla di visori e realtà virtuale. Di contro, i videogiochi 2D sono spesso molto più incentrati sulle meccaniche di gioco e sulle sfide piuttosto che creare un'esperienza completamente immersiva.

5. Realismo visivo: mentre i videogiochi 2D si affidano a una grafica più semplice e stilizzata, che è anche meno dispendiosa ma allo stesso tempo può essere accattivamente in un modo tutto suo, i videogiochi 3D renderizzano ambienti di gioco altamente dettagliati e realistici, oltre che personaggi che accrescono significativamente l'esperienza di gioco. Il realismo visivo nei videogiochi 3D contribuisce a creare un mondo molto più intrigante e visualmente "appetibile" per i giocatori.

Queste differenze nelle meccaniche di gioco tra i videogiochi 2D e quelli 3D provvedono a soddisfare diverse preferenze dei giocatori e contribuiscono nelle molteplici esperienze di gioco offerte da ciascun formato.

In conclusione, la distizione tra i videogiochi 2D e quelli 3D si estende ben oltre i loro stili grafici, includendo perfino le maccaniche di gioco, i fattori di immersione e lo sviluppo tecnologico. Entrambi gli stili offrono esperienze uniche che soddisfano le diverse preferenze che esistono tra i giocatori, che stiano essi cercando una semplicità nostalgica oppure un realismo immersivo. Tramite la comprensione delle differenze che

esistono tra le due dimensioni del gaming, i giocatori possono apprezzare la diversità dell'espressione creativa ed esplorare nuovi orizzonti dei mondi virtuali.

1.3.3 Software release life cycle

Chiunque sia appassionato ai videogiochi, ne abbia giocato almeno uno o che abbia letto tutte le notizie riguardanti un videogiochi che doveva uscire si sarà imbattuto almeno anche solo una volta nei termini "Pre-alpha", "Alpha" e "Beta".

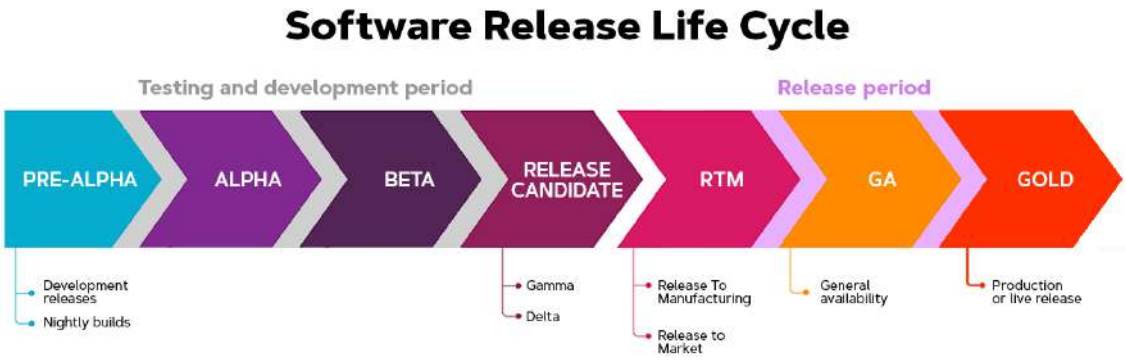
Pensiamo ai famosissimi beta-testers, parola diventata di uso comune anche grazie al popolare anime "Sword Art Online", in cui il protagonista era proprio un beta-tester. I beta-tester sono un gruppo di professionisti e/o semplici utenti che si occupano di provare e collaudare un videogioco non ancora pubblicato, con lo scopo di trovare eventuali errori (bug). Questo videogioco non ancora pubblicato si dice essere in fase "beta", da cui il nome beta-tester.

Ma cosa è esattamente questa versione "beta"? E la versione "alpha" e "pre-alpha" invece? Vengono prima o dopo? Qual è il giusto ordine?

Spesso si fa grande confusione tra questi termini e, soprattutto, vengono usati in maniera impropria. In questo paragrafo ci occuperemo di chiarire meglio cosa sono, il loro ordine, cosa rappresentano e perché rivestono un ruolo così importante nello sviluppo dei videogiochi.

Le fasi "pre-alpha", "alpha" e "beta" sono stadi di un processo produttivo chiamato "Software release life cycle" che, in realtà, non si applica solo ai videogiochi, ma a qualsiasi tipologia di software.

Il "Software release life cycle" è dunque un processo di sviluppo, testing e distribuzione di un prodotto software (come ad esempio un videogioco appunto o un sistema operativo). Esso è tipicamente costituito da diversi stadi, quali pre-alpha, alpha, beta e release candidate (letteralmente "candidati per il rilascio"), prima che la versione finale, o "gold", venga rilasciata al pubblico.



Per fornire un'idea generale di alcune delle fasi principali del "Software release life cycle", è utile paragonare gli stadi di sviluppo a come un artista potrebbe procedere per creare un'opera.

[5]

Come prima cosa viene fatto un abbozzo (Pre-Alpha), poi, con il tempo vengono aggiunti altri dettagli e colori (Alpha e Beta) fino ad arrivare al prodotto finito (Rilascio).

Gli sviluppatori hanno imparato nel corso degli anni che più sono le persone coinvolte nel testing che forniscono feedback come il processo di sviluppo evolve, più il prodotto finale è migliore.

1.3.3.1 Pre-alpha

Con il termine "Pre-alpha" si indicano tutte quelle attività eseguite durante il progetto software prima del testing formale. Queste attività possono includere l'analisi dei requisti, il software design, lo sviluppo software e lo unit testing. In un tipico ambiente di sviluppo open source, vi sono diverse tipologie di versioni pre-alpha. Le versioni *Milestone* includono specifici insiemi di funzioni che vengono rilasciate appena una feature viene completata.

In altre parole, con "Pre-alpha" si intende un software che non è ancora feature-complete e che di solito non viene rilasciato al pubblico. Spesso gli sviluppatori stanno ancora decidendo quali feature il software deve avere.

Il testing "Pre-alpha" viene principalmente eseguito "in casa" dal team di sviluppo. Questo è il primo stadio in assoluto del "Software release life cycle".

1.3.3.2 Alpha

La fase alpha è la prima fase del software testing (alpha è la prima lettera dell'alfabeto greco, utilizzata come il numero 1).

In questo stadio, generalmente gli sviluppatori testano il software utilizzando techniche white-box (metodologia che testa la struttura interna o il funzionamento di un

[5] Fonte: Ashes Of Creation General Discussion

applicativo). Una validazione aggiuntiva viene poi eseguita mediante techniche black-box (metodologia che esamina le funzionalità di un applicativo senza "sbirciare" alla sua struttura o funzionaemnto interno) o gray-box (una combinazione di white-box testing e black-box testing che mira a cercare, se ve ne sono, i difetti dell'applicativo dovuti a una struttura o un uso improprio di quest'ultimo) da un altro team di testing. Spostarsi al black-box testing all'interno dell'organizzazione è conosciuto come *alpha release*

Un software alpha non è meticolosamente testato dagli sviluppatori prima che venga rilasciato ai clienti. Un software alpha può contenere seri errori e, inoltre, qualsiasi instabilità risultante potrebbe portare a crash o perdita di dati.

Un software alpha potrebbe non contenere tutte le feature pianificate per la versione finale. In generale, la disponibilità esterna di software alpha non è comune per software proprietari, mentre, spesso, software open source hanno versione alpha pubblicamente disponibili.

La fase alpha termina solitamente con un "feature freeze" (letteralmente "congelamento delle feature"), che sta ad indicare che nessun altra feature verrà aggiunta al software da quel momento in poi. A questo punto, il software è detto feature-complete. Un beta test viene eseguito subito dopo un collaudo presso il fornitore (l'alpha test) e immediatamente prima del rilascio generale del software come prodotto.

Riassumendo, con il termine "Alpha" si intende dunque il software in uno dei suoi stadi primordiali, quando si trova in uno stato abbastanza stabile da facilitare il testing limitato al di fuori del team di sviluppo principale.

1.3.3.2.1 Feature-complete

Una versione feature-complete (FC) di un pezzo di software vede implementate tutte le feature pianificate o primarie ma non è ancora finita a causa di bug e problemi di performance o stabilità. Durante lo sviluppo, ciò si verifica alla fine dell'alpha testing.

Di solito, software feature-complete devono ancora essere sottoposti a beta testing e bug fixing, così come miglioramenti di performance o stabilità prima di poter diventare un "release candidate", e finalmente raggiungere il "gold status".

1.3.3.3 Beta

Beta, così chiamato dalla seconda lettera dell'alfabeto greco, è la fase di sviluppo software che segue la fase alpha. Una fase beta inizia di solito quando il software è feature-complete ma molto probabilmente contiene diversi bug noti o non noti. Software nella fase beta avranno generalmente molti più bug al loro interno rispeto al software completato e problemi di velocità o di performance posso ancora causare crash o perdite di dati.

Il focus del beta testing è quello di ridurre gli impatti sugli utenti, di solito incorporando test di usabilità. Il processo di consegnare una versione beta agli utenti è detto *beta release* ed è tipicamente la prima volta che il software viene reso disponibile al di fuori dell'organizzazione che lo ha sviluppato. Le software beta release possono essere sia aperte che chiuse, a seconda del fatto di essere apertamente disponibili oppure disponibili solamente a una cerchia ristretta di pubblico.

Le versioni beta di un software sono spesso utilizzate per le dimostrazioni e le preview all'interno di una organizzazione e ai possibili clienti. Alcuni sviluppatori si riferiscono a questo stadio come *preview*, *preview release*, *prototype*, *technical preview* o *technology preview (TP)*, o *early access*.

Come già accennato in precedenza, i *beta tester* sono quelle persone che riportano attivamente i problemi riscontrati con la versione beta. Essi sono di solito clienti o rappresentanti di eventuali clienti dell'organizzazione che sta sviluppando il software. I beta tester tendono a offrire i loro servizi senza ricevere in cambio un compenso monetario ma ricevono spesso una versione del prodotto che testano, sconti sul prezzo di rilascio o altri incentivi.

Per versione beta si intende, dunque, a una versione del software più "raffinata", che contiene la maggior parte delle feature ma non è ancora pronta per le prime iterazioni. La fase beta è l'ultimo maggior stadio di testing prima del rilascio e spesso coinvolge una vasta player base in modo tale da testare i sistemi di gioco sotto stress e trovare ogni bug possibile o problemi di performance sfuggiti alla fase alpha.

1.3.3.3.1 Perpetual beta

Alcuni software sono tenuti in un cosiddetto stato di *perpetual beta*, dove le nuove feature sono continuamente aggiunge al software senza fissare mai un rilascio finale "stabile".

1.3.3.3.2 Open and closed beta

Gli sviluppatori posso rilasciare o una *closed beta* o una *open beta*; le versioni closed beta sono rilasciate a un gruppo ristretto di individui per un test utente su invito, mentre gli open beta tester provengono da gruppi più ampi, o comuqnue comprendono qualsiasi persona sia interessata.

Le closed beta (o *private beta*) possono essere adatte per quella tipologia di software che è capace di fornire un certo valore ma non è ancora pronta per essere usata da tutti o a causa di problemi di scala, mancanza di documentazione oppure perché manca ancora di feature vitali. I tester riportano qualsiasi bug che riescono a trovare, e spesso consigliano feature aggiuntive che pensano dover essere presenti nella versione finale.

Le open beta servono al doppio scopo di dimostrare un prodotto a potenziali clienti e, contemporaneamente, a testarlo all'interno di un'ampia user base in modo tale da fare in modo di portare alla luce errori nascosti che non sarebbero riusciti a trovare un gruppo molto più ridotto di tester.

1.3.3.4 Release candidate

Un *release candidate (RC)*, noto anche con il nome di *gamma testing* o "*going silver*", è una versione beta con il potenziale per diventare un prodotto stabile, pronto per essere rilasciato a meno della comparsa di bug significativi.

In questo stadio di stabilizzazione del prodotto, tutte le feature che devono far parte del software sono già state progettate, programmate e testate mediante una o più cicli beta senza nessun noto bug bloccante.

Una release è detta *code complete* quando il team di sviluppo concorda che nessun codice sorgente completamente nuovo verrà aggiunto a quest'ultima. Potrebbero, tuttavia, ancora esservi cambiamenti nel codice sorgente mirati ad aggiustare difetti, alla documentazione, ai data file e al codice esterno per i test case o le utility.

I beta tester, se selezionati privatamente, verranno spesso citati per aver usato la release candidate come se fosse il prodotto finito. Il beta testing viene condotto presso la sede del cliente o dell'acquirente e per testare il software dal punto di vista dell'utente.

1.3.3.4.1 Stable release

Una *stable release* (letteralmente "rilascio stabile"), anche detta *production release*, è l'ultimo release candidate (RC) che ha passato tutti gli stadi di verifica e di test. Qualsiasi bug rimanente conosciuto è considerato accettabile. Questa release viene mandata in produzione.

Alcuni prodotti software presentano anche *long-term support (LTS)* release che sono basate su full release che sono già state provate e testate e ricevono solamente aggiornamenti di sicurezza. Questo permette agli sviluppatori di impiegare più tempo allo sviluppo del prodotto, invece di aggiornare il codice o trovare e risolvere nuovi bug introdotti a causa di assunzioni datate riguardanti il sistema usato, la lingua o le librerie sottostanti.

Una release o *live* o *production release* è dunque il prodotto finale pronto per essere rilasciato al pubblico. La release finale è tipicamente scelta dagli sviluppatori da un insieme di release candidate che sono stati costruiti dopo lo stadio beta. Anche se una volta rilasciato, il software è generalmente noto come "stable release", il termine formale spesso dipende dalla metodologia di rilascio e viene diviso in: physical media, online release, o web application (o web app).

1.3.3.5 Release to manufacturing (RTM)

Il termine "*release to manufacturing*" (RTM), anche noto come *going gold*, è un termine utilizzato quando un prodotto software è pronto per essere consegnato. Questa build può essere firmata digitalmente, consentendo all'utente finale di verificare l'integrità e l'autenticità dell'acquisto software. Un copia della build RTM conosciuta con il nome di "*gold master*" o GM viene inviata per la produzione di massa o per la duplicazione disco, se applicabile. La terminologia è presa dall'industria della registrazione audio, specificamente al processo di masterizzazione. RTM precede la general availability (GA) (letteralmente "disponibilità generale") quando il prodotto viene rilasciato al pubblico. Una build gold master (GM) è tipicamente la build finale di una parte di software che si trova nello stadio beta per gli sviluppatori.

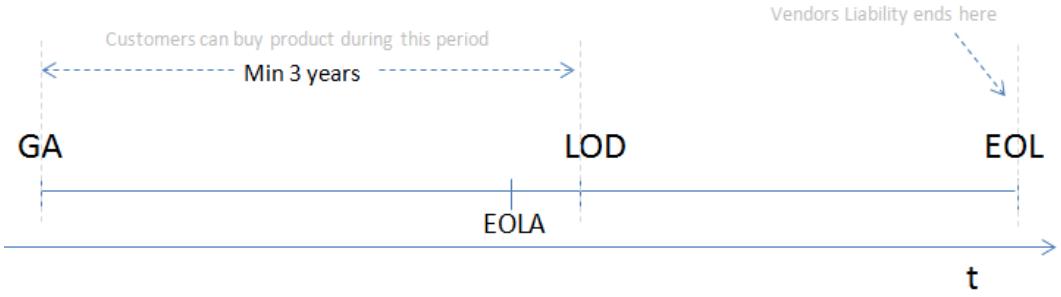
RTM è tipicamente usato in certi contesti per la produzione di massa di software al dettaglio, in opposizione a una produzione di software specializzato o progetto in una produzione e distribuzione commerciale o governamentale, dove il software viene venduto come parte di un bundle in una vendita di computer hardware correlata e tipicamente dove il software e il correlato hardware devono essere disponibili e venduti in massa/pubblico presso negozi al dettaglio per indicare che il software ha raggiunto una livello di qualità definita ed è pronto per la distrubuzione di massa al dettaglio.

In altri contesti, RTM può inoltre star a significare che il software è stato consegnato o rilasciato al cliente o all'acquirente per l'installazione e la distribuzione dell'hardware correlato nei computer o nelle macchine dell'utente finale. Il termine non definisce il meccanismo di consegna o il volume, ma stabilisce solamente che la qualità è sufficiente per la distribuzione di massa. Il prodotto finale dell'organizzazione ingegneristica è frequente nella forma di un golden master media usato per la duplicazione o per produrre l'immagine per il web.

1.3.3.6 General availability (GA)

Per *general availability (GA)* si intende quello stadio di marketing dove tutte le attività di commercializzazione necessarie sono state completate e ed è già disponibile all'acquisto

un prodotto software, a seconda, tuttavia, dalla lingua, la regione, e le disponibilità elettroniche o di media. Le attività di commercializzazione potrebbero includere la sicurezza e le prove di conformità, così come la localizzazione e la disponibilità globale. Il tempo necessario per passare da una RTM a una GA dipende dal tempo impiegato per completare tutte le attività di commercializzazione richieste da una GA. Prima che una generally available release possa essere annunciata, possono trascorrere dunque dai singoli giorni a svariati mesi. Il software che ha raggiunto questo stadio è detto *gone live*.



[6]

1.3.3.7 Release to the Web (RTW)

Una *release to the Web (RTW)* o *Web release* è un mezzo di consegna del software che utilizza l'internet per la distribuzione. Questo tipo di meccanismo di rilascio non prevede la produzione di supporti fisici da parte del produttore.

Le web releases sono diventate molto più comuni con la crescita e la diffusione di Internet.

1.3.3.7.1 Support

Durante il periodo di vita supportato, il software è spesso soggetto a servizi di rilascio, patch o service pack, a volte chiamati "*interim releases*" o "*maintenance releases*" (MR). Ad esempio, Microsoft rilasciò tre service pack maggiori per l'edizione a 32-bit di Windows XP e due service pack per le edizioni a 64-bit. Questi servizi di rilascio contiene una collezione di aggiornamento, fix, e migliorie, consegnate sotto forma di un singolo pacchetto installabile. Questi possono anche implementare nuove feature.

Alcuni software vengono rilasciati con l'aspettativa di un supporto regolare. Classi di software che generalmente comprendono un supporto prolungato come norma sono le suite di anti-virus e i massivi giochi online multiplayer. Continuando sulla falsa riga dell'esempio di Windows XP, Microsoft effettivamente offrì aggiornamenti a pagamento per ulteriori 5 anni dopo il termine dell'esteso periodo di vita supportato.

1.3.3.7.2 End-of-life (EOL)

Quando il software non viene più venduto o supportato, il prodotto è detto aver raggiunto la fine della sua vita (end-of-life), per essere interrotto, ritirato, deprecato, abbandonato o reso obsoleto, ma la fedeltà degli utenti potrebbe prolungarne l'esistenza per qualche tempo, addiruttura anche molto dopo che la piattaforma diventi obsoleta.

[6] Fonte: Wikipedia

Solitamente, dopo la data di end-of-life, gli sviluppatori non implementeranno più nessuna nuova feature, né risolveranno problemi esistenti, bug o vulnerabilità (siano esse note prima di questa data o no), o provvederanno qualsiasi supporto per il prodotto. Se gli sviluppatori lo desiderano, potrebbero addirittura rilasciare il codice sorgente, così che la piattaforma possa continuare a vivere ed essere mantenuta da volontari, oppure, essere reverse-engineered in un secondo momento, quando sarà diventata *abandonware*, ovvero abbandonata dal proprio produttore e manutentore.

1.3.3.7.3 End of life announcement (EOLA)

L'*end of life announcement* (EOLA) è l'inizio dell'end-of-life. L'EOLA precede il last order date (LOD) di almeno 90 giorni. I clienti possono continuare a ordinare il prodotto fino alla last order date; superata questa data esso non sarà più disponibile.

1.3.3.7.4 Last order date (LOD)

La *last order date* (LOD) è la data prima della quale i clienti possono ancora comprare il prodotto. Dopo questa data, il supporto principale viene terminato.

1.3.4 Game Experience

Il concetto di *Game Experience* (letteralmente "esperienza di gioco"), non comprende solo il gameplay, ma il pacchetto totale di quello che è il gioco e quello che va il gioco a influenzare. È il gioco stesso, la sua presentazione, il suo stile, le interazioni con la community, la natura della community stessa, e il modo in cui il gioco impatta sull'interpretazione che i giocatori hanno di altri videogiochi, media e del resto del mondo.

La qualità della game experience non è qualcosa che può essere determinato e rilasciato il giorno di rilascio di un videogioco come invece si fa con i punteggi delle recensioni; basti pensare che le esperienze dei videogiochi di 30 e 40 anni fa vengono scritte ancora oggi.

Tutto ciò che è stato detto va a intrecciarsi con la (sempre in evoluzione) definizione di videogioco. Sul mercato si vedono molti videogiochi che non meritano di essere chiamati tali, ma che comunque non presentano nessun altro modo per essere descritti. Forse questo è un altro modo per guardare a questi giochi, ovvero non a "giochi" nel verso senso della parola, ma più come game experience: la parola "esperienza" potrebbe difatti essere il termine più adatto per descrivere questi "giochi-non-giochi". Questi ultimi possiedono, difatti, un certo grado di controllo interattivo del giocatore, che può ricordare quello dei videogiochi, ma questo è tutto ciò di simile che c'è tra le due cose, non c'è più nient'altro. Il resto è semplicemente un mondo costruito attorno una natura interattiva. È un modo interattivo per vivere una storia, ammirare dell'arte, sedersi e andare in giro senza pensieri, dimenticandosi momentaneamente dei proprio problemi. Come si può intuire, lì da qualche parte, c'è effettivamente un qualcosa che ricorda vagamente un videogioco ma, tuttavia, quello in cui si è davvero coinvolti, più di ogni altra cosa, è la game experience.

1.3.4.1 Esempi: Game vs Experience

Ovviamente, "game experience" non è una definizione completamente valida per i quasi-giochi. Tutti i videogiochi hanno la proprio game experience: la chiave consiste

nell'enfatizzare ciò che ci cattura interesse.

In questo caso, quale sarebbe la definizione di "gioco"? Pensiamo a qualcosa come il Tetris. Il gioco fa certamente parte di una game experience più grande, tuttavia, quando si gioca a Tetris, cioè che tiene incollati alla sedia è il gioco stesso. Il processo di ruotare pezzi, piazzarli con cura e lasciarli cadere è un processo di gameplay; è ciò che spinge il giocatore ad andare avanti ma è anche ciò che porta lo stesso a rigiocare il gioco. Quando si gioca a Tetris, dunque, si è principalmente attratti dal gioco.

Poi però vi sono quei videogiochi che possono essere meglio descritti come game experience. Un gioco come Heavy Rain rientra perfettamente in questa categoria. Il giocatore si fa strada attraverso eventi di gioco e partecipa attivamente al processo ma, la forza motrice principale che si nasconde dietro Heavy Rain è la storia. Nella maggior parte dei casi, i giocatori non lo giocano principalmente per completare obiettivi o muoversi all'interno del mondo di gioco: lo giocano per vedere gli eventi della storia, proprio come si fa con un film. La storia è una parte importante di un videogioco ed è direttamente connessa al gameplay, il quale non è, dunque, la fonte principale di intrattenimento. Si può dire quindi che il giocatore è coinvolto nella game experience piuttosto che nel gioco in sé per sé. Proteus, per fare un altro esempio, consiste solamente nel camminare da posto a posto. Gli obiettivi lasciano il tempo che trovano. Adesso, a meno che non si trovi qualcosa di intrigante nell'atto del camminare, Proteus non è qualcosa a cui si è attratti per il gameplay, ma per la game experience. Il creare le proprie motivazioni per camminare o anche il semplice domandarsi "Perché sto perdendo il mio tempo così?" sono esempi lampanti di game experience.

Quando si pensa ai concetti di "gioco" e "game experience" in questo modo, si comincia a incappare in molta soggettività. Il sentirsi più coinvolto nel gioco (e quindi nel gameplay) o nella game experience dipende prevalentemente dall'approccio individuale che si ha con il videogioco. Pensiamo a The Last Of Us. Molte persone lo giocano per il gameplay; sono ferrei nel voler migliare le proprie abilità, specializzarsi nelle meccaniche di gioco ed essere quanto più bravi possibili. Altri, invece, lo giocano come se stessero guardando un bel film. La storia li immerge completamente e li spinge ad andare avanti: lo vivono dunque come game experience.

1.3.4.2 Connessioni tra game e game experience

Secondo quanto detto in precedenza, la game experience è sempre più grande del gameplay; il gioco è sempre contenuto all'interno della game experience. Tuttavia, entrambi sono nulla se non vi è una "motivazione" che spinge il giocatore a progredire nel gioco. La motivazione ad andare avanti da un evento all'altro del gioco è un altro elemento chiave del gameplay. È qui che sorgono dunque una serie di domande interessanti: se si è più motivati dalla game experience rispetto al gioco stesso, non significa dunque che la game experience è parte del gameplay? Ma ciò non significa quindi che si è più motivati dal gameplay che dalla game experience? Ma dunque la game experience è ancora più ampia? Ma allora si è più motivati dal gioco o da questa nuova game experience? Se si guarda a tutte queste domande, la relazione che sussiste tra il gioco e la game experience è un qualcosa di ciclico. Se si gioca un videogioco spinti da un senso di nostalgia, allora è la game experience che sta portando il giocatore a raggiungere l'obiettivo del gioco, ovvero del gameplay.

1.3.4.3 Game Experience come prodotto

Alla luce di tutto ciò, risulta dunque molto difficile tracciare una linea netta che separa i giochi dai quasi-giochi. Mentre è certamente vero che una certa parte di game experience viene sperimentata al di fuori del software, un'altra parte può essere creata come parte integrante del gameplay ed essere inclusa nel pacchetto. Quando si inizia per la prima volta il gameplay di Amnesia: The Dark Descent, un wall of text suggerisce al giocatore di non giocare il gioco con l'obiettivo di vincere ma bensì con quello di immergersi nella storia e nell'ambientazione. I fan consiglieranno di giocare il gioco non perché è divertente o difficile ma bensì perché è spavetnoso, viscerale e semplicemente intrigante. In questo senso, Amnesia si pubblicizza direttamente (e indirettamente) come game experience e non come un gioco nel senso di gameplay puro.

Riassumendo, alcuni giochi sono costruiti appositamente per il gameplay, mentre altri per guardare oltre. Nessuno dei due approcci è necessariamente migliore dell'altro ed entrambi contengono elementi condivisi: il "patrimonio" emotivo che lasciano e ciò che attrae e intrattiene il pubblico.

1.3.5 Game development

Per *Video game development*, *Game Development* o a volte abbreviato in *gamedev*, si intende il processo di creazione di un videogioco, dall'idea iniziale fino al prodotto finito. Il game development è una pratica multidisciplinare che coinvolge la programmazione, il design, l'arte, l'audio, la user interface (UI) e la scrittura. Ciascuno di questi ambiti può poi essere a sua volta composto da settori più specializzati; ad esempio, l'arte include la modellazione 3D di oggetti o personaggi, l'animazione, gli effetti visivi e così via.

Lo sviluppo è inoltre supportato, nelle sue vari fasi, dalla gestione del progetto, dalla produzione e dal controllo qualità. I team, invece, possono essere composti da centinaia di persone, un piccolo gruppo o addirittura essere formati da una sola persona.

1.3.5.1 Game design vs. game development

Il game development e il game design, come vedremo in seguito, anche se molto spesso erroneamente usati come sinonimi l'uno dell'altro, sono due concetti completamente diversi:

- con il termine "game design" ci si riferisce al lato concettale delle cose: la vision iniziale, le meccaniche, la storia, i personaggi, le locations, e così via
- il "game development" è invece un termine più generale che comprende il game design e l'implementazione tecnica dei concetti di gioco

In molti piccoli studi di game development, gli stessi membri del team indossano più cappelli e sono responsabili di entrambi i settori. Nelle compagnie più grandi, invece, design e development sono spesso gestiti separatamente.

1.3.5.2 Il processo di game development

Il processo di game development può essere suddiviso in 4 fasi principali:

1. **Concept Phase** ("fase concettuale"), in cui viene delineata la game experience generale

2. **Pre-Production Phase** ("fase di pre-produzione"), in cui vengono definiti tutti i dettagli del videogioco

3. **Production Phase** ("fase di produzione"), nella quale viene implementato tutto ciò che è stato definito nella Pre-Production Phase.

4. **Post-Production Phase** ("fase di post-produzione"), in cui si monitora il videogioco rilasciato e si sviluppano le espansioni.

1.3.5.2.1 Concept Phase

Nella "*concept phase*" o "*planning phase*", ovvero la fase iniziale del processo, il team di sviluppo definisce, appunto, il concept del gioco, stabilisce una vision e i mission statement, determina il pubblico target, definisce lo scope del progetto e fissa un budget e una timeline.

Un planning adeguato è cruciale per assicurarsi che il team abbia una chiara comprensione degli obiettivi del gioco e per far si che esso possa lavorare in maniera efficiente per creare un prodotto di successo.

Tra le tante domande fondamentali di questo stadio, che necessitano di avere una risposta troviamo:

- In che genere rientra il videogioco?
- Sarà un videogioco 2D o 3D?
- Che tipo di stile grafico utilizzare?
- Che tipo di meccaniche si vogliono includere?
- Chi è l'eroe e chi il cattivo?
- Conviene utilizzare un game engine? Se sì, quale?

In questo stadio, le idee crescono o si riducono di numero, vengono scartate, cambiano e vengono migliorate regolarmente, al punto tale che un concept del giorno prima potrebbe già essere irriconoscibile il giorno successivo.

Questa fase è una delle più importanti del processo di game development. Modificare i concetti fondanti più avanti nel tempo, potrebbe portare a grossi problemi. Per fare un esempio, il videogioco "Bad Fur Day", inizialmente intitolato "Conker's Quest", nacque come un tranquillo 3D platformer adatto ai bambini su stile Super Mario 64, prima che gli sviluppatori scartarono completamente la purezza dei personaggi, durante la produzione, a favore di un tono maturo e satirico. Tale cambiamento ritardò il gioco di ben due anni, e causò molti dissenzi all'interno del team di sviluppo.

1.3.5.2.1.1 Proof of concept

Una volta aver risposto alle domande viste in precedenza, l'ultima parte della concept phase consiste nella creazione di una *proof of concept*, la quale, non solo determina le risorse necessarie per dare vita al videogioco, ma anche come cercare di attirare l'attenzione di un publisher.

La proof of concept deve rispondere alle seguenti domande:

- Quanto costerà realizzare il videogioco?
- Come verranno guadagnati i soldi necessari per la realizzazione?
- Quanto tempo è necessario per realizzare il gioco?
- Si possiedono tutte le skill necessarie per realizzare il gioco o bisogna assumere un team/migliorare le proprie capacità?
- Se si ha bisogno di un team, quante persone? Quanto sarà grande il team e di quali ruoli bisogna occuparsi di persona?
- Come verrà monetizzato il videogioco?
- Su quale piattaforma verrà pubblicato?

Gli *Indie developers*, ovvero game developers indipendenti che non lavorano per grandi studi di sviluppo o publisher come Sega e Ubisoft, spesso ricorrono a metodi alternativi come le campagne crowdfunding o le early access release, per finanziare il proprio videogioco.

1.3.5.2.2 Pre-production Phase

Con la seconda fase del game development, conosciuta come "*Pre-production Phase*", si inizia a entrare veramente nel vivo del progetto. In questo stadio, infatti, si studia il lavoro da fare, creano storyboards e prototipi e si decide quali sono le idee vincenti e quali sono destinate per essere abbandonate.

Nella pre-production phase bisogna valutare ogni singolo aspetto del videogioco sia come entità a sé stante, che come ingranaggio di una macchina più grande:

- gli artisti devono assicurarsi che l'art style e le colour palette si abbinino al tema e al genere del gioco. Ad esempio, se il videogioco è oscuro e sinistro, proprio come il platformer 2D Tamashii, gli artisti non dovrebbero lavorare con colori brillanti o estivi.
- gli sviluppatori (developers) devono scegliere i meccanismi e la fisica del gioco, oltre che alla modalità in cui esso processerà modelli e oggetti. Alcune di queste decisioni necessiteranno dell'opinione degli artisti, scrittori e ingegneri, a seconda se tali decisioni impattino sul copione o sul gameplay.
- gli ingegneri dovranno dire al team quali sono i propri limiti. Il team di scrittura potrebbe ad esempio richiedere un grande finale cinematografico che però non può essere renderizzato dal game engine. Gli sviluppatori, d'altro canto, potrebbero voler utilizzare delle meccaniche che porterebbero a problemi di performance. Questa fase è il momento giusto per delineare questi limiti.

- i responsabili del progetto ("project leads") devono fungere da chiave di volta e cercare di bilanciare le richieste di ciascun team (team di scrittura, team di sviluppo, team artistico, ecc. . .), compiendo le decisioni finali, rimuovendo ostacoli e tenendo tutti sulla stessa pagina.

- gli scrittori devono accordarsi sul copione, i personaggi e il mondo in cui questi vivono. Il copione (la trama) soprattutto avrà un impatto notevole sull'arte, le meccaniche e sull'ambiente che deve essere realizzato.

È dunque in questa fase che le idee iniziano a trovare voce e forma. Con così tante parti in movimento però, bisogna cercare di essere anche un po' flessibili. È molto raro, infatti, che un videogioco superi sia la concept phase che la pre-production phase senza qualche sacrificio.

È importante sottolineare inoltre che, a prescindere dai problemi che si possono riscontrare durante le altre fasi del game development, le decisioni che vengono prese durante la pre-production phase sono le radici che tengono saldo il progetto.

Alcune delle domande più comuni che devono dunque trovare una risposta entro la fine di questa fase sono:

- Qual è l'idea alla base del videogioco e come può essere riassunta in un intricante game pitch?

- Dove e quando è ambientato il gioco?

- Chi sono i personaggi?

Durante la pre-production phase è solito inoltre prototipare gli scenari, i personaggi, i control schemes e altri elementi in-game. Gran parte dell'impegno viene inoltre investito nel worldbuilding. Le idee vengono sviluppate sotto forma di storyboards, concept art, modelli di interfacce, e così via, per farsene un'idea, capire come vengono percepite e come interagiscono l'una con l'altra.

7

1.3.5.2.3 Production Phase

La *production phase* è lo stadio più lungo e importante del processo di game development: è in questa fase che vengono spesi la maggior parte del tempo e del denaro, e il videogioco inizia ad avere veramente vita.

Quando i videogiochi entrano in produzione:

- gli sviluppatori e i designers creano il mondo e costruiscono e programmano le dinamiche ambientali che complementano la storia, la direzione artistica e le meccaniche di gioco.
- vengono disegnati, renderizzati e animati i modelli dei personaggi principali

[7]Fonte: Ubisoft Entertainment

e degli NPCs (dall'inglese "Non-player character" o "Non-Playable Character", letteralmente "personaggio non giocante")

- i voice actor registrano (quasi sempre più di una volta sola) la loro parte di copione e i singoli dialoghi nella giusta tonalità

- i sound designer creano soundtracks e effetti in-game che spaziano dai *beep* del menu, al rumore dei passi del giocatore

- gli scrittori revisionano il copione e si occupano di tutti i compiti di copywriting che saltano fuori, come, ad esempio, dare nomi agli NPCs e fornire una descrizione agli oggetti di gioco.

La production phase è la fase in cui c'è tutto il divertimento, ma è anche quella in cui devono essere prese decisioni importanti e le idee devono passare un ultimo test.

A volte, può però capitare di dover fare qualche piccolo cambiamento. Se si prende Spyro the Dragon come esempio: durante la production phase, l'aerostiere che, nel gioco, porta il giocatore da un mondo a un altro, era in realtà un barcaiolo vichingo.

[8]

Il motivo del cambiamento non fu mai rivelato ufficialmente, tuttavia, sapendo che il mondo di Dream Weavers è un'isola fluttuante nel cielo, molto probabilmente avranno realizzato che una barca non è proprio il modo migliore per raggiungerla.
In altre occasioni, invece, interi segmenti, meccaniche, o personaggi devono essere interamente scartati per il bene del progetto. Durante lo sviluppo di Simpsons Hit & Run, ad esempio, dozzine di macchine, modelli, interni di case, e perfino due interi livelli furono completamente eliminati in modo da alleggerire il carico di lavoro del team di sviluppo.

[8] Fonte: Next Generation Magazine #42

1.3.5.2.4 Post-production phase

Una volta che la versione finale del videogioco viene finalmente rilasciata, si entra nell'ultima fase del processo di game development: la *post-production phase*.

L'obiettivo principale della post-production phase è la manutenzione del gioco, la quale comprende principalmente:

- bug-fixing; nonostante l'impegno dei tester, al momento del rilascio la maggior parte dei videogiochi contiene ancora dei piccoli bug: i primi mesi della post-production phase sono tipicamente trascorsi identificando e risolvendo questi bug

- nuovi contenuti; la post-production phase include anche aggiornamenti software regolari del gioco, che spaziano da patch per il game-balancing, a nuovi DLC (dall'inglese "downloadable content", ovvero, "contenuto scaricabile").

Non esistono due videogiochi uguali e perfino gli studi di game development esperti che hanno alle spalle centinaia di giochi spesso trovano difficoltà nell'affrontare cambiamenti dell'ultimo minuto, strette deadline, differenze creative e altre problematiche. Questa è la natura dell'industria. Vi sono molteplici aspetti che contribuiscono al successo di un videogioco che però non sono sotto il controllo immediato di uno sviluppatore: tali parametri rappresentano ancor di più una ragione più che valida per avere un processo di game development strutturato con deadline e obiettivi di produzione chiari.

1.3.6 Game Direction

Non avere una game direction è un ingrediente perfetto per la ricetta del fallimento. Decisioni scaturite dal potenziale di un videogioco o azioni da intraprendere durante lo sviluppo, non possono essere prese basandosi sull'intuito o sull'istinto. Scegliere una game direction solida, prima di impegnarsi su un'idea di gioco, è il giusto modo per mitigare i rischi e determinare il "perché" di un videogioco. In questo modo, si può delineare un'idea di gioco chiara, comunicarla agli altri e avere così una luce guida che può accompagnarci nelle varie fasi dello sviluppo.

In questo paragrafo, capiremo cosa è esattamente una game direction e che ruolo ricopre all'interno del processo di sviluppo di un videogioco (game development).

1.3.6.1 Le fondamenta del proprio Game Design

Un gioco inizia sempre con un'idea che nasconde un certo potenziale da valutare esplorandola, prima di, se funziona, iniziare a concretizzarla. Tutto questo viene fatto durante la Concept Phase, il cui requisito è proprio la game direction.

Per valutare il potenziale di un'idea bisogna dunque collegare quest'ultima con una solida game direction. Ciò funziona perché, la game direction, rappresenta l'obiettivo che si vuole raggiungere con il videogioco. Una volta che si possiede una game direction completa, infatti, si hanno tutte le informazioni necessarie per poter valutare di trasformare l'idea in un gioco oppure no. Ma non finisce qui: la game direction rimane fondamentale per tutto il game development poiché funge da luce guida nella scelta delle decisioni.

Ma cos'è esattamente la game direction?
La *Game Direction* è la linea di confine della *target game experience* o *target experience*,

la quale, come già detto in precedenza, rappresenta ciò che si vuole ottenere con il videogioco. La game direction comprende quindi i vincoli concreti fino a dove la target experience può spingersi.

La game direction ha due componenti che prese insieme definiscono la target game experience:

1. *Game Pillars*, ovvero quei parametri che delineano e guidano il processo e la realizzazione di un videogioco (esplorazione, combattimento, crafting, stealth, puzzle-solving, farming, gestione dell'inventario, ecc. . .)

2. *Thematic Structure*, ovvero quello che si vuole trasmettere al giocatore

Nessuna delle due componenti presenta inoltre contenuti di gioco, ed è proprio questa la cosa più importante da ricordare: una game direction non deve mai comprendere contenuti di gioco; il suo scopo è semplicemente quello di descrivere un'esperienza che si desidera che il videogioco susciti e non il tipo di gioco che sia in grado di crearla.

1.3.6.2 Una game direction non definisce un videogioco

Una game direction rappresenta uno spazio di possibili giochi. Il numero di giochi in grado di suscitare la target game experience è infinito. Una game direction impone dunque dei vincoli in questo spazio infinito che contiene tutti i giochi possibili immaginabili, e definisce, così facendo, "un sottospazio di giochi possibili". È proprio questo il punto fondamentale da capire: si chiama game direction perché stabilisce dei vincoli all'interno dello spazio di possibili esperienze, fornendo così una "direzione" (direction) su quello che si può fare; è come decorare una stanza di dimensioni limitate: poiché non si può fare tutto quello che si vuole, si è costretti a dover prendere determinate decisioni.

La direzione in cui si è propensi, rappresenta l'insieme di regole che il design può assumere per generare la target game experience senza lasciare lo spazio che si ha delimitato. Ma cosa significa tutto ciò per un game designer? Un game designer si muove liberamente all'interno dello spazio della game direction e non può mai lavorare all'infuori di questo; il che è un buon vincolo poiché, se la game direction è ben construita, consente comunque molto movimento al game designer. Questo è anche il motivo per il quale non si devono mai mettere degli elementi di gioco concreti all'interno di una game direction: farlo equivarrebbe a limitare troppo l'esplorazione creativa e, ancora peggio, porterebbe inevitabilmente la Pre-Production Phase a convergere in un gioco predefinito; mentre tutto quello che si vuole è semplicemente uno spazio che si possa esplorare liberamente, alla ricerca di quel gioco che susciti al meglio la target game experience. Questa è, inoltre, una delle molteplici manifestazioni concrete di come un videogioco sia, in realtà, un mezzo per un fine, che è la game experience.

Si può dire dunque che la game direction dona autonomia e uno scopo al game design.

1.3.7 Game design

Tutti coloro che hanno a che fare con i videogiochi avranno certamente sentito parlare, almeno una volta nella loro vita, di *Game Design*. La maggior parte di queste persone sosterrà, inoltre, di sapere cosa sia, prima di passare poi, quando gli si chiede di spiegarlo, un brutto quarto d'ora alla disperata ricerca di una definizione improvvisata. Non solo

non è sufficiente giocare tonnellate e tonnellate di videogiochi per capire veramente in cosa consista il game design ma, addirittura, a volte non basta nemmeno progettare un gioco per capirlo, dato che molti game designer professionisti fanno ancora fatica a ben descrivere il loro lavoro.

In questo paragrafo, cercheremo di spiegare tutto quello che c'è da sapere sul game design come area di studi, in modo da chiarire, per quanto possibile, le idee sbagliate che si sono create, con il tempo, nella mente delle persone. Si consiglia, inoltre, di rileggere più volte le parti ritenute ostiche da comprendere, farsi delle domande, rispondersi e creare degli esempi mentali, in modo da meglio assimilare i concetti.

1.3.7.1 Che cos'è il game design

Il game design è una branca del *Game development* che mira a strutturare qualsiasi aspetto di una esperienza di gioco ("game experience"). Fare game design vuol dire inventare, definire e descrivere ogni singolo elemento che fa parte di una game experience e le sue interazioni con altri elementi. Qualsiasi cosa si veda, senta o percepisca mentre si gioca un videogioco è dovuta alla decisione di un designer che mirava ad una particolare game experience.

Il game design vive all'interno di un topic più generale del game development, che raggruppa insieme tutti i ruoli necessari per la realizzazione di un videogioco (design, programmazione, arte, suoni, testing, ecc...). Il game design ha inoltre grandi responsabilità, specialmente in altri ambiti del game development.

Il game design è la base di partenza sopra cui ogni singolo altro reparto lavora. Ciascun reparto in carico di qualsiasi altro aspetto del gioco (arte, suoni, programmazione, ecc...) lavora infatti in base a come il game design ha definito gli elementi di gioco. Se si immagina un videogioco come una casa, il game design è la cianografia, il progetto, che illustra come deve essere costruita.
Proviamo a fare qualche esempio. Gli artisti non pensano al design di un personaggio dal nulla, ma prendono bensì ispirazione da aspetti come cosa fa quel particolare personaggio nel videogioco e in quale contesto narrativo vive.
Un livello di gioco ha un certo mood e una certa gestione dei suoni perché è stato progettato per creare una specifica game experience, meticolosamente studiata a priori dal game design.

Il game design, modellando l'esperienza finale del giocatore, stabilisce dunque come il gioco viene giocato.

1.3.7.2 Che cosa non è il game design

A causa della sua natura, si è spesso portati a scambiare il game design con molte altre cose che invece non lo sono.

Se si chiede di dare una spiegazione del termine "game design" al grande pubblico, analizzandone le reazioni, scopriremo che il game design è in realtà un concetto molto astratto. È proprio per questo motivo che si cerca di allinearlo con aree di studio e figure professionali già esistenti. Nel fare ciò, bisogna tuttavia prestare particolare attenzione a non confondere il game design con settori più "comuni" come:

- scrivere storie (è compito dello scrittore)
- dirigere il team (è compito del game director)

- disegnare (è ciò di cui si occupano gli artisti)
- modellare (è ciò che fanno gli artisti 3D)
- programmare il gioco (per fortuna ci sono i programmatori, cioè noi)
- generare idee (è compito di ciascun membro del team)

Questa lista è sicuramente incompleta e si potrebbe continuare ad aggiungere altri mestieri, incarichi o ruoli all'interno del team di sviluppo che creano confusione nella mente delle persone. Questi esempi sono tuttavia sufficienti per comprendere e iniziare a ricordare che, tutto ciò che è a loro collegato, non fa parte del game design ma può bensi essere utilizzato per miglirare l'insieme delle competenze di un game designer.

Che cosa sia il game design e di cosa esso si occupi, sono due aspetti estremamente differenti, inoltre, poiché il game design è sempre strettamente legato alla cultura, qualsiasi elemento di conoscenza potrebbe essere d'aiuto per migliorare l'arsenale delle competenze di un game designer.

L'idea di game designer che abbiamo illustrato fino a questo momento è dunque quella di uno "strano ruolo mutevole" e, nonostante una volta giunti a questo punto si dovrebbe aver ormai capito di cosa si occupa, tale mestiere può certamente non rientrare sempre all'interno dei confini che abbiamo delineato. Un game designer non è un artista, ma potrebbe aver bisogno delle abilità di un artista; non è uno scrittore, ma potrebbe aver bisogno di sapere come scrivere una storia; non è un programmatore, ma potrebbe trovarsi a dover ragionare come tale, ecc...

1.3.7.3 Che differenza c'è tra game design e game programming?

Il game design ruota attorno allo stabire la struttura della game experience. Come già detto in precedenza, il game design è simile al progetto di un edificio: riguarda più il dover definire la struttura degli elementi di gioco, i loro comportamenti e le loro interazioni, piuttosto che la loro creazione. Dunque, fare game design significa, tra le tante cose:

- definire cosa il giocatore può fare nel gioco
- definire come i nemici reagiranno alle azioni del giocatore
- definire la struttura dei livelli di gioco
- capire quali informazioni necessita il giocatore, quando si trova in un qualsiasi punto del gioco, per andare avanti
- definire come il giocatore può progredire all'interno del gioco
- escogitare le metodologie di erogazione della narrativa o, in altre parole, modi per narrare al giocatore la storia del videogioco (cutscenes, lettere, dialoghi, ecc...)

Nonostante questi siano solo degli scorci di quello che fa un game designer, tutte queste mansioni, nessuna esclusa, richiedono molto lavoro.

Il game design ricopre un ruolo fondamentale, ma non è abbastanza. Dopotutto, il gioco deve essere creato in qualche modo, e il team necessita di un modo per rendere concrete queste decisioni. Qui è dove il game programming entra in gioco.

Il game programming consiste nello scrivere codice che da vita alla game experience; è un altro ramo del game devlopment e riguarda lo strutturare e lo scrivere il codice necessario a realizzare il gioco finale.
Inventarsi le cose non rientra tra i compiti di un game programmer (egli tuttavia può e dovrebbe suggerire idee), ma implementarle all'interno di un game engine sì. Tutto il lavoro di programmazione si basa su quello che è stato definito e descritto dal game design. In sostanza, un game designer produce una serie di documenti che descrivono una feature, e che vengono poi utilizzati da un game programmer per implementarla nel gioco tramite codice.
Bisogna però fare attenzione a tenere sempre bene a mente che queste due figure (game designer e game programmer) non lavorano in due universi differenti, ma sono bensì strettamente legati. Questi due ruoli, infatti, sono in constante comunicazione tra di loro, in modo da poter la miglior game experience possibile.
La comunicazione è cruciale per qualsiasi ruolo e siede al centro di qualsiasi team ben performante. Questo è particolarmente vero quando si parla di game design e game programming. La comunicazione tra designers e programmers deve essere costante, così da ridurre il più possibile i fraintendimenti e cogliere al meglio le metodologie di lavoro dell'altra parte.

1.3.7.4 Il game design tende ad essere più sul lato creativo o su quello tecnico?

Il game design è una disciplina tecnica con scopi creativi, incentrata a conoscere il funzionamento di un gioco e come i giocatori interagiscono con esso.
Per far questo, i game designers fanno leva su diversi strumenti (documenti, flowcharts, spreadsheets, ecc...) per manipolare gli elementi di gioco, fare decisioni e strutturare le esperienze del target. Ma il game design è anche creare una game experience significativa che il giocatore sarà portato a vivere, obiettivo raggiunto solo attraverso sforzi creativi e modi per esprimere sé stessi all'interno del gioco mediante dei significati.
Alla luce di tutto ciò, si può quindi dire che il game design possiede un cuore artistico all'interno di un guscio tecnico.

Alcune persone tendono a mettere il game design sullo stesso piano della scienza, ma questo non è nient'altro che un errore raffazzonato. Il game design non è "una" scienza, ma utilizza alcuni concetti e principi basati "sulla" scienza; esso non esce fuori da un esperimento di laboratorio, e i suoi effetti non sono neanche misurabili con precisione.

Il game design è dunque una disciplina che possiede modelli, strumenti e metodologie che i game designers utilizzano per creare giochi migliori. Quindi, nonostante essere spesso basati su fatti scientifici, questi modelli non sono né magia né verità assoluta, ma bensì una semplice rappresentazione della struttura del videogioco.

1.3.7.5 In che modo il game design rientra nel game development?

Come già visto in precedenza, il game development consiste di quattro fasi principali:

1. Concept Phase
2. Pre-Production Phase
3. Production Phase

4. Post-Production Phase

Ogni gioco è un universo a sé, ma queste fasi sono sempre le stesse.

Il game design è enormemente coinvolto nella Concept Phase e nella Pre-Production Phase; dopotutto rappresenta la base di un videogioco, quindi ha senso che sia coinvolto a partire dall'inizio e che i game designers lavorino maggiormente in queste due fasi. Nonostante l'intero team sia coinvolto in questi stadi, infatti, i game designers svolgono sicuramente la maggior parte del lavoro, per aggiungere tutti quei dettagli del gioco che verrano implementati durante la Production phase.

Passando un po' più nel dettaglio:

- nella Concept Phase, il Game Design riguarda la game experience nel suo complesso. Il game design è una sorta di "lavoro di reverse engineering". Si sceglie una game experience a cui si vuole mirare, detta "target experience", e si crea il miglior gioco possibile in grado di suscitarla.
  Durante la Concept Phase, il Game Design è completamente incentrato sulla creazione di una descrizione generale della target experience. In sostanza, l'obiettivo di questa fase è sviluppare una *Game Direction* che funga da luce guida in tutte le altre fasi. In questa fase, i game designers definiscono dunque i temi (il contesto in cui il gioco si svolge) e i messaggi (inteso come il messaggio che i designers voglio trasmettere ai giocatori mediante il gioco), i pilastri di gioco, il core gameplay, il contesto narrativo, ecc....

- nella Pre-Production Phase, il Game Design ruota invece attorno al mapping di ogni singolo dettaglio. Adesso che la game direction è chiara, il team deve levare l'ancora e iniziare a navigare. La Pre-Production Phase è la fase più cruciale poiché è quella in cui tutti gli elementi di gioco vengono sistemati e definiti in ogni loro singolo dettaglio. Il game design, qui, è al 100% un ciclo iterativo incentrato sul capire il funzionamento del gioco e migliorare ogni feature con il passare del tempo. Se la Pre-Production Phase viene gestita in modo inadeguato o precipitoso, il gioco e il team potrebbero soffrire tremendamente durante la Production fase.

- nelle fasi rimanenti del game development, il game design riguarda per lo più il checking e il fixing. Durante la Production phase, il game design riduce significativamente la propria mole di lavoro, senza però sparire mai completamente del tutto. La Production phase è la fase di cui hanno bisogno principalmente programmatori e artisti 2D/3D per poter realizzare il gioco che verrà rilasciato. In questa fase, il game design consiste nell'effettuare hard testing del gioco e assicurarsi che tutto vada come previsto. Ma, dato che le cose non vanno mai come le si hanno pianificate, i designer devono reagire e risolvere prontamente i problemi che possono presentarsi, il tutto bilanciando deadline e target experience. Lo stesso discorso dicasi per la Post-Production Phase, anche se la mole di lavoro potrebbe aumentare se il team decidesse di creare una game expansion.

1.3.7.6 I 6 rami del game design

Il game design è più ampio di quanto si possa immaginare.

Il game design di per sé non è altro che un vasto campo di studi come la chimica, la medicina, la fisica, la letteratura, la filosofia e così via dicendo.... Anche se in un

primo momento questo paragone con i campi di studio potrebbe sembrare "presuntuoso" o, addirittura, "imbarazzante", credeteci quando diciamo che non lo è: il game design è un settore enorme.

Il game design viene normalmente diviso in 6 branche:

1. Gameplay Design
2. Level Design
3. System Design
4. Narrative Design
5. User Interface Design
6. User Experience Design

Diventare abili in tutti i settori è un'impresa praticamente impossibile e, inoltre, come se non bastasse, queste aree di studio spesso non si allineano perfettamente con le professioni aziendali. Quello che fa un gameplay designer o un system designer varia enormente a seconda dell'azienda che si sta guardando. Eppure, la natura del Gameplay Design rimane la stessa.
Il "Gameplay Designer" è semplicemente il nome che si da a un ruolo per meglio comprendere una proposta di lavoro. Le imprese non si allineano con campi di studio formali, né hanno necessità di farlo. Un team di game development potrebbe avere specifiche necessità da dover soddisfare a causa della portata o della struttura di project management del videogioco. Prendiamo ad esempio il termine "Combat Designer". Occuparsi del combat (ovvero il "combattimento") rientra tra i compiti del Gameplay Design, tuttavia, se si ha necessità di realizzare un videogioco con un sistema di combattimento ("combat system") complesso, si potrebbe pensare di incaricare qualcuno nello specifico e chiamare dunque il nuovo ruolo assegnatogli "Combat Designer".

1.3.7.7 Gameplay Design

Il *Gameplay Design* consiste nel definire tutti gli elementi e i comportamenti di gioco; è il regno delle meccaniche, dinamiche, regole e situazioni di gameplay. Se si pensa nuovamente a un gioco come a una casa, fare gameplay design vuol dire definire tutti i mattoni e capire come si relazionano l'uno con l'altro. Prendiamo ad esempio:

1. sparare a un nemico
2. muoversi all'interno di un'ambientazione
3. saltare in una buca nel pavimento
4. bloccare un attacco nemico con uno scudo
5. spingere e tirare oggetti

Non vi sono però mattoni uguali; il giocatore è il mattone più importante di tutti e il gameplay design deve occuparsi di tutte le possibili azioni e interazioni che esso potrebbe compiere.

lI gameplay design è uno dei rami più difficili (e poco capiti) del game design, ma è sicuramente il più importante, per un semplice motivo: il gameplay è alla base dell'intero design del gioco. Un videogioco senza gameplay non è un videogioco. Il gameplay è la base su cui lavorano tutti gli altri rami del game design. È quella zuppa primordiale di elementi, interazioni e situazioni interessanti sulle quali devono far leva le altre branche, per creare un'esperienza sbalorditiva.

Quando si hanno alcuni elementi di gameplay, si ha bisogno di un modo per metterli da qualche parte; qui entra in gioco il Level Design.

1.3.7.8 Level Design

Il *Level Design* è incentrato sul creare i livelli di gioco e gestire lo spazio interattivo; esso mira a definire la struttura (dimensioni e distanze) e il contenuto di un livello di gioco. Alcuni esempi possono essere la posizione dei nemici, il tipo e il numero di alcune zone, il layout di un'area o come queste sono connesse tra loro, ecc.... Il lavoro del level design è cruciale: è da qui che la target experience prende vita assumendo per la prima volta una forma concreta.

Il Level Design, tuttavia, da solo non può funzionare, poiché dipendente da quello che un giocatore può fare o non può fare. Il level design utilizza meccaniche, regole e dinamiche definite dal gameplay design per creare ambienti interattivi. Non si possono posizionare elementi in un livello se questi in primo luogo non esistono, ed è proprio questo il motivo per cui il level design non si preoccupa di crearne di nuovi. Il level design utilizza dunque tutti gli insiemi delle possibilità stabilite dal gameplay design per creare uno scenario interattivo, facendo leva su queste possibilità quanto più possibile. Se si considera ad esempio un personaggio nemico, mentre il gameplay design definisce le regole che ne governano il comportamento, il level design si occupa di tutti gli aspetti della zona in cui tale nemico verrà posizionato.

Il gameplay all'interno di un ambiente interattivo è un ottimo scheletro ma un videogioco, per potersi concentrare sulla target experience, necessita di essere rifinito: è il turno del System Design.

1.3.7.9 System Design

Il *System Design* definisce i comportamenti e le interazioni dei sistemi di gioco ("Game Systems").

Ma cos'è un sistema di gioco ("Game System")?. Un game system è un insieme di meccaniche, dinamiche e regole tutte appartenenti allo stesso contesto (combat system, movement system, weather system, ecc...).

Un errore comune è quello di pensare che il system design riguardi la creazione dei game system: quello è compito del gameplay design; il system design utilizza i modelli matematici e staticisti (insieme a tanto testing) per definire come un game system si comporta e interagisce con gli altri.

È fondamentale ricordare che il system design non inventa nulla di nuovo da aggiungere al gioco, ma bensì bilancia e gestisce tutti i game systems che il level design e il gameplay design hanno preparato. Si può dire dunque che una semplificazione ragionevole su quello che fa il system design è: "bilanciamento" ("balancing"). "Bilanciare qualcosa" significa gestire nel tempo il valore di un elemento di gioco, per far si che il videogioco possa produrre la target experience desiderata. Un esempio sono gli elementi di sistema in "Zelda Breath of The Wild". Quando Link utilizza le frecce elettriche su

uno specchio d'acqua, ben 2 sistemi interagiscono tra loro: il Character Combat System e l'Elemental System. Definire questi 2 sistemi e il loro comportamento è compito del gameplay design; occuparsi del loro posto all'interno del mondo spetta al level design e, infine, stabilire il loro comportamento in ogni situazione (quantità di danno di tipo elettrico, portata, durata, ecc...) è ciò di cui si occupa il system design.

Con gameplay design, level design, e system design, si ha già progettato un'ampia fetta di gioco, anche se parecchio lontana dall'essere completa. A questo punto, si potrebbe aggiungere qualcosa che non è strettamente necessario per definire il concetto di "videogioco", ma che rende comunque il tutto più avvincente: il contesto.

1.3.7.10 Narrative Design

Il *Narrative Design* si occupa di definire la struttura narrativa della game experience.

Uno dei primissimi errori che si commette quando si sente parlare di narrative design, è confinare quest'ultimo alla "scrittura creativa". Anche se questa associazione è parzialmente vera, nel senso che il narrative design comprende anche la storia e il worldbuilding di cui potrebbero occuparsi un semplice scrittore; fare narrative design significa, tuttavia, tra le altre cose, scegliere la struttura della storia, mantenere la narrativa e il gameplay consistente, definire come la storia viene espressa (ad esempio, cutscenes, dialoghi in-game, environmental storytelling), ecc...

Il contenuto narrativo non è semplicemente un layer separato che si sovrappone al gioco: il narrative design aggiunge una nuova dimensione alla game experience attraverso il contesto. In un videogioco il contesto è fondamentale, può cambiare terribilmente il significato del gameplay e del level design. Quello che fa un giocatore in un contesto specifico è diverso se tradotto in un altro contesto.
Un esempio che può rendere molto questo concetto è un arco narrativo del popolare manga "DanDaDan", in cui la protagonista si ritrova intrappolata all'interno di un gioco da tavolo e l'unico modo che ha per uscire è, come succede nella maggior parte dei giochi, sconfiggere tutti i boss nemici. Una volta essere riuscita nell'impresa le viene però rivelato che il gioco da tavolo è in realtà un baule maledetto in cui era imprigionato un potente yokai (entità sovrannaturale giapponese) che l'aveva ingannata a spezzare i proprio sigilli spacciandoli, appunto, per boss del gioco, ed essere così nuovamente libero. Come si può immediatamente intuire, le azioni della protagonista assumono un significato completamente diverso a seconda del contesto: sconfiggere dei nemici per liberare sé stessi è sicuramente molto più diverso e "innocente" dall'aver liberato un'entità maligna in grado di distruggere il mondo intero.
Il narrative design deve essere dunque sempre connesso e consistente con ogni altra branca del design per poter così creare una game experience significativa.

Trasmettere un significato è quasi impossibile senza un contesto, questo perché il giocatore non può "sentire" il risultato delle proprie azioni; e questo diminuisce enormemente il potere della game experience.

Ora che si ha un gioco calato all'interno di un contesto, si necessitano però di ulteriori informazioni che permettano al giocatore di interagire con le regole. La User Interface Design esiste proprio per risolvere questo problema.

1.3.7.11 User Interface (UI) Design

Lo *User Interface (UI) Design* riguarda l'aspetto e il comportamento delle interfacce di gioco.

Una interfaccia di gioco ("Game Interface") è esattamente quello che si è portati a pensare: timer, contatori di munizioni, barre della vita (health bars), schermata dell'inventario, menu principale, ecc...

Fare UI design significa definire questi elementi di interfaccia, i loro limiti estetici (per gli artisti UI) e il loro comportamento nel gioco. La maggior parte delle interfacce non fanno strettamente parte del gioco perché non interferiscono in maniera diretta con altri elementi ma rappresentano semplicemente qualcosa per il giocatore. Ciononostante, le UI sono un aspetto essenziale della player experience.

L'UI design studia quali sono le informazioni necessarie al giocatore in un determinato istante. Il giocatore ha bisogno di informazioni mentre sta giocando, come ad esempio le munizioni rimanenti, il livello e le statistiche del proprio personaggio. Senza delle interfacce appropriate, il giocatore sarebbe costretto a ricavarsi tutte le informazioni di cui necessita direttamente dal gioco stesso e, dunque, molte rimarrebbero inevitabilmente nascoste.

Un buon modo per apprezzare l'importanza dell'UI Design può essere quello di confrontare uno screenshot di una scena di un gioco con e senza le UI attive: potrebbe risultare molto difficile capire anche solo quanta vita si possiede. Genshin Impact è un ottimo esempio.

Con la UI design dalla nostra parte, ciò che resta è usare lo Experience Design per modellare la sensazione di interazione del giocatore.

1.3.7.12 User Experience (UX) Design

Lo *User Experience (UX) Design* definisce l'usabilità, il feedback, la facilità e il piacere dell'interazione del giocatore con il videogioco (game interaction). Chi ha mai deviato in Sekiro, saltato in Super Mario 64, guidato in Gran Turismo o sparato in Doom può sicuramente comprendere la bella sensazione al quale si sta alludendo. Sono tutti esempi di "Game Feel", termine che sta a indicare la piacevole sensazione che si prova nel compiere azioni di gioco, il quale è parte fondamentale dell'UX design.

L'UX design si preoccupa anche di studiare come una tipologia specifica di utente (non giocatore) interagisce con il software. All'interno di questa area di studio sono dunque comprese l'usabilità inerente le interfacce, il feedback (visivo, uditivo e tattile), l'accessibilità per gli utenti con disabilità, ecc.... Per di più, utilizzando il feedback, l'UX design può guidare le informazioni insieme all'UI design, in modo tale da far in modo che il giocatore si accorga di qualcosa quando necessario. Ad esempio, la riproduzione di un VFX (Visual Effect) o di un SFX (Sound Effect), quando un personaggio colpisce o viene colpito da qualcosa. Anche se queste aree possono sembrare diverse l'una dall'altra, esse condividono tutte lo stesso obiettivo.

L'UX design si concentra dunque sulla sensazione di gioco, per fare in modo di aumentare la consapevolezza e ridurre la frustazione del giocatore. Questo campo di studi si basa fortemente sulla psicologia e sulle neuroscienze. Fare UX design significa studiare come gli essere umani si comportano e pensano e fare leva su queste conoscenze per migliorare la game experience. Lo UX design mira dunque a ridurre la frustazione del giocatore quanto più possibile così che il giocatore posso rimanere concentrato nel gioco. Un gioco senza UX design funzionerebbe comunque, ma avrebbe un layer di difficoltà così inutile e pericoloso che rovinerebbe al 100% l'esperienza del giocatore.

1.3.7.13 Sound Design

Nonostante il nome potrebbe farlo pensare, il *Sound Design* non è un ramo del Game Design. Anche se il termine "design" è utilizzato ogni volta che c'è un qualcosa da progettare, il Game Design riguarda la progettazione specifica della game experience, e nient'altro. Il sound design non farà dunque parte del game design, ma è certamente una branca del design nel suo complesso.

Il sound design non si occupa solamente di decisioni riguardanti i suoni di gioco, ma bensì di trovare o creare quei suoni che sono stati stabiliti, nella maggior parte dei casi, dallo UX design. Infatti, il team di sound design è generalmente composto da tecnici del suono (che registrano gli effetti sonori), ingegneri del suono (che gestiscono i suoni tramite codice) e compositori (che registrano le soundtracks). Quello che viene chiamato "sound designer", nella realtà è spesso un tecnico del suono che ha bisogno di trovare o registrare da zero il giusto effetto sonoro.

Il sound design riguarda dunque il creare e gestire l'esperienza sonora (sound experience) del gioco. Chiunque abbia mai giocato a un videogioco con l'audio mutato, sa benissimo che l'esperienza di gioco è tremendamente differente, e questo perché il suono riveste un ruolo importante: evoca emozioni, fornisce feedback al giocatore, accresce il feeling delle sue azioni nel gioco, ecc....
Avere o no tutto ciò in un videogioco è una decisione di User Experience Design, o comunque riguarda la Game Direction. Il sound design si occupa solamente di scegliere il suono giusto, proprio come fa un artista nel disegnare la giusta opera d'arte o un programmatore nello scrivere il codice corretto di una feature.

1.4 Categorie del game programming

La programmazione di videogiochi (game programming) comprende molte categorie, specialmente per chi cerca lavoro nell'industra dei videogiochi. In questo paragrafo, tuttavia, introdurremo solamente le tre categoria nelle quali ci si imbatte con più facilità:

1. Game Scripting
2. Third Party Modules
3. Game Engine Programming

In questo documento, ci occuperemo principalmente di game scripting.

1.4.1 Game Scripting

Lo *scripting*, conosciuto anche come *coding* o *programming*, è il modo di scrivere istruzioni per l'esecuzione di un videogioco, utilizzando un linguaggio veramente specifico che può essere compreso dai computer responsabili di eseguire il gioco. Ciò è molto simile alle regole di un gioco da tavolo, dove vegono spiegare le diverse "procedure": l'ordine e gli step dei turni, come capire chi inizia, le differenti opzioni che si hanno in momenti particolari, ecc.... L'unica vera grande "differenza" è che le istruzioni di un gioco da tavolo sono scritte per essere capite dagli umani, e non dai computer.

Con i Core Frameworks (strumenti visivi di sviluppo che non necessitano di conoscenze nella programmazione), oggetti di gameplay e strumenti di design ambientale si possono creare game experience complete in Core, senza appunto scrivere

neanche una riga di codice. L'utilizzo di script consente, invece, di creare opzioni di gameplay più specifiche e personalizzate, programmi di movimento delle diverse entità e sequenze o eventi randomici che offrono ai giocatori più opzioni per interagire con il gioco.

1.4.2 Third Party Modules

Se ci si ferma a riflettere qualche secondo, un gioco non è nient'altro che tanti "piccoli progetti" messi insieme a formare quello che viene chiamato "videogioco". Sono dunque proprio questi "piccoli progetti" che includono un dialog system, un pause menu, la fisica dei proiettili, delle chiamate a internet, un animation system, ecc..., quando lavorano insieme, a dare vita a un videogioco.

Molte volte, è meglio investire il proprio tempo utilizzando software di terze parti (Third Party Modules) piuttosto che creare il proprio "piccolo progetto" da zero. Ad esempio, non è raro che un game programmer compri un dialog system come software di terze parti in modo che non abbia bisogno di incentrasi su di esso, ma possa bensì occuparsi di realizzare il gioco vero e proprio.

Comprare un software di terze parti velocizza lo sviluppo a discapito della flessibilità, poiché si è limitati alle feature che tale software fornisce e non ve se ne possono aggiungere di nuove in maniera arbitraria.

Se si vuole vedere il tutto da un'altra prospettiva: lavorare come sviluppatore di software di terze parti è un buon modo per fare soldi.

1.4.3 Game engine programming

Il game engine programming è una delle cose più difficili da programmare quando si parla di game development. Come dato di fatto, ai principianti è infatti fortemente sconsigliato di iniziare il loro percorso in questo ambito, questo perché, per molti, potrebbe finire per essere la motivazione che li spingerà ad abbandonare per sempre il game programming.

Il game engine programming consiste nel lavorare alla creazione o all'aggiunta di feature di un game engine che i game developer finiranno per utilizzare quando svilupperano un videogioco.