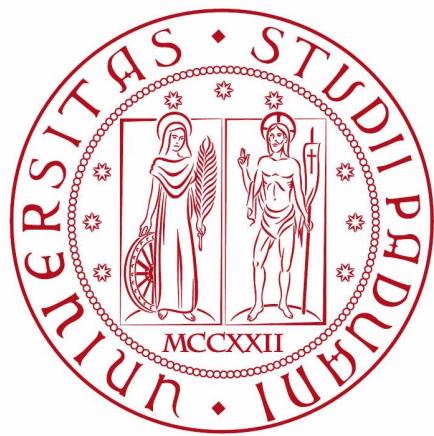


Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA

CORSO DI LAUREA IN INFORMATICA



**Design e implementazione di tool e
funzionalità di debug grafico per lo sviluppo
di videogame 3D multipiattaforma**

Tesi di laurea

Relatore

Prof. Claudio Enrico Palazzi

Laureando

Maurizio Biancucci

“Live as if you were to die tomorrow. Learn as if you were to live forever.”

— Mahatma Gandhi

Dedicato alla mia famiglia

Sommario

Lo scopo del presente documento è descrivere il lavoro svolto dal laureando Maurizio Biancucci durante lo stage, della durata di 320 ore, svolto presso Milestone S.r.l. Gli obiettivi erano molteplici: dapprima lo sviluppo di strumenti atti a velocizzare il debug e la stesura di file XML e successivamente, il disegno di elementi grafici di debug all'interno di giochi multipiattaforma.

“Don’t let the fear of losing be greater than the excitement of winning.”

— Robert Kiyosaki

Ringraziamenti

Desidero innanzitutto ringraziare il professor Claudio Enrico Palazzi per avermi seguito durante la scrittura della tesi e durante l’attivazione dello stage.

Vorrei inoltre esprimere la mia gratitudine agli amici che mi hanno sorretto e sostenuto durante gli anni di studio e, soprattutto, in questo difficilissimo ultimo periodo. Grazie per il tempo insieme.

Per ultima, ma non per importanza, voglio ringraziare la mia famiglia che mi ha sempre sostenuto in tutte le mie scelte.

Padova, Settembre 2014

Maurizio Biancucci

Indice

1	Introduzione	1
1.1	Descrizione dell'azienda	1
1.2	Motivazioni all'attivazione dello stage	1
1.3	Obiettivi dello stage	1
1.4	Scopo del documento	2
1.5	Organizzazione del testo	2
1.6	Nota sul codice prodotto	2
2	Milestone	3
2.1	Milestone S.r.l. nel mercato	3
2.2	Pirateria	4
2.3	Suddivisione della forza lavoro	5
2.4	Game workflow	7
2.5	Ruolo dei 3D programmer nel workflow	8
2.6	Configuration e data Management	10
2.7	Asset	11
2.8	Code build	12
2.9	Metadati	13
2.10	Norme di codifica	14
2.11	Assert e Log	15
3	Multiplatform File Analyzer	17
3.1	Introduzione	17
3.1.1	Premessa	17
3.1.2	Lo scopo	17
3.2	Casi d'uso	17
3.2.1	UC 1: Caso d'uso principale	18
3.2.2	UC 1.1: Analisi dei file di un gioco multipiattaforma	18
3.2.3	UC 1.1.1: Inserimento dei dati in input all'analisi	19
3.2.4	UC 1.1.1.1: Inserimento percorso radice dei file da analizzare .	20
3.2.5	UC 1.1.1.2 Inserimento dei nomi delle cartelle delle piattaforme	20
3.2.6	UC 1.1.1.3 Inserimento del warning time	20
3.2.7	UC 1.1.2 Avvio dell'analisi	21
3.2.8	UC 1.1.3 Visualizzazione errori sui dati in input	21
3.2.9	UC 1.1.4 Visualizzazione dei risultati dell'analisi	21
3.2.10	UC 1.1.4.1 Visualizzazione di tutti i file presenti in almeno una cartella di piattaforma	22
3.2.11	UC 1.1.4.2 Visualizzazione del percorso per il file visualizzato .	23
3.2.12	UC 1.1.4.3 Per ogni file visualizzato, visualizzazione del tempo di ultima modifica per ogni piattaforma per cui è presente . .	23
3.2.13	UC 1.1.4.4 Per ogni file visualizzato, visualizzazione della massima differenza fra i tempi di ultima modifica delle piattaforme in cui è presente	23



INDICE

3.2.14 UC 1.1.4.5 Visualizzazione dei file che hanno la massima differenza fra i tempi di ultima modifica delle piattaforme in cui è presente, maggiore del tempo di warning	24
3.2.15 UC 1.1.4.6 L'utente può aprire l'explorer del sistema operativo per visualizzare dove questo si trovi	24
3.3 Tracciamento dei requisiti	24
3.3.1 Requisiti funzionali	25
3.3.2 Requisiti di qualità	27
3.3.3 Requisiti di vincolo	27
3.4 Tecnologie e strumenti	27
3.4.1 Qt® Framework	27
3.4.2 Qt® Creator	28
3.5 Ciclo di vita del software	28
3.6 Progettazione	28
3.6.1 Diagramma delle classi	28
3.6.2 Pacchetto controller	30
3.6.3 Pacchetto observer	30
3.6.4 Pacchetto data	30
3.6.5 Pacchetto view	31
3.6.6 Pacchetto core	31
3.7 Codifica	31
3.8 Uso pratico	32
3.9 Conclusioni	33
4 XML Editor	35
4.1 Introduzione	35
4.1.1 Premessa	35
4.1.2 Lo scopo	35
4.2 Casi d'uso	35
4.2.1 UC 1: Caso d'uso principale	36
4.2.2 UC 1.1: Creazione di un nuovo file XML	37
4.2.3 UC 1.1.1: Inserimento del percorso dove memorizzare il nuovo XML	38
4.2.4 UC 1.1.2: Inserimento del nome del file del nuovo XML	38
4.2.5 UC 1.2: Apertura di un file XML preesistente	38
4.2.6 UC 1.3 Editing del file XML correntemente aperto	39
4.2.7 UC 1.3.1 Visualizzazione del nodo riferito dal nodo selezionato	41
4.2.8 UC 1.3.2 Visualizzazione degli errori occorsi durante la ricerca del nodo riferito	41
4.2.9 UC 1.3.3 Aggiunta di un nuovo nodo figlio al nodo selezionato	41
4.2.10 UC 1.3.4 Editing del nodo selezionato	41
4.2.11 UC 1.3.4.1 Modifica del valore dell'elemento	42
4.2.12 UC 1.3.4.2 Inserimento di un nuovo attributo	42
4.2.13 UC 1.3.4.3 Modifica di un attributo esistente	43
4.2.14 UC 1.3.4.4 Rimozione di un attributo esistente	43
4.2.15 UC 1.3.5 Duplicazione del nodo selezionato	43
4.2.16 UC 1.3.6 Copia del nodo selezionato	43
4.2.17 UC 1.3.7 Incollare il nodo precedentemente copiato	44
4.2.18 UC 1.3.8 Rimozione del nodo selezionato	44
4.2.19 UC 1.3.9 Rimozione di tutti i nodi figli del nodo selezionato	44
4.2.20 UC 1.3.10 Istanziazione di una nuova relazione	44
4.2.21 UC 1.3.11 Visualizzazione degli errori occorsi durante l'istanziazione di una nuova relazione	45
4.2.22 UC 1.3.12 Annullamento dell'ultima modifica eseguita	45
4.2.23 UC 1.3.13 Ripristino dell'ultima modifica annullata	45



4.2.24 UC 1.4 Salvare le modifiche di un file modificato	46
4.2.25 UC 1.5 Editing del filtro sul nome dell'attributo da visualizzare	46
4.2.26 UC 1.6 Editing dei file associati	46
4.2.27 UC 1.6.1 Aggiunta di un nuovo file associato	47
4.2.28 UC 1.6.2 Rimozione di un file precedentemente associato	47
4.2.29 UC 1.7 Editing delle relazioni	47
4.2.30 UC 1.7.1 Inserimento di una nuova relazione	48
4.2.31 UC 1.7.2 Modifica di una relazione precedentemente inserita	48
4.2.32 UC 1.7.3 Rimozione di una relazione precedentemente inserita	48
4.2.33 UC 1.8 Importazione delle relazioni da file	49
4.2.34 UC 1.9 Esportazione delle relazioni su file	49
4.2.35 UC 1.10 Avvio del check sulle relazioni	49
4.2.36 UC 1.11 Visualizzazione del risultato del check sulle relazioni	49
4.2.37 UC 1.11.1 Visualizzazione degli errori	50
4.2.38 UC 1.11.2 Selezione dei filtri sulle categorie di errori da visualizzare	50
4.2.39 UC 1.11.3 Visualizzazione dell'elemento causa dell'errore nell'alfabeto XML	51
4.2.40 UC 1.12 Visualizzazione dell'about del programma	51
4.2.41 UC 1.13 Visualizzazione degli errori occorsi durante l'apertura del file	51
4.2.42 UC 1.14 Visualizzazione degli errori occorsi durante il salvataggio del file	52
4.2.43 UC 1.15 Visualizzazione degli errori occorsi durante l'importazione	52
4.2.44 UC 1.16 Visualizzazione degli errori occorsi durante l'esportazione	52
4.3 Tracciamento dei requisiti	53
4.3.1 Requisiti funzionali	53
4.3.2 Requisiti di qualità	59
4.3.3 Requisiti di vincolo	59
4.4 Tecnologie e strumenti	59
4.4.1 Qt [®] Framework	59
4.4.2 Qt [®] Creator	59
4.5 Ciclo di vita del software	60
4.6 Progettazione	60
4.6.1 Diagrammi delle classi	60
4.6.2 Pacchetto observer	62
4.6.3 Pacchetto data	63
4.6.4 Core	64
4.6.5 Pacchetto command	64
4.6.6 Pacchetto view	65
4.7 Codifica	67
4.8 Uso pratico	68
4.9 Conclusioni	70
5 Funzionalità di disegno di elementi grafici 3D a scopo di debug	73
5.1 Stampa dei livelli di LOD di moto e piloti	73
5.2 Rifattorizzazione della gerarchia di stampa	77
5.3 Stampa del nome degli spazi di riferimento	79
5.4 Vista gerarchica dei world nel Beholder	82
5.5 Disegno degli scheletri di animazione	93
6 Conclusioni	105
6.1 Raggiungimento degli obiettivi	105
6.2 Conoscenze acquisite e valutazione della preparazione accademica . . .	106
6.3 Valutazione personale	107



INDICE

Glossario	109
Bibliografia	113

Elenco delle figure

2.1 Screenshot di EMotion Studio durante la visualizzazione dello skeleton di animazione di un meccanico in T-pose	9
2.2 Screenshot di EMotion Studio durante la visualizzazione della gerarchia dei bone di animazione dello skeleton di un meccanico	10
2.3 Screenshot di Perforce	11
3.1 Multiplatform File Analyzer - UC 1: Caso d'uso principale	18
3.2 Multiplatform File Analyzer - UC 1.1: Analisi dei file di un gioco multipiattaforma	19
3.3 Multiplatform File Analyzer - UC 1.1.1: Inserimento dei dati in input all'analisi	20
3.4 Multiplatform File Analyzer - UC 1.1.4: Visualizzazione dei risultati dell'analisi	22
3.5 Diagramma delle classi di Multiplatform File Analyzer	29
3.6 Screenshot di Multiplatform File Analyzer nel quale si può osservare la segnalazione di errori sui dati in input all'analisi	32
3.7 Screenshot di Multiplatform File Analyzer nel quale si può osservare la presentazione dei risultati dell'analisi	33
4.1 XML Editor - UC 1: Caso d'uso principale	37
4.2 XML Editor - UC 1.1: Creazione di un nuovo file XML	38
4.3 XML Editor - UC 1.3 Editing del file XML correntemente aperto	40
4.4 XML Editor - UC 1.3.4 Editing del nodo selezionato	42
4.5 XML Editor - UC 1.6 Editing dei file associati	46
4.6 XML Editor - UC 1.7 Editing delle relazioni	47
4.7 XML Editor - UC 1.11 Visualizzazione del risultato del check sulle relazioni	50
4.8 Diagramma dell'integrazione delle classi model-view di Qt® per la visualizzazione dell'albero XML	61
4.9 Diagramma delle classi usate per l'editing dei file XML	62
4.10 Screenshot di XML Editor nel quale si può osservare la dialog di edit di un nodo dell'albero XML	68
4.11 Screenshot di XML Editor nel quale si può osservare la visualizzazione dei file XML come alberi	69
4.12 Screenshot di XML Editor nel quale si può osservare la dialog che chiede all'utente come procedere dopo il drag & drop di un file da Windows .	70
4.13 Screenshot di XML Editor nel quale si può osservare la dialog che permette l'editing dei file associati	70
4.14 Screenshot di XML Editor nel quale si può osservare la dialog che permette l'editing delle relazioni	71
4.15 Screenshot di XML Editor nel quale si può osservare la dialog che permette la visualizzazione dell'esito del risultato del check delle relazioni	71



ELENCO DELLE FIGURE

5.1	Screenshot di MotoGP 14	74
5.2	Screenshot di MotoGP 14 nel quale si può osservare la vecchia stampa dei livelli di LOD	74
5.3	Screenshot di MotoGP 14 nel quale si può osservare la nuova stampa dei livelli di LOD per le moto	76
5.4	Screenshot di MotoGP 14 nel quale si può osservare la nuova stampa dei livelli di LOD per i piloti	77
5.5	Screenshot del Beholder nel quale è possibile vedere dove attivare e disattivare la stampa dei LOD delle moto	77
5.6	Diagramma UML della gerarchia di stampa del testo. Sono riportati solo i metodi più significativi	78
5.7	Screenshot del Beholder nel quale è possibile lanciare il disegno degli assi tramite l'azione “ToggleDraeAxes” dello “SceneDirector” visibile in alto a sinistra	80
5.8	Screenshot di MotoGP 14 in cui nel main menu è possibile vedere gli assi dei punti in cui viene ancorata la camera e la stampa del nome corrispondente	83
5.9	Screenshot del Beholder in cui è possibile visualizzare la versione a lista piatta sopra e la versione ad albero sotto per la bike 306	84
5.10	Sintesi della terminologia usata nelle animazioni scheletrali	93
5.11	Screenshot del Beholder in cui è possibile visualizzare dove è permesso attivare e disattivare il disegno dello skeleton del rider	94
5.12	Screenshot n. 1 del disegno dello skeleton del rider	98
5.13	Screenshot n. 2 del disegno dello skeleton del rider	99
5.14	Screenshot del Beholder in cui è visibile lo SkeletonDebugDrawerManager	100
5.15	Screenshot di MotoGP 14 in cui è stato attivato il disegno degli scheletri di animazione di 2 meccanici	100
6.1	Iistogramma del riepilogo ore a preventivo e consuntivo	106
2	Rappresentazione grafica di vertical e horizontal slice	111

Elenco delle tabelle

3.1 Requisiti funzionali di Multiplatform File Analyzer	27
3.2 Requisiti di qualità di Multiplatform File Analyzer	27
3.3 Requisiti di vincolo di Multiplatform File Analyzer	27
4.1 Requisiti funzionali di XML Editor	59
4.2 Requisiti di qualità di XML Editor	59
4.3 Requisiti di vincolo di XML Editor	59
6.1 Riepilogo ore a preventivo e consuntivo	105

Capitolo 1

Introduzione

1.1 Descrizione dell'azienda

Milestone S.r.l. (MI) nasce a Milano nel 1996, e ancora oggi rappresenta la più grande realtà italiana impegnata nello sviluppo di videogiochi per console e PC. L'azienda è riconosciuta a livello mondiale come uno dei migliori team di sviluppo nel settore racing, sia asfalto che off-road, sia motociclismo che automobilismo[Site].

Fra i suoi titoli più importanti si vuol ricordare “SCAR - Squadra Corse Alfa Romeo”, “MXGP”, “WRC 4 Word Rally Championship”, “la serie Superbike”, “la serie MotoGP”. Tutti sviluppati sotto licenze ufficiali.

Al momento la casa è impegnata nello sviluppo sulle nuove console, affrontando la sfida delle nuove tecnologie, cercando al contempo di mantenere lo stesso ritmo produttivo.

1.2 Motivazioni all'attivazione dello stage

Lo stage nasce dall'esigenza dell'azienda, in fase di forte espansione, di aumentare il proprio personale, investendo quindi in formazione di figure *junior_g* da inserire successivamente in pianta stabile nell'organico dell'azienda.

In particolare, vista la sempre crescente realistica grafica dei videogame, l'azienda ha l'esigenza di ampliare il personale del *team_g* di programmazione 3D.

1.3 Obiettivi dello stage

L'obiettivo dello stage è quello di inserire lo studente all'interno del team di programmazione 3D e fargli assaggiare le sfide con cui il team si confronta quotidianamente.

Lo stage prevede quindi una prima parte in cui si introduce lo studente ai problemi tipici chiedendogli di: analizzare, progettare e implementare dei tool di supporto allo scopo di velocizzare il lavoro dell'intero team.

La seconda parte dello stage prevede lo studio ad alto livello del funzionamento di un *engine_g* di gioco, in particolare i flussi presenti nel settore di pertinenza del team. Seguito dalla progettazione ed implementazione di alcune funzionalità grafiche di *debug_g*. È prevista la possibilità, in fase di progettazione, di attuare del *refactoring_g* sul codice di debug grafico già presente, allo scopo di ottenere una maggiore manutenibilità ed estendibilità futura.



1.4. SCOPO DEL DOCUMENTO

1.4 Scopo del documento

Il presente documento ha lo scopo di presentare gli esiti delle attività sostenute dallo studente durante il periodo di stage sostenuto presso Milestone S.r.l.

- Nel capitolo 2 verranno presentati i flussi di lavoro aziendali che portano un videogame da essere una semplice idea a diventare un prodotto maturo pronto alla vendita. Particolare attenzione verrà dedicata alle responsabilità del team di programmazione 3D.
- Nel capitolo 3 si presenterà il tool Multiplatform File Analyzer, analizzandone approfonditamente tutti gli aspetti.
- Nel capitolo 4 verrà presentato il tool XML Editor, del quale sarà fornita una profonda analisi a partire dai requisiti alla progettazione.
- Nel capitolo 5 verranno presentate le funzionalità di disegno di elementi grafici a scopo di debug progettate e implementate in game.
- Nel capitolo 6 sarà presente una valutazione finale sull’esperienza di stage.

1.5 Organizzazione del testo

Per tutte le parole e le sequenze di parole in *italico* seguite dalla sequenza di caratteri “|g|” a pedice, è presente un piccolo approfondimento nel Glossario in Appendice.

1.6 Nota sul codice prodotto

Il codice dei tool sviluppati, in accordo con l’azienda, è stato pubblicato sulla piattaforma [GitHub®¹](#) sotto licenza GNU GPL v3.

Riguardo il codice sviluppato all’interno dell’engine di gioco, è stato inserito il minimo necessario per permettere al lettore di comprendere e al contempo non rivelare informazioni sensibili dell’azienda. Tutto il codice di gioco presente all’interno di questo documento è materiale protetto da copyright appartenente a Milestone S.r.l. ed è pubblicato sotto autorizzazione. Ogni riproduzione è severamente vietata, ogni richiesta di ulteriore documentazione va recapitata direttamente a Milestone S.r.l.

¹Si invita il lettore a guardare nei capitoli 4 e 5 sotto la sezione Codifica, gli url dove reperire il codice sorgente.

Capitolo 2

Milestone

2.1 Milestone S.r.l. nel mercato

Milestone si pone come software house di spicco nel settore dei videogiochi a livello mondiale. Grazie all'esperienza acquisita nel proprio ambito (racing) riesce a competere con case produttrici molto più grandi e blasonate senza per niente sfigurare.

Attualmente l'azienda sviluppa i propri giochi per le seguenti piattaforme: PC, Sony Playstation Vita®, Sony Playstation 3®, Sony Playstation 4®, Microsoft Xbox 360® e Microsoft Xbox One®. Quest'ultima piattaforma al momento non ha ancora avuto un titolo Milestone nello scaffale ma, visto l'attuale impegno, un primo titolo per Xbox One® non tarderà ad arrivare.

Come già detto, i prodotti Milestone sono venduti in tutto il mondo. Il solo mercato italiano procura il 12% del fatturato aziendale. Il resto del ricavato arriva principalmente dall'Europa in cui Milestone ha una lunga tradizione. Di recente acquisizione è il mercato Americano, dove verrà a breve distribuito un nuovo gioco grazie ad un accordo di sub-licensing con Namco Bandai. Anche nel resto del mondo: Giappone, Russia, Australia e Sud Africa i giochi sono distribuiti tramite accordi di sub-licensing¹.

La divisione dei ricavati sulle varie piattaforme indica la rapida crescita delle console di ultima generazione, confermando anche per i giochi Milestone la stessa tendenza (Xbox One® al momento esclusa). Ricopre invece un ruolo marginale il mercato PC, la cui quota è davvero minimale.

Milestone vende sia in formato digitale tramite gli store di Sony, Microsoft e Valve (Steam), sia in formato retail con CD e copertina rigida. La produzione delle copie retail e la distribuzione fisica nei negozi è appaltata esternamente.

Milestone è una delle poche aziende al mondo che ricopre anche il ruolo di Publisher. Questo ruolo consiste nel dialogo con Sony e Microsoft per accordare la pubblicazione di un videogame sulle loro console. Solitamente, la maggior parte delle software house, si affida ad aziende terze. Ricoprire questo ruolo internamente consente all'azienda di avere un dialogo diretto con i console owner, permettendo una più rapida interazione. Per capire il motivo per il quale un rapido dialogo sia così importante bisogna introdurre le TRC/TCR². Esse sono delle condizioni tecniche/qualitative che i giochi devono rispettare per essere pubblicati sulle console. Ogni console ha le sue specifiche. I console owner hanno interi team per testare i game secondo le TRC/TCR e quindi

¹Sita.

²Sitc.



2.2. PIRATERIA

l'utilità di questo reparto si spiega da sola³. Lo scopo delle TRC/TRC è quello di assicurare un alto standard qualitativo del software che viene distribuito sulle console. Inoltre è uno strumento utilizzato dai console owner per poter meglio controllare l'universo attorno le loro console.

La curva dei profitti in relazione all'uscita del gioco sul mercato è molto semplice. Nel primo mese a partire dal *Day One*_{|g|} si concentra la maggior parte del profitto dell'intera vendita del videogioco. Al passare del tempo inizia un repentino calo delle vendite e ribasso del prezzo. Nel frattempo inoltre escono titoli concorrenti più nuovi che contribuiscono alla tendenza. Dopo la rapida decrescita si succede una fase molto più lenta che prosegue fino alla morte del prodotto.

2.2 Pirateria

Nel contesto della curva di profitto di un gioco si inserisce la lotta contro la pirateria. L'azienda si affida, per la protezione anti-copia, a software diversi in base alla piattaforma target. Per le console tutte le aziende del settore, tra cui Milestone, si affidano ai meccanismi anti-copia offerti dai console owner per le proprie piattaforme. Su PC l'azienda si affida a Solid Shield⁴ per le versioni retail e alla protezione build-in offerta da Steam.

Il problema della pirateria informatica su PC è sempre stato molto sentito dalle software house, in quanto in Microsoft Windows⁵, essendo un sistema operativo aperto, risulta molto più semplice ovviare alle protezioni digitali anti-pirateria rispetto alle console, sulle quali gira un sistema chiuso in cui non è permesso installare software non pre-approvati dal console owner.

Solid Shield funziona in maniera trasparente al codice del gioco, in quanto, una volta prodotto l'eseguibile finale di esso, questo viene passato a loro, i quali costruiscono un guscio esterno di protezione sbloccabile soltanto con un codice valido, solitamente distribuito insieme alla copia del gioco.

Su Steam invece la fase di sviluppo passa attraverso un utente aziendale riconosciuto come sviluppatore di quel particolare gioco, il quale lo può avviare anche senza possederlo nella propria libreria, a patto di avere un file nella stessa cartella dell'eseguibile con all'interno il codice identificativo segreto del gioco. L'hacking della protezione Steam passa proprio attraverso questo meccanismo, scoprendo il codice segreto e facendo credere al sistema di essere un utente sviluppatore.

Al momento PS3® e Xbox 360® sono le uniche console, per cui Milestone sviluppa, che sono state *hackerate*_{|g|} e compromesse. PS3®, ad esempio, è stata violata quando era presente il firmware versione 3.55. La casa è riuscita però a proteggere tutte le console che sono nate con o sono state aggiornate a firmware successivi. Rimangono però attualmente scoperte le console non aggiornate per via ufficiale. Sony ha quindi attuato politiche di *ban*_{|g|} definitivo dal Playstation Network® (*PSN*[®]_{|g|}) delle console hackerate. La comunità pirata ha però nel tempo rilasciato versioni modificate che permettono a coloro che hanno come base una console con firmware 3.55 o inferiore di installare nuove versioni aggiornate dello stesso con protezioni anti-ban per permettere comunque l'accesso al PSN®. Ne è susseguita una danza ciclica in cui Sony rilasciava un nuovo firmware che correggeva il buco di protezione seguito dopo poco da un

³Sitb.

⁴<http://www.solidshield.com/>

⁵Microsoft Windows è il sistema maggiormente diffuso sul quale vengono giocati i videogiochi su PC e per questo viene preso ad esempio



2.3. SUDDIVISIONE DELLA FORZA LAVORO

rilascio di un nuovo firmware che trovava un altro modo per permettere comunque di averne uno aggiornato e la protezione per l'accesso al PSN®. Inoltre i nuovi giochi richiedono espressamente versioni del firmware molto recenti per proteggersi da copie hackerate del firmware non aggiornate.

Da tutto questo ne consegue quindi uno sforzo contro la pirateria da parte di Milestone solo nelle primissime fasi dopo il day one, successivamente tra prezzi scontati, calo delle vendite e azione della pirateria limitata quasi soltanto al PC, dal quale provengono una piccolissima parte dei ricavati, non giustificano lo sforzo economico per continuare la battaglia.

2.3 Suddivisione della forza lavoro

Milestone S.r.l. conta ad oggi circa 120 impiegati, di questi circa 90 appartengono ai reparti che si occupano dello sviluppo, ovvero creazione degli asset (modelli 3D, piste, animazione ecc), programmazione e game design. Il resto si occupa del publishing (circa 10 persone), della pianificazione a lungo raggio e management aziendale, del quality assurance (QA), del configuration e data management, delle human resources (HR), della amministrazione e del settore IT.

Il team di Human Resources si occupa di gestione del personale. Suo il compito di selezionare e arruolare nuove leve, promuovere o licenziare personale. Essendo personale non tecnicamente specializzato in game development si affida ai team leader per eseguire le valutazioni tecniche dei candidati.

Il team di configuration e data management si occupa di gestire l'utilizzo del repository aziendale, tra cui la gestione dei branch. Inoltre si occupano delle build aziendali, la gestione e trasformazione degli asset dai formati nativi dei programmi di sviluppo ai formati compressi e ottimizzati con i quali verranno distribuiti insieme al gioco.

L'amministrazione esegue compiti non appartenenti al core business dell'azienda ma comunque indispensabili quali: l'amministrazione del personale, gestione dei locali e molto altro.

Il team IT si occupa invece della manutenzione e della gestione del parco macchine PC aziendale. Gestisce tecnicamente i server e tutti i vari servizi (esempio: mail, MantisBT).

Ogni reparto ha il proprio leader che sovrintende e gestisce i lavori. Ricorsivamente ogni reparto è suddiviso in team, i quali hanno a loro volta il proprio leader. I leader sono coloro che si occupano della pianificazione. Nel momento in cui la direzione necessita di una pianificazione su un nuovo progetto essa la chiede ai tre leader principali (programmazione, artist e game designer), i quali a loro volta la girano ai sotto leader e, ottenuti i risultati delle varie parti compongono la stima totale.

Esiste un'altra organizzazione di leader trasversale ma complementare alla precedente. Ogni reparto ha un team leader per progetto (videogame). Questa organizzazione è detta “matriciale” ed è importantissima poiché i giochi contemporaneamente sviluppati sono sempre almeno 2 e, mentre i team leader normali hanno la visione generale, i team leader per game mantengono l'obiettivo del singolo game evitando di disperdere le energie.

Il team di QA ha il compito di scovare e segnalare tutti i *bug*_g presenti sui prodotti.



2.3. SUDDIVISIONE DELLA FORZA LAVORO

Essi tracciano tutti gli errori scovati tramite l'utilizzo del tool open source MantisBT⁶. MantisBT è un famoso strumento che permette la segnalazione di bug e la gestione completa fino alla risoluzione. Esso è stato personalizzato e adattato alle esigenze specifiche dell'azienda, ottenendo uno strumento semplice ma potente.

La procedura prevede che il personale del QA segnali il bug scovato e tutti i dettagli per riconoscerlo e riprodurlo, inoltre ne assegna la risoluzione al team di programmazione più pertinente. A questo punto il team leader che riceve il bug decide a chi farlo risolvere all'interno del proprio team, seguendo la regola di assegnarlo alla persona che si suppone possa sistemarlo nel minor tempo possibile. È prevista la possibilità che il team leader riassegni il bug ad un altro reparto, ma solo se il team di QA abbia sbagliato oppure se la risoluzione avverrà più velocemente dal nuovo team. La direzione tiene traccia di questi "rimbalzi" allo scopo di evitare che il principio di minimo tempo di risoluzione non venga mai meno. Un'ulteriore possibilità è che il team leader, insieme alla direzione, valuti la risoluzione dell'errore di non importanza strategica. A questo punto il bug non verrà mai risolto. Quando il reparto di programmazione segnala un bug come sistemato, il team QA si assicura che il problema sia stato davvero rimosso, altrimenti la procedura ricomincia d'accapo.

Il reparto di sviluppo si divide in 3 reparti: programmazione, artist e game design. Il reparto di artist è quello che si occupa della creazione degli asset per i giochi. I game designer sono coloro che ideano i giochi e decidono come saranno. Essi descrivono dettagliatamente tutti i contenuti, i flussi e le feature che vorrebbero avere in un gioco. Dopodiché i team di programmazione, gli artist e la direzione stimano il tempo necessario e nel caso tagliano qualche contenuto.

Il reparto di programmazione si divide nei seguenti team: UI, Online, programmazione 3D, gameplay e R&D (ricerca e sviluppo).

Il team UI scrive e mantiene le pagine di interfaccia utente. Inoltre si occupa dalla gestione della grande macchina a stati finiti che rappresenta il gioco ed il passaggio e la consistenza dei dati in essa (esempio: i dati prestazionali di una moto). Gestisce i salvataggi e trasforma l'input utente in azioni in base al contesto nel quale sono stati premuti. Ad esempio la pressione del tasto X su tutte le Playstation® seleziona l'azione di l'azione corrente quando un menu è visualizzato, mentre scala una marcia durante lo stato di gara.

Il team Online, come suggerisce il nome, gestisce la componente multiplayer online di ogni videogame. Si occupa dal creare e configurare le partite (match making) alla comunicazione di rete durante le gare, fino alla gestione delle $leaderboard_{|g|}$. La comunicazione online di basso livello si basa sul $middleware_{|g|} Raknet_{|g|}$ ⁷.

Il team di gameplay gestisce le meccaniche di gioco in race. Si occupa della simulazione fisica della gara e della gestione dell'AI (Artificial Intelligence) che controlla gli avversari del giocatore.

Il reparto di ricerca e sviluppo si occupa dello sviluppo dell'engine grafico, ricercando sempre nuovi soluzioni e tecniche per mantenerlo all'avanguardia nel panorama mondiale. Tutti i giochi si basano sull'engine grafico, denominato GEM. Lo sviluppo di GEM avviene parallelamente allo sviluppo dei giochi, semplicemente quando parte la programmazione di un nuovo gioco viene usata l'ultima release stabile. Questo team si occupa inoltre di tutti gli strumenti a supporto dello sviluppo grafico. Sua responsabilità è lo sviluppo e la manutenzione degli editor di gioco, quali ad esempio

⁶<https://www.mantisbt.org/>

⁷<http://www.jenkinssoftware.com/>



2.4. GAME WORKFLOW

il GEM Editor, giunto alla release 2.

L'organizzazione dei task per ciascun impiegato è gestita tramite il tool *Microsoft Project_g*⁸. Come già spiegato la cura dei task è gestita gerarchicamente dai team leader che assegnano al proprio team.

Il team dove lo stagista è stato inserito è il team 3D, il quale, ad alto livello, utilizza l'engine grafico per disegnare le scene del gioco. Verrà maggiormente approfondito il ruolo del team successivamente alla spiegazione delle fasi di sviluppo di un videogame in Milestone S.r.l.⁹

2.4 Game workflow

Lo sviluppo di tutti i giochi in Milestone S.r.l. segue sempre le seguenti fasi:

1. Tutto parte da un **Macro Design** del gioco. Se il gioco è completamente nuovo, oppure se si basa su un altro già completato evolvendolo, si procede a identificare i punti chiave e le feature innovative che si vuol inserire. Con queste in mano si effettua, con le procedure prima descritte, ad una stima di tempo e fattibilità del gioco.
2. Approvato il progetto si procede alla creazione di un **Concept Design** e contemporaneamente alla **pre-production**. Il concept design è un approfondimento, di dettaglio, di come sarà il gioco, sarà la base ufficiale sulla quale verrà tecnicamente progettato e sviluppato. Una versione più ridotta viene inviata a Sony e Microsoft, in modo tale da ottenere il loro permesso per poter pubblicare quel gioco sulle loro console. La pre produzione è invece l'inizio della progettazione della struttura degli asset e l'inizio della loro creazione. Si vuole informare il lettore che la quantità di asset necessaria per lo sviluppo di un titolo *AAA_g* è davvero enorme. L'azienda procederà quindi ad appaltare esternamente lo sviluppo di parte di questi.
3. Terminata la fase di design si procede alla creazione di un **prototipo** nel quale saranno presenti le feature più innovative e complesse del gioco. È da questo momento che la programmazione ha inizio. Le feature inserite sono solitamente stabili e testabili. Il prototipo è un ibrido tra incrementale ed usa e getta, ad esclusivo utilizzo interno. L'utilità di questo prototipo è saggiare se le nuove feature, le più complesse, sono fattibili nei tempi pianificati e soprattutto verificare la realizzabilità. Procedere subito a testarle permette di poter correggere subito il tiro in caso di problemi, evitando il fallimento del progetto, inevitabile se problemi del genere fossero scoperti troppo tardi.
4. Questa fase è un mix tra una classica **First Playable_g** ed una **Vertical Slice_g**. L'azienda ha deciso di unificare queste due fasi che solitamente tendono ad essere separate. La motivazione è prettamente strategica, in quanto la realizzazione del prototipo ha provveduto già ad evidenziare le maggiori difficoltà e dato l'alto numero di giochi sviluppati contemporaneamente, una compressione a questo punto risulta del tutto accettabile. L'esperienza del team sui giochi di corsa, aiuta inoltre a poter gestire l'unione delle due fasi. Da questa fase esce quindi un prototipo incrementale ad uso pubblico. Esso è molto più simile ad una first playable poiché gli asset, vista la loro vastità, tendono ad arrivare in versione finale molto tardi. Solitamente viene mostrato ai giornalisti di tutto il mondo. Si vuol evidenziare come a partire da questa fase la velocità di sviluppo

⁸<http://office.microsoft.com/en-001/project/>

⁹Si veda la sezione 2.5.



2.5. RUOLO DEI 3D PROGRAMMER NEL WORKFLOW

cresce repentinamente, così come l'impegno complessivo dedicato al gioco in termini di risorse investite.

5. A questo punto si arriva alla cosiddetta versione **Alpha** del gioco in cui tutti gli asset presenti richiedono memoria e prestazioni in gioco come i definitivi. Per evitare equivoci tutto ciò che non è definitivo viene marcato come tale. In questo modo sia i giornalisti che i developer possono riconoscere e ricordare ciò che deve essere completato. Le feature sono state tutte inserite, magari non complete e/o completamente stabili ma presenti. È possibile così valutare le prestazione e le performance di gioco e, nel caso venissero riscontrati problemi c'è ancora tempo per risolverli.
6. La **Beta** presenta tutti gli asset nella loro versione finale. Tutte le feature iniziano a diventare stabili ed il reparto di QA svolge un ruolo molto importante in questa fase, alla ricerca di tutti i bug.
7. Quando la direzione ritiene il gioco abbastanza stabile, si procede alla **Submission** tramite il reparto di publishing, il quale inoltra una copia del gioco ai console owner, Microsoft e Sony, allo scopo di ottenere l'approvazione per la pubblicazione definitiva del gioco. Per ottenerla, i giochi sono testati, secondo le TRC/TCR per ciascuna console. Questa fase è davvero delicata in quanto ogni submission richiede circa due settimane ed in caso di fallimento, bisogna rifare la procedura fino ad ottenere l'approvazione. Fallire la submission comporta innanzitutto una preziosa perdita di tempo e secondariamente, una perdita economica. In quanto, per ciascun gioco, per ciascuna piattaforma, le submission dalla terza in poi iniziano a costare davvero salate e conseguentemente a pesare sul budget. È quindi priorità dell'azienda superare la submission al primo tentativo.
8. **Gold** è il nome della versione finale, completamente stabile senza bug gravi, pronta per essere distribuita worldwide.

2.5 Ruolo dei 3D programmer nel workflow

Durante la seconda fase i 3D programmer progettano insieme agli artisti la struttura sulla quale verranno creati gli asset. Per essere concreti, ad esempio, dell'asset di una moto, decidono in quanti e quali modelli sarà suddivisa la $mesh_{|g|}$. Come saranno le $texture_{|g|}$ e molto altro. Fissata la struttura potranno poi gli artisti (interni ed esterni), creare tutti gli asset necessari.

Durante tutto lo sviluppo essi scrivono e/o aggiornano il codice necessario alla gestione in memoria degli asset. Questo è un compito molto delicato in quanto deve tener conto di diverse necessità ed effettuare una sintesi scrupolosa. La necessità primaria è la velocità di caricamento, in un mondo ideale si vorrebbe avere tutti gli asset caricati in memoria centrale pronti per essere usati (todo mettere riferimento al capitolo 3 in cui si introduce una strategia per svolgere questo compito). Nella realtà invece il programmatore si scontra con i diversi limiti hardware delle varie piattaforme target, è quindi costretto a implementare logiche intelligenti e molto efficienti.

Caricare gli asset in memoria significa inoltre riassemblarli. Per non sprecare spazio, ogni componente degli asset che serve più di una volta, viene memorizzata una volta sola nei dati grezzi. Questo significa che per ricomporre un oggetto 3D, bisogna recuperare tutti i sotto componenti e ricomporre la struttura ad albero originale. Esiste ovviamente un database, nei giochi Milestone scritto in linguaggio $XML_{|g|}$, in cui è scritto come ricomporre le strutture originali degli asset.



2.5. RUOLO DEI 3D PROGRAMMER NEL WORKFLOW

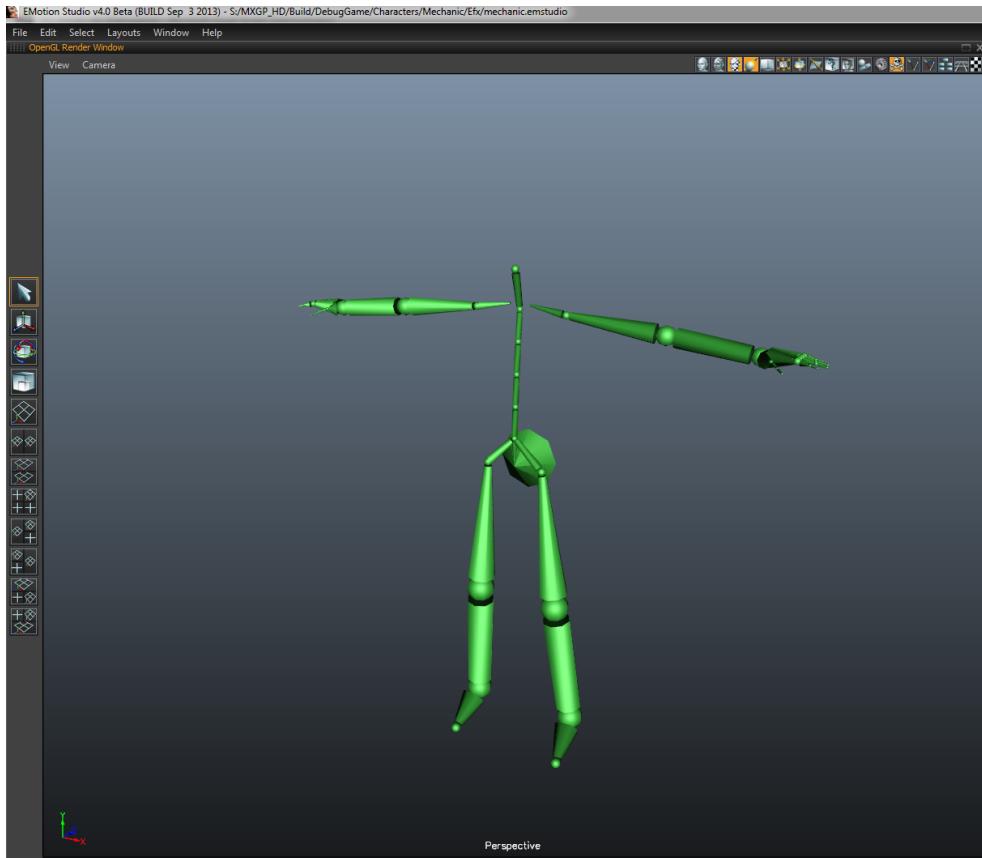


figura 2.1: Screenshot di EMotion Studio durante la visualizzazione dello skeleton di animazione di un meccanico in T-pose

Altra responsabilità è l'utilizzo dell'engine GEM per disegnare le scene 3D richieste dal design del gioco. Il mondo di gioco viene gestito a “World”, questo significa che tutti gli oggetti 3D sono contenuti all'interno di questo mondo. Compito del team è scrivere un'implementazione efficiente del World per disegnare tutti gli oggetti presenti.

I World si dividono in ambienti di gara e non. Negli ambienti non di gara il team gestisce anche le cosiddette **cut-scene**, esse sono le scene in cui il giocatore è spettatore passivo della scena 3D. Un esempio è la sequenza animata che simula la cerimonia del podio alla fine di una gara. In Milestone S.r.l. esistono due sistemi di gestione delle cut-scene. Il primo, in ordine temporale di adozione, prevede la descrizione della scena in linguaggio XML (MotoGP 14), mentre il secondo utilizza degli script $LUA_{|g|}$ ¹⁰ (MXGP).

Ultimo ma non meno importante compito, è la gestione degli ambienti di gara. In essi il team gestisce il collegamento degli oggetti 3D con la simulazione fisica, la quale evolve i parametri degli oggetti di gioco. Tramite questi parametri si richiede l'esecuzione dell'animazione, la quale aggiorna i parametri visibili degli oggetti 3D, ad esempio la posizione.

Per eseguire le animazioni il team utilizza il middleware EMotion FX 4¹¹, il quale dato il blend tree sviluppato dagli artisti, esegue in real time le animazioni sugli oggetti

¹⁰Per maggiori approfondimenti [MM13] e [S1d]

¹¹<http://www.mysticgd.com/website/index.php?id=17>



2.6. CONFIGURATION E DATA MANAGEMENT

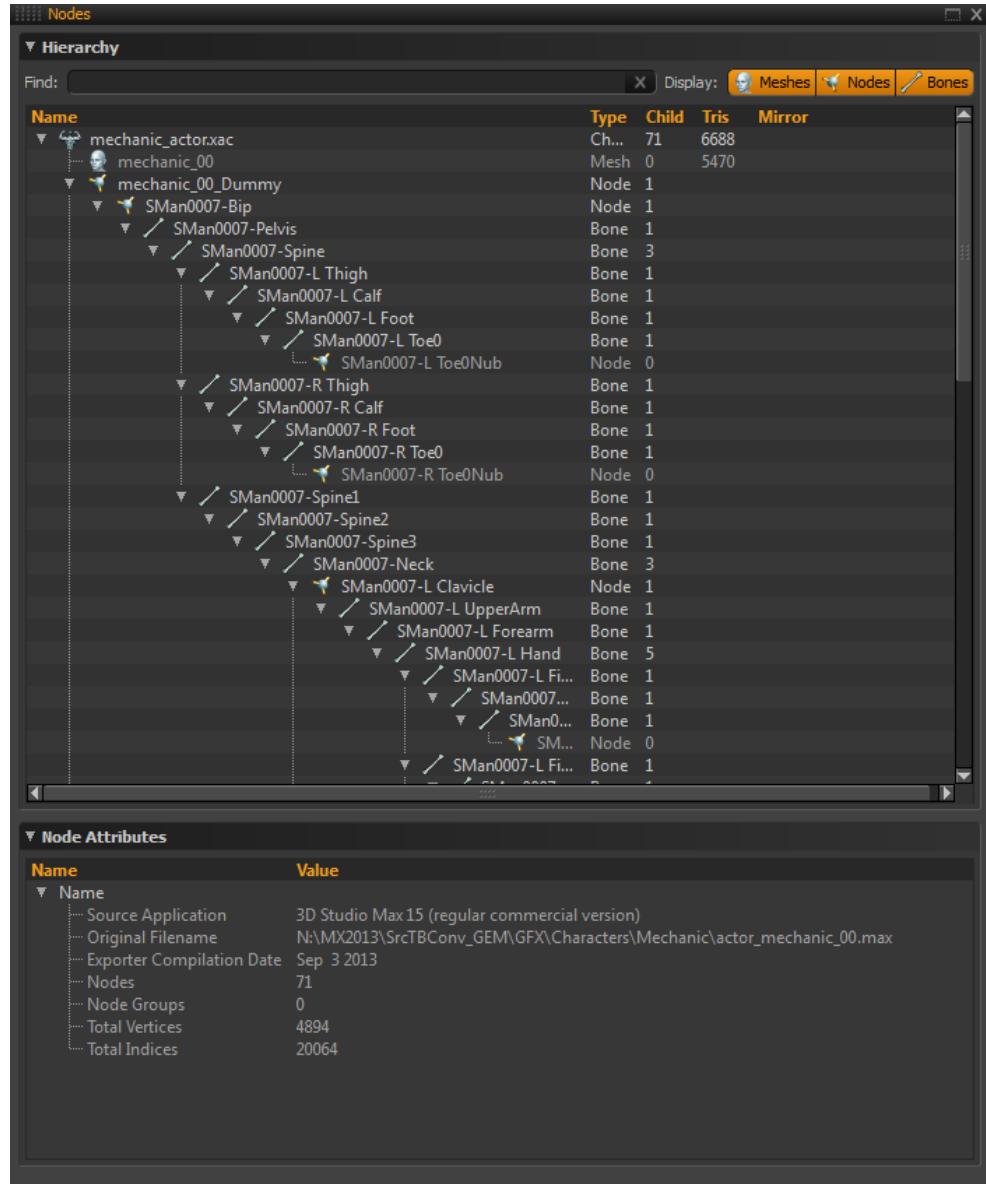


figura 2.2: Screenshot di EMotion Studio durante la visualizzazione della gerarchia dei bone di animazione dello skeleton di un meccanico

3D. Gli artisti, in Milestone, sviluppano i blend tree tramite il tool EMotion Studio fornito dalla stessa casa.

2.6 Configuration e data Management

Il software di versionamento utilizzato in Milestone è *Perforce®_{|g|}*¹², che recentemente ha rimpiazzato *Alienbrain®_{|g|}*. Un punto di forza che distingue Perforce® dagli altri software di versionamento sono gli Stream. Questi sono dei branch intelligenti, in altre parole, al posto di effettuare una copia vera e propria per creare un branch, come fanno ad esempio Git® ed SVN®, qui sono gestiti interamente in termini di *delta_{|g|}* rispetto

¹²Per il lettore appassionato è disponibile una semplice ma efficacie introduzione a Perforce® al seguente link: <http://www.perforce.com/perforce/doc.current/manuals/intro/intro.pdf>



2.7. ASSET

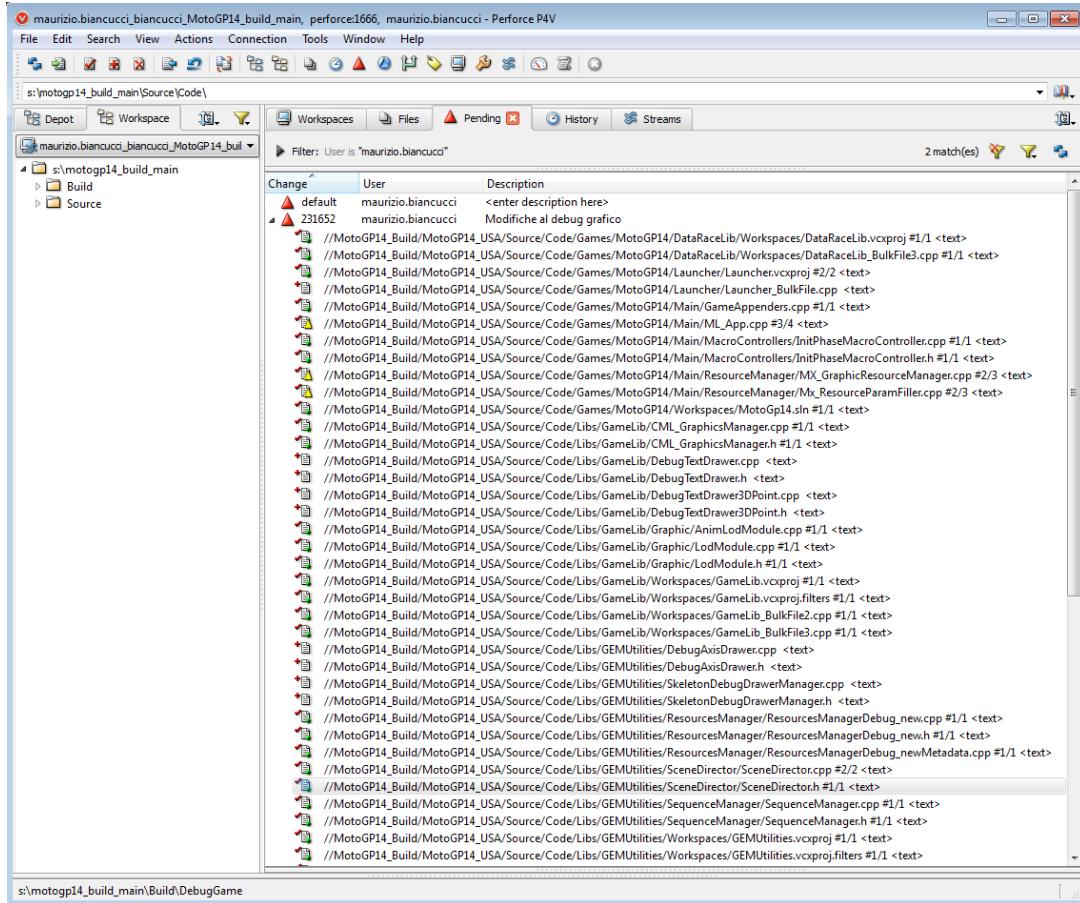


figura 2.3: Screenshot di Perforce

al branch master. Altra particolarità è la modalità di commit su shelve (scaffale). Tale modalità permette di eseguire il commit (check-in in Perforce[®]) su uno spazio personale. Questo significa che un altro programmatore quando eseguirà la get (pull in Git[®], update in SVN[®]) non otterrà la versione in shelve. Dovrà manualmente richiederla per poterla avere. Questa modalità risulta molto utile ad esempio per eseguire il commit di alcune modifiche che non sono ancora state testate, oppure sono una soluzione alternativa in attesa di approvazione.

2.7 Asset

Come già accennato, gli asset subiscono una serie di trasformazioni prima di arrivare nella forma che viene distribuita insieme al gioco. Essi possono essere trovati in uno dei seguenti stati:

1. **rough**: i dati sono nella forma originale fornita dall'artista. Tipicamente sono nel formato nativo del programma con il quale sono stati sviluppati.
2. **middle**: i dati sono parzialmente ottimizzati, tramite compressione e *serializzazione*_[g] (se ad esempio il formato originale era XML).
3. **final (mixed)**: i dati sono stati unificati in pochi file chiamati “mixed”. Da migliaia di file si passa ad una manciata di grossi file con estensione “.mix”.

La trasformazione degli asset nel formato finale ha un duplice scopo. Essendo ci pochi ma grandi file le performance del gioco in fase di caricamento aumentano



2.8. CODE BUILD

notevolmente¹³. Inoltre, avendo convertito gli asset in un formato proprietario (il “.mix”), si corre meno il rischio che gli asset possano essere “rubati” e riutilizzati senza il permesso dell’azienda.

In Milestone S.r.l. è presente un *sistema di integrazione continua*_{|g|} che ogni notte procede a eseguire le *build*_{|g|} di tutti gli asset aggiornati. Viene eseguita una build per ogni stato disponibile. Ogni build termina con una mail collettiva (agli interessati) con gli esiti. Nel caso la build fosse stata rotta, viene mandata una mail a tutti coloro che hanno contribuito a modificare l’asset fallito. La procedura di build è stata settata per utilizzare tutti i PC dell’azienda contemporaneamente. Questo è molto importante poiché, vista l’enorme quantità di asset da convertire, di molteplici giochi, è l’unico modo per poter riuscire a finire tutto entro la mattina successiva. La pecca di questo sistema è che, per poter visionare e testare le modifiche agli asset nei formati n°2 e 3, bisogna per forza aspettare il giorno lavorativo successivo.

2.8 Code build

Tutti i giochi Milestone sono sviluppati contemporaneamente per molte piattaforme¹⁴. Questo comporta l’utilizzo contemporaneamente di molti compilatori. Ogni piattaforma ha il suo compilatore, ad esempio sia Sony, sia Microsoft hanno sviluppato un compilatore proprietario per le proprie console. Questo comporta una sforzo aggiuntivo da parte di tutti i programmati, i quali si devono assicurare che il codice scritto compili e funzioni correttamente su ciascuna piattaforma prima di eseguire la commit.

Sono state definiti diversi target di compilazione, i quali variano per quantità di ottimizzazione e capacità di interazione con i diversi stati degli asset:

1. **debug**: build completamente non ottimizzata a totale scopo di debug;
2. **release**: non standard, è parzialmente ottimizzata ma contiene ancora i simboli per il debug;
3. **official**: build completamente ottimizzata. Si invita il lettore a prestare attenzione alla differenza con lo stato “final” degli asset.

I target debug e release hanno la capacità di leggere gli asset in tutti gli stati possibili. La official invece, può utilizzare solo le versioni final. Tipicamente le combinazioni utilizzate sono le seguenti:

1. **debug/release target + rough asset**: per debug. Soffre della lentezza di caricamento determinata dagli asset divisi in tantissimi piccoli file. Date le limitate capacita di PS3®, 256 MB di RAM, capita che verso le fasi finali di sviluppo non si riesca ad avviare questa combinazione.
2. **official target + final asset**: è la versione per l’utente finale e solitamente usata dal QA.
3. **debug/release target + final asset**: molto più veloce a caricarsi, è utilissima per testare una modifica sul codice che non coinvolge gli asset.

Come il lettore può aver intuito, il codice di gioco si compone di moltissime librerie, le quali contengono complessivamente migliaia di file. Allo scopo di ridurre significativamente i tempi di compilazione, si è adottata la tecnica di compilazione **bulk**. Essa consiste semplicemente nel creare un unico file per libreria, nel quale si

¹³Si ricorda al lettore la lentezza degli hard disk tradizionali, o anche peggio dei CD, nel leggere tanti piccoli file, anche se fisicamente contigui

¹⁴link a dove si dice quali sono le piattaforme



includono tutti i .cpp. Altra tecnica usata è l'utilizzo delle **forward declaration** nei file header. Diminuendo gli include e sostituendoli con le dichiarazioni incomplete, si riesce a tenere sotto controllo l'esplosione dei tempi di compilazione.

Al momento, tutti i giochi Milestone per PC si basano sul compilatore di Visual Studio 2010, il quale supporta in maniera sperimentale il C++ 11. È scelta aziendale quella di rimandare l'utilizzo del C++ 11 a quando l'implementazione diventerà quasi identica fra i compilatori attualmente usati. Si ipotizza di iniziare ad usarlo quando si passerà all'ultima versione del compilatore di Microsoft.

Anche per il codice di gioco, è in atto un sistema di integrazione continua, il quale usa le stesse procedure di quello utilizzato per gli asset.¹⁵.

2.9 Metadati

I giochi Milestone utilizzano il sistema dei metadati. Lo scopo è di ottenere delle funzionalità simili alla riflessione anche in C++. Queste funzionalità spaziano dal real-time editing delle variabili, update da e verso XML e/o binari. Questa tecnica permette uno scambio di dati disaccoppiando le due classi a discapito del controllo degli accessi. È infatti possibile esporre come metadati anche membri di classe privati.

Tutti i metadati presenti sono esposti via rete, solo nei target debug e release, nel **Beholder**. Questo è un client che permette la visualizzazione e l'editing in tempo reale dei metadati. Un efficace esempio di utilizzo è l'esposizione nel beholder di dati XML, seguito dal tuning in real time di codesti dati, e il salvataggio automatico dei nuovi valori se soddisfatti del risultato.

Un altro esempio di utilizzo del Beholder, molto usato dal team 3D, è l'esposizione di funzioni attivabili direttamente dall'interfaccia grafica di quest'ultimo. Le più usate sono quelle per disegnare geometrie di debug, quali ad esempio gli assi di origine degli spazi di riferimento degli oggetti presenti nelle scene 3D.

L'utilizzo dei metadati all'interno del codice avviene tramite un set di macro definite in GEM. Un facile esempio di utilizzo è il seguente.

Listing 2.1: Dog.h

```
class Dog
{
public:
    #define ENABLE_METADATA;

    // methods
    void Walk();
    void Jump();

    // events
    TypedEvent<> OnFall;
    TypedEvent< GString > OnDie;

private:
    int m_age;
    float m_weight;
};
```

¹⁵Per ulteriori dettagli si invita il lettore a consultare la sezione 2.7



2.10. NORME DI CODIFICA

Di seguito è presente il codice per eseguire l'esposizione di tutta la classe nei metadati.

Listing 2.2: Dog.cpp

```
META_BEGIN_CLASS( Dog )

META_BEGIN_PROPERTIES
META_NAMED_PROPERTY("Age", m_age),
META_NAMED_PROPERTY("Weight", m_weight)
META_END_PROPERTIES

META_BEGIN_ACTIONS
META_ACTION( Walk ),
META_ACTION( Jump ),
META_END_ACTIONS

META_BEGIN_EVENTS
META_EVENT( OnFall ),
META_EVENT( OnDie ),
META_END_EVENTS

META_END_CLASS
```

2.10 Norme di codifica

Allo scopo di uniformare il codice prodotto ed aumentare la comprensibilità, l'azienda ha deciso di adottare la notazione ungherese, alla quale sono state applicate delle piccole personalizzazioni.

Tutto il codice è scritto in lingua inglese, compresi i commenti e le annotazioni. Di seguito viene presentato un piccolo riassunto delle regole della notazione adottata in Milestone S.r.l.

- **prefissi:**

- **m:** usato per le variabili membro (esempio: `m_keyName`);
- **s:** usato per le variabili membro statiche (esempio: `s_numInstance`);
- **p:** usato per le variabili puntatore (esempio: `m_pkeyNameList`);
- **sigla del tipo:** usato per indicare il tipo delle variabili di tipi primitivi (esempio: `uiGroupIndex`);
- **i:** usato per le variabili input alla funzione che sono usate come dati di input(esempio: `i_resource`);
- **o:** usato per le variabili input alla funzione che sono usate come output (esempio: `o_foundKeys`).

- **capitalizzazione:**

- **variabili e parametri:** iniziano sempre in minuscolo e usano una lettera minuscola per ogni parola (esempio: `m_parentWidget`, `returnValue`);
- **macro ed enum:** completamente in maiuscolo con le diverse parole separate dal carattere ‘_’ (esempio: `UPDATE_CODE`);
- **classi, funzioni e metodi:** iniziano sempre in maiuscolo e ogni parola diversa parte in maiuscolo (esempio: `CObject3d`);



2.11 Assert e Log

GEM fornisce anche un utilissimo meccanismo di logging. Esso viene messo a disposizione tramite la classe statica LogManager. Il programmatore deve semplicemente settare il contesto, la severità e il messaggio. La vera importanza del sistema di logging unificato deriva dal fatto che, in questo modo, ciascun programmatore può settare i contesti per i quali vuole visualizzare i log.

Altra funzionalità offerta da GEM sono un set di assert personalizzato. Questi assert sono scartati dalla compilazione nei target release e official. Un corretto utilizzo degli assert è quello di inserirne uno per ciascuna precondizione. In altre parole, molto spesso, le funzioni si basano su delle condizioni che assumono vere al momento dell'invocazione, condizioni che sono alla base della correttezza dell'algoritmo. Se invece di spiegarle con commenti testuali, si utilizza una assert, esse sono sempre verificate a run-time e aiutano efficacemente a scovare bug molto difficili da trovare con altri mezzi.

Capitolo 3

Multiplatform File Analyzer

In questo capitolo è presente una analisi completa del tool Multiplatform Fyle Analyzer

3.1 Introduzione

3.1.1 Premessa

Tutti i giochi multipiattaforma prodotti da Milestone sono saggiamente organizzati in modo che ogni qual volta debbano ricercare un file dato un percorso, prima eseguono la ricerca nella cartella specifica della piattaforma di esecuzione e poi, se non presente, nel percorso originale dato¹. Grazie a questa organizzazione si riesce facilmente a differenziare gli asset in maniera molto semplice e veloce ogni qual volta sia necessario. La differenziazione degli asset nasce principalmente dalle differenti specifiche tecniche e capacità di calcolo di ciascuna piattaforma che costringono a creare versioni apposite. Ovviamente la specializzazione è un costo aggiuntivo in termini di tempo di creazione e di manutenzione e va eseguita il meno possibile.

3.1.2 Lo scopo

Inizialmente il metodo adottato funzionava senza problemi evidenti ma, al veloce crescere del numero dei file e delle piattaforme target², si è notata l'esigenza di avere un maggior controllo.

Multiplatform File Analyzer è stato realizzato appositamente per rimediare a questo problema, offrendo una panoramica semplice ma completa riguardo le differenti versioni di ogni file specializzato per almeno una piattaforma.

3.2 Casi d'uso

Per meglio capire e tracciare l'esperienza d'uso che un developer di un gioco multipiattaforma avrà con il tool sono stati creati dei diagrammi di casi d'uso gerarchici.

I diagrammi di casi d'uso fanno parte della famiglia dei diagrammi UML e descrivono le funzionalità offerte dal prodotto così come sono percepite dagli attori che interagiscono con il sistema.

Ogni caso d'uso ha un codice univoco gerarchico, nella forma:

UC[codice univoco del padre].[codice progressivo di livello]

¹Lo stesso sistema è utilizzato anche per il caricamento dei file contenenti i testi specifici per ogni differente localizzazione del gioco.

²Le piattaforme target sono attualmente sei: *Playstation Vita*[®], *Playstation 3*[®], *Xbox 360*[®], *Playstation 4*[®] e *Xbox One*[®].



3.2. CASI D'USO

Il codice progressivo può includere diversi livelli di gerarchia separati da un punto.

3.2.1 UC 1: Caso d'uso principale

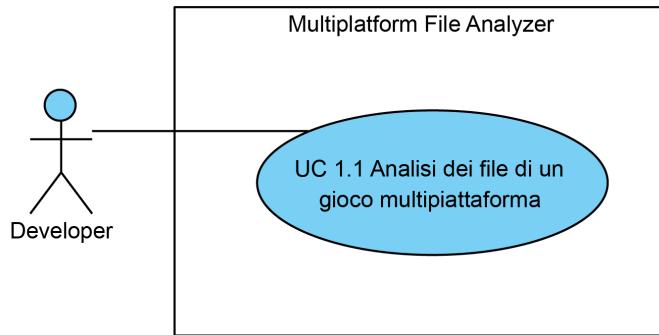


figura 3.1: Use Case - UC 1: Caso d'uso principale

- **Attori:** developer.
- **Descrizione:** un developer può eseguire un'analisi dei file di un gioco multipiattaforma.
- **Precondizione:** il developer abbia lanciato il tool sotto un sistema Windows 7 o superiore.
- **Flusso principale degli eventi:**
 1. *Analisi dei file di un gioco multipiattaforma* (UC 1.1).
- **Postcondizione:** il sistema ha erogato le funzionalità richieste dal developer.

3.2.2 UC 1.1: Analisi dei file di un gioco multipiattaforma

- **Attori:** developer.
- **Descrizione:** un developer deve poter inserire i dati necessari all'analisi, visualizzare eventuali errori oppure il risultato dell'analisi.
- **Precondizione:** il developer abbia lanciato il tool sotto un sistema Windows 7 o superiore.
- **Flusso principale degli eventi:**
 1. *Inserimento dei dati in input all'analisi* (UC 1.1.1);
 2. *Avvio dell'analisi* (UC 1.1.2);
 3. *Visualizzazione del risultato dell'analisi* (UC 1.1.4).
- **Estensioni**
 1. *Visualizzazione errori sui dati in input* (UC 1.1.3).
- **Postcondizione:** il sistema ha erogato le funzionalità richieste dal developer.

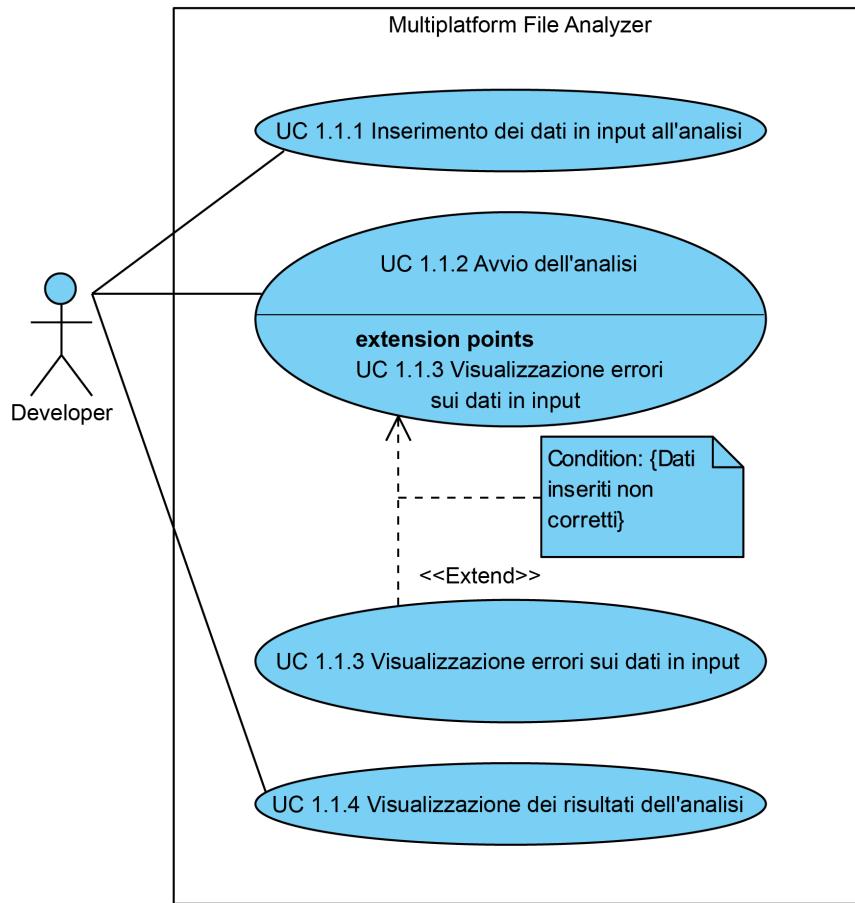


figura 3.2: Use Case - UC 1.1: Analisi dei file di un gioco multipiattaforma

3.2.3 UC 1.1.1: Inserimento dei dati in input all'analisi

- **Attori:** developer.
- **Descrizione:** un developer deve poter inserire i dati necessari all'analisi composti da un percorso di base, i nomi delle piattaforme e il tempo di warning.
- **Precondizione:** il developer abbia lanciato il tool sotto un sistema Windows 7 o superiore.
- **Flusso principale degli eventi:**
 1. *Inserimento percorso radice dei file da analizzare* (UC 1.1.1.1);
 2. *Inserimento dei nomi delle cartelle delle piattaforme* (UC 1.1.1.2);
 3. *Inserimento del warning time* (UC 1.1.1.3).
- **Postcondizione:** il sistema ha preso in carico i dati che verranno usati per la prossima analisi.

3.2. CASI D'USO

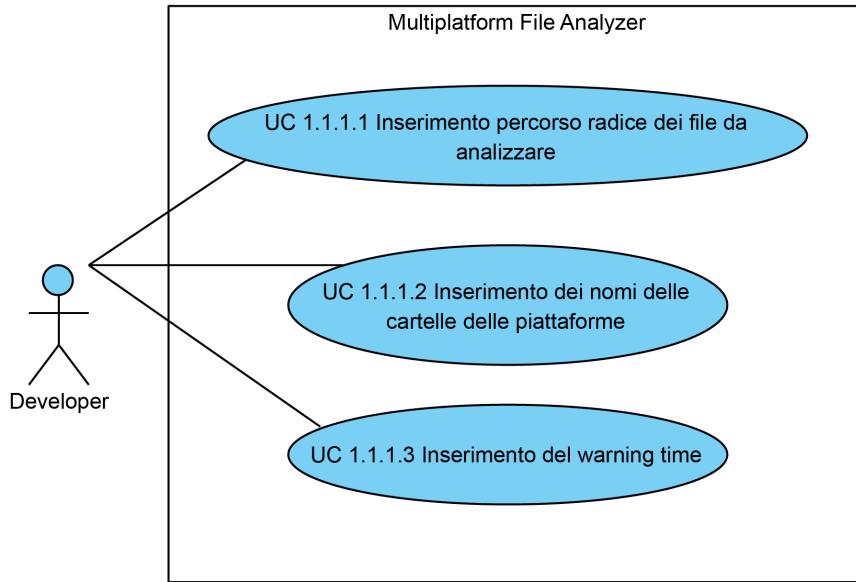


figura 3.3: Use Case - UC 1.1.1: Inserimento dei dati in input all'analisi

3.2.4 UC 1.1.1.1: Inserimento percorso radice dei file da analizzare

- **Attori:** developer.
- **Descrizione:** il developer deve poter inserire il percorso di base dell'analisi.
- **Precondizione:** il developer abbia lanciato il tool sotto un sistema Windows 7 o superiore.
- **Scenario principale:** il developer sceglie il percorso di base da cui l'analisi partirà.
- **Postcondizione:** il sistema ha preso in carico il percorso di base inserito che verrà usato per la prossima analisi.

3.2.5 UC 1.1.1.2 Inserimento dei nomi delle cartelle delle piattaforme

- **Attori:** developer.
- **Descrizione:** il developer deve poter inserire i nomi delle piattaforme da analizzare.
- **Precondizione:** il developer abbia lanciato il tool sotto un sistema Windows 7 o superiore.
- **Scenario principale:** il developer sceglie i nomi delle piattaforme.
- **Postcondizione:** il sistema ha preso in carico i nomi delle piattaforme inserite che verranno usate per la prossima analisi.

3.2.6 UC 1.1.1.3 Inserimento del warning time

- **Attori:** developer.
- **Descrizione:** il developer deve poter inserire il tempo di warning.



3.2. CASI D'USO

- **Precondizione:** il developer abbia lanciato il tool sotto un sistema Windows 7 o superiore.
- **Scenario principale:** il developer il tempo superato il quale vuole essere avvertito.
- **Postcondizione:** il sistema ha preso in carico il tempo di warning che verrà usato per la prossima analisi.

3.2.7 UC 1.1.2 Avvio dell'analisi

- **Attori:** developer.
- **Descrizione:** il developer deve poter avviare la ricerca.
- **Precondizione:** il developer abbia lanciato il tool sotto un sistema Windows 7 o superiore.
- **Scenario principale:** il developer sceglie di avviare la ricerca.
- **Postcondizione:** il sistema inizia a eseguire l'analisi.

3.2.8 UC 1.1.3 Visualizzazione errori sui dati in input

- **Descrizione:** il developer ha commesso uno dei seguenti errori durante l'inserimento dei dati in input all'analisi:
 - il percorso inserito non è valido;
 - il developer non ha inserito nessuna piattaforma.
- **Precondizione:** il developer abbia avviato la ricerca.
- **Scenario principale:** viene fornita una spiegazione dell'errore commesso e come risolverlo.
- **Postcondizione:** l'errore commesso è stato visualizzato e spiegato, l'analisi non è stata avviata.

3.2.9 UC 1.1.4 Visualizzazione dei risultati dell'analisi

- **Attori:** developer.
- **Descrizione:** un developer deve poter inserire i dati necessari all'analisi composti da un percorso di base, i nomi delle piattaforme e il tempo di warning.
- **Precondizione:** il developer abbia avviato l'analisi e che questa abbia terminato senza comunicare nessun errore sui dati in input.
- **Flusso principale degli eventi:**
 1. *Visualizzazione di tutti i file presenti in almeno una cartella di piattaforma* (UC 1.1.4.1);
 2. *Visualizzazione del percorso per il file visualizzato* (UC 1.1.4.2);
 3. *Per ogni file visualizzato, visualizzazione del tempo di ultima modifica per ogni piattaforma per cui è presente* (UC 1.1.4.3);
 4. *Per ogni file visualizzato, visualizzazione della massima differenza fra i tempi di ultima modifica delle piattaforme in cui è presente* (UC 1.1.4.4);

3.2. CASI D'USO

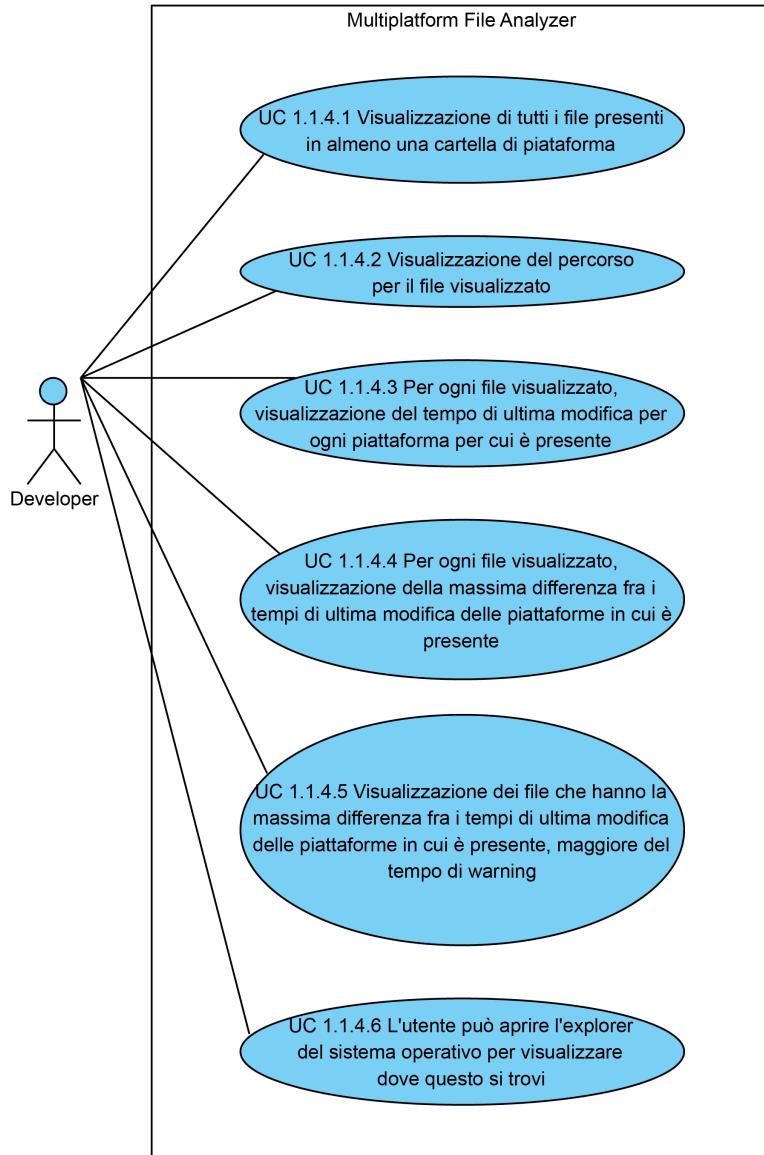


figura 3.4: Use Case - UC 1.1.4: Visualizzazione dei risultati dell'analisi

5. *Visualizzazione dei file che hanno la massima differenza fra i tempi di ultima modifica delle piattaforme in cui è presente, maggiore del tempo di warning* (UC 1.1.4.5);
6. *Il developer può aprire l'explorer del sistema operativo per visualizzare dove questo si trovi* (UC 1.1.4.6);

- **Postcondizione:** i dati dell'analisi sono stati visualizzati.

3.2.10 UC 1.1.4.1 Visualizzazione di tutti i file presenti in almeno una cartella di piattaforma

- **Attori:** developer.
- **Descrizione:** il developer deve poter visualizzare tutti i file che sono stati trovati in almeno una piattaforma.



3.2. CASI D'USO

- **Precondizione:** il developer abbia avviato l'analisi e che questa abbia terminato senza comunicare nessun errore sui dati in input.
- **Scenario principale:** vengono visualizzati i file presenti in almeno una piattaforma.
- **Postcondizione:** il risultato dell'analisi è stato visualizzato.

3.2.11 UC 1.1.4.2 Visualizzazione del percorso per il file visualizzato

- **Attori:** developer.
- **Descrizione:** il developer deve poter visualizzare il percorso del file visualizzato.
- **Precondizione:** il developer abbia avviato l'analisi e che questa abbia terminato senza comunicare nessun errore sui dati in input.
- **Scenario principale:** il developer visualizza il percorso del file visualizzato.
- **Postcondizione:** il percorso del file è stato visualizzato.

3.2.12 UC 1.1.4.3 Per ogni file visualizzato, visualizzazione del tempo di ultima modifica per ogni piattaforma per cui è presente

- **Attori:** developer.
- **Descrizione:** il developer deve poter visualizzare il tempo di ultima modifica per ciascuna piattaforma in cui il file è stato trovato.
- **Precondizione:** il developer abbia avviato l'analisi e che questa abbia terminato senza comunicare nessun errore sui dati in input.
- **Scenario principale:** viene visualizzata la data di ultima modifica per ciascuna piattaforma.
- **Postcondizione:** il risultato dell'analisi è stato visualizzato.

3.2.13 UC 1.1.4.4 Per ogni file visualizzato, visualizzazione della massima differenza fra i tempi di ultima modifica delle piattaforme in cui è presente

- **Attori:** developer.
- **Descrizione:** il developer deve poter visualizzare la massima differenza fra i tempi di ultima modifica delle piattaforme in cui è presente il file.
- **Precondizione:** il developer abbia avviato l'analisi e che questa abbia terminato senza comunicare nessun errore sui dati in input.
- **Scenario principale:** viene visualizzata la massima differenza fra i tempi di ultima modifica.
- **Postcondizione:** la massima differenza fra i tempi di ultima modifica è stata visualizzata.



3.3. TRACCIAMENTO DEI REQUISITI

3.2.14 UC 1.1.4.5 Visualizzazione dei file che hanno la massima differenza fra i tempi di ultima modifica delle piattaforme in cui è presente, maggiore del tempo di warning

- **Attori:** developer.
- **Descrizione:** il developer deve poter visualizzare e riconoscere i file per i quali la massima differenza fra i tempi di ultima modifica è maggiore del tempo di warning.
- **Precondizione:** il developer abbia avviato l'analisi e che questa abbia terminato senza comunicare nessun errore sui dati in input.
- **Scenario principale:** vengono visualizzati tutti i file che superano il tempo di warning.
- **Postcondizione:** i file che superano il tempo di warning sono stati visualizzati.

3.2.15 UC 1.1.4.6 L'utente può aprire l'explorer del sistema operativo per visualizzare dove questo si trovi

- **Attori:** developer.
- **Descrizione:** il developer deve poter aprire il programma che permettere l'esplorazione del file system nel punto in cui è presente il file correntemente selezionato.
- **Precondizione:** il developer abbia avviato l'analisi e che questa abbia terminato senza comunicare nessun errore sui dati in input.
- **Scenario principale:** viene visualizzato il file nella sua posizione all'interno del file system.
- **Postcondizione:** il file viene visualizzato nella sua posizione all'interno del file system.

3.3 Tracciamento dei requisiti

Partendo dai casi d'uso si è provveduto a stilare una precisa analisi dei requisiti per il tool in questione. I requisiti trovati sono stati inoltre tracciati in relazione al caso d'uso di origine.

Di seguito vengono riportati tutti i requisiti individuati. Per essere più leggibili verranno separati in tabelle a seconda della loro categoria. Di ogni requisito verranno indicati: tipologia, importanza e provenienza.

I requisiti dovranno essere classificati per tipo e importanza e utilizzeranno la seguente sintassi:

R[Importanza][Tipo][Codice]

- **Importanza:** può assumere solo uno fra i seguenti valori:
 - 0: requisito obbligatorio;
 - 1: requisito desiderabile;
 - 2: requisito opzionale.
- **Tipo:** può assumere solo uno fra i seguenti valori:



3.3. TRACCIAMENTO DEI REQUISITI

- *F*: funzionale;
- *Q*: di qualità;
- *P*: prestazionale;
- *V*: vincolo.

- **Codice:** è il codice gerarchico univoco di ogni vincolo espresso in numeri (esempio: 1.3.2).

Per ogni requisito vengono inoltre specificati:

- **descrizione:** breve ma completa ed il meno ambigua possibile;

- **fonte:** può essere soltanto una o più tra le seguenti:

- *caso d'uso:* il requisito è stato extrapolato da un caso d'uso. In questo caso va indicato il codice univoco del caso d'uso. È possibile indicare come fonte più di un caso d'uso.
- *interno:* è stato ritenuto giusto aggiungere questo requisito per completezza.
- *tutor aziendale:* il requisito è stato espressamente richiesto dal tutor aziendale.

3.3.1 Requisiti funzionali

Requisito	Descrizione	Fonti
R0F1	Un utente può effettuare l'analisi dei file di un gioco multipiattaforma.	UC 1.1
R0F1.1	L'analisi richiede l'inserimento di dati in input.	UC 1.1.1
R0F1.1.1	L'analisi richiede l'inserimento di un percorso assoluto dal quale far partire l'analisi.	UC 1.1.1.1
R0F1.1.2	L'analisi richiede l'inserimento dei nomi delle cartelle identificative di file di piattaforma.	UC 1.1.1.2
R0F1.1.2.1	Il programma interpreta i nomi delle piattaforme in modo case-insensitive.	UC 1.1.1.2
R0F1.1.2.2	Il programma richiede l'inserimento dei nomi delle piattaforme come unica stringa utilizzando il carattere ';' come separatore.	UC 1.1.1.2
R0F1.1.3	L'analisi richiede l'inserimento di un tempo di warning, superato il quale, a fine l'analisi, verrà segnalato il problema se presente.	UC 1.1.1.3
R0F1.1.3.1	Il formato del tempo di warning accettato prevede l'utilizzo esclusivo di numeri in formato inglese e con massimo due cifre decimali.	UC 1.1.1.3



3.3. TRACCIAMENTO DEI REQUISITI

R0F1.1.4	Il programma deve memorizzare e pre inserire i dati della precedente analisi (se esistenti) negli appositi campi per l'input dei dati, anche se tra un'analisi e la successiva il programma è stato chiuso.	Interno
R0F1.3	Il programma mostra gli eventuali errori sui dati in input all'analisi trovati durante l'avvio della stessa.	UC 1.1.3
R0F1.3.1	Il programma ferma l'avvio dell'analisi se il percorso di base non viene fornito.	UC 1.1.3
R0F1.3.2	Il programma ferma l'avvio dell'analisi se il percorso di base inserito non è un percorso assoluto.	UC 1.1.3
R0F1.3.3	Il programma ferma l'avvio dell'analisi se il percorso di base inserito non esiste nel file system.	UC 1.1.3
R0F1.3.4	Il programma ferma l'avvio dell'analisi se il contenuto del percorso di base inserito non è leggibile.	UC 1.1.3
R0F1.4	Il programma permette la visualizzazione dei dati output dell'analisi.	UC 1.1.4
R0F1.4.1	Nell'output dell'analisi è presente ciascun file presente in almeno un cartella di piattaforma.	UC 1.1.4.1
R0F1.4.2	Per ciascun file presente nell'output è visibile la data di ultima modifica per ciascuna piattaforma in cui è presente.	UC 1.1.4.2
R0F1.4.3	Per ciascun file presente nell'output è indicato il percorso di relativo a partire dal percorso di base inserito in input all'analisi.	UC 1.1.4.3
R0F1.4.4	Per ciascun file presente nell'output è indicato il nome del file completo di estensione.	UC 1.1.4.4
R0F1.4.5	Per ciascun file presente nell'output è indicata la massima differenza tra le date di ultima modifica delle piattaforme per cui è presente	UC 1.1.4.5
R0F1.4.5.1	Se il tempo massimo per ciascuno file è superiore al tempo di warning inserito in input allora questo viene evidenziato come warning.	UC 1.1.4.5



3.4. TECNOLOGIE E STRUMENTI

R0F1.4.5.2	Il tempo è mostrato nella stessa unità di misura in cui è stato inserito il tempo di warning.	UC 1.1.4.5
R0F1.4.6	Il programma permette di aprire il programma per esplorare il file system di default del sistema e visualizzare la cartella dove è presente il file	UC 1.1.4.6

tabella 3.1: Requisiti funzionali di Multiplatform File Analyzer

3.3.2 Requisiti di qualità

Requisito	Descrizione	Fonti
R0Q1	Viene fornito insieme al programma un l'help con la guida alla compilazione e al deploy per sistemi Windows.	Tutor interno
R0Q2	Viene fornito un l'help che spiega come utilizzare il programma.	Tutor interno

tabella 3.2: Requisiti di qualità di Multiplatform File Analyzer

3.3.3 Requisiti di vincolo

Requisito	Descrizione	Fonti
R0V1	Il programma deve funzionare su Windows 7 SP1 32 e 64 bit.	Tutor interno
R0V2	Il programma deve essere scritto utilizzando il linguaggio C++ 98.	Tutor interno

tabella 3.3: Requisiti di vincolo di Multiplatform File Analyzer

3.4 Tecnologie e strumenti

Di seguito viene fornita una panoramica delle tecnologie e strumenti utilizzati per lo sviluppo di Multiplatform File Analyzer.

3.4.1 Qt® Framework

Per lo sviluppo del tool è stato usato Qt® Framework versione 5.3.1. Qt® è un potente Framework multipiattaforma che offre una vastissima serie di librerie di utilità,



3.5. CICLO DI VITA DEL SOFTWARE

grazie alle quali lo sviluppo diviene agevole e veloce, permettendo quasi sempre di non legarsi a specifiche implementazioni per piattaforma.

In particolare è stata usata la libreria per costruire le interfacce grafiche e la libreria per accedere al file system e leggerne le informazioni.

3.4.2 Qt® Creator

Come IDE per lo sviluppo ci si è affidati a Qt® Creator 3.1.2. IDE semplice e performante che si integra perfettamente con il mondo Qt®, offrendo addirittura la consultazione veloce della documentazione direttamente nell'IDE.

3.5 Ciclo di vita del software

Essendo Multiplatform File Analyzer uno strumento di piccole dimensioni si è adottato un semplice modello di ciclo di vita incrementale, con un singolo incremento corrispondente alla consegna finale.

3.6 Progettazione

Il software è stato progettato per offrire una semplice espandibilità, allo scopo la strutturazione generale segue il collaudato Design Pattern *MVC*, dove trovano luogo i seguenti pacchetti: **controller**, **view** e **data**. È inoltre presente il pacchetto **core** in cui è inserita l'implementazione degli algoritmi di business del tool, ad esempio l'algoritmo che esegue l'analisi.

Il pacchetto **core** è implementato con il Design Pattern *Strategy* allo scopo di permettere una facile estensione o sostituzione dell'algoritmo che esegue l'analisi, anche a run-time.

Per permettere una facile e veloce connessione tra gli strati descritti dall'*MVC* è stato adottato il Design Pattern *Observer*, implementato nell'omonimo pacchetto *observer*.

È stato scelto di utilizzare il Design Pattern *Strategy* per rendere modificabile facilmente l'implementazione dell'algoritmo che effettua l'analisi, all'interno del pacchetto **core**.

È stato inoltre cercato di utilizzare le classi del Framework Qt® in modo isolato alle sole parti strettamente necessarie, ovvero la GUI e l'analisi del file system. Questo allo scopo di rendere il tool slegato dalle implementazioni specifiche, in favore di una più semplice espandibilità.

3.6.1 Diagramma delle classi

Di seguito è presente il diagramma delle classi espresso tramite il formalismo UML 2.0. Nel diagramma è inoltre possibile vedere le principali dipendenze verso le classi del Framework Qt®.



3.6. PROGETTAZIONE

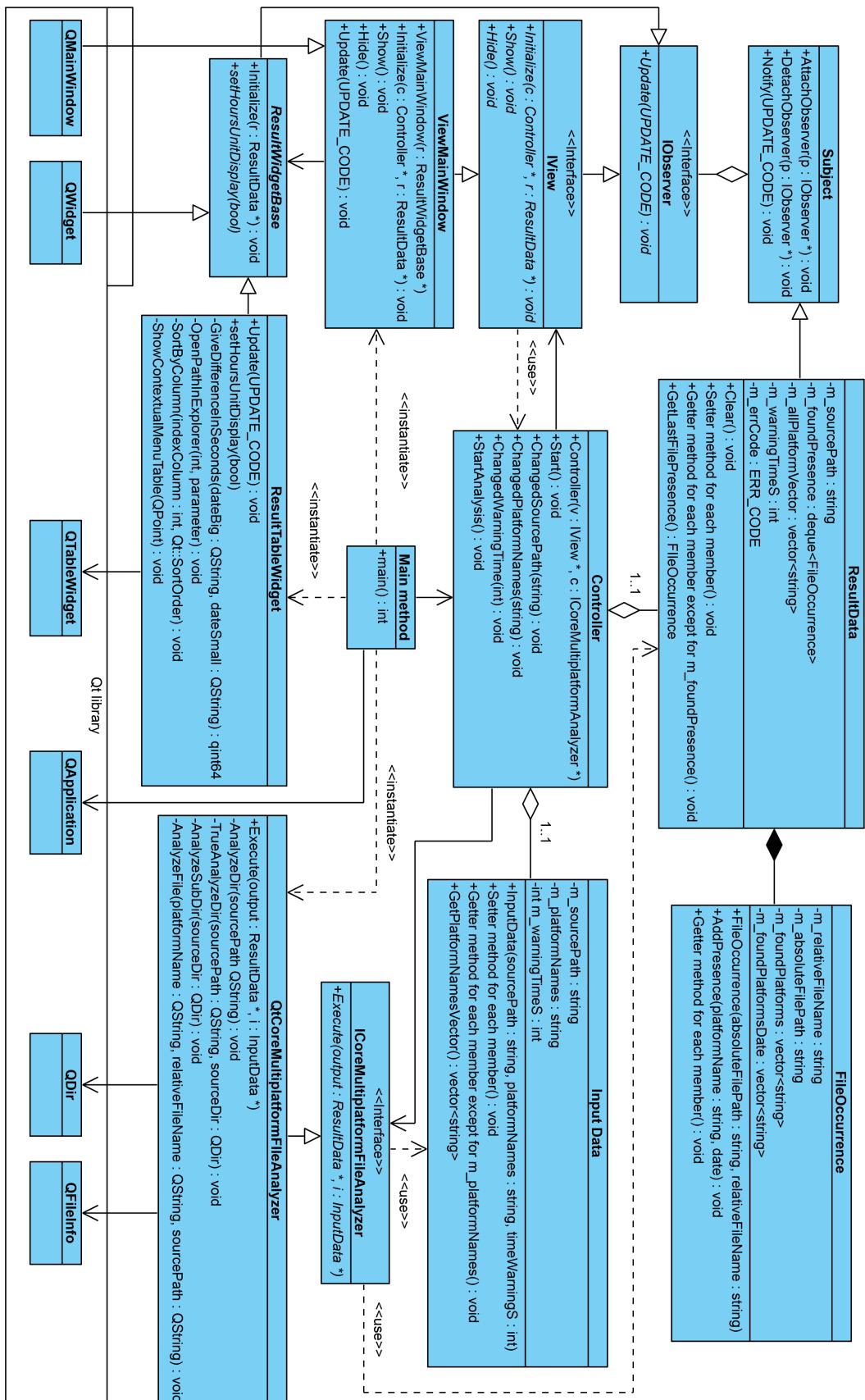


figura 3.5: Diagramma delle classi di Multiplatform File Analyzer



3.6. PROGETTAZIONE

3.6.2 Pacchetto controller

Il pacchetto è composto dalla sola classe `Controller` che svolge il compito di controllore del sistema. È questo il solo pacchetto utilizzato dalla funzione `main`.

Classe Controller

La classe `Controller` si occupa di avviare il sistema. Essa provvede ad allocare tutti gli oggetti necessari e al relativo rilascio al termine del programma. Inoltre seleziona la view e la connette con le classi che rappresentano i dati. Infine raccoglie i comandi utente provenienti dalla view e li esegue grazie al pacchetto `core`.

3.6.3 Pacchetto observer

Questo pacchetto rappresenta l'implementazione del Design Pattern *Observer* ed è stato utilizzato per tenere aggiornata la view al cambiamento dei dati.

Interfaccia IObserver

`IObserver` è l'interfaccia che rappresenta gli oggetti che osservano le modifiche di altri oggetti. Essa contiene il metodo virtuale puro `Update` che i soggetti osservati invocano per segnalare il fatto che sono stati modificati e che quindi un aggiornamento è necessario. Le classi concrete che avranno la necessità di osservare un soggetto deriveranno da `IObserver` implementando il metodo `Update`.

Classe Subject

La classe `Subject` rappresenta un soggetto che può essere osservato. Essa contiene il codice necessario per notificare tutti gli osservatori dell'avvenuto cambiamento. Mantiene inoltre una lista degli osservatori da notificare, i quali possono iscriversi se desiderano ricevere la notifica, oppure rimuoversi se non è più necessario ricevere aggiornamenti. Le classi che devono essere osservate deriveranno da questa.

3.6.4 Pacchetto data

Il pacchetto `data` contiene tutte le classi che rappresentano i dati di business del programma. Allo scopo di rendere il programma eseguibile anche su un semplice terminale e quindi disaccoppiato dal mondo Qt®, le classi di dati sono state implementate con la Standard Template Library (STL), piuttosto che le classi collezione offerte dal framework in uso.

Classe InputData

La classe `InputData` raccoglie tutti i dati che sono di input all'analisi. Contiene quindi il percorso di partenza, i nomi delle piattaforme ed il tempo di warning. È questa classe che si occupa della trasformazione della stringa contenente tutte le piattaforme in un `Vector`.

Classe FileOccurrence

La classe `FileOccurrence` rappresenta un file trovato in almeno una cartella di piattaforma durante l'analisi. Per il file trovato, la classe contiene il percorso assoluto, il nome, tutte le piattaforme in cui è stato trovato e, per ciascuna, la data di ultima modifica.



Classe ResultData

La classe `ResultData` contiene tutti i dati presenti come output di una analisi. Contiene quindi una collezione di `FileOccurrence` e tutti i dati di input all'analisi. Essa deriva dalla classe `Subject` per permettere ad un osservatore, tipicamente un elemento della View, di aggiornarsi al variare dei risultati. Permettendo di mostrare i risultati dell'analisi in modo interattivo durante l'inserimento di ogni nuovo `FileOccurrence` trovato.

3.6.5 Pacchetto view

Questo pacchetto si occupa delle interfacce utente, derivando e personalizzando le classi offerte dal Framework Qt®.

Interfaccia IView

`IView` è una classe astratta che rappresenta una generica view.

Classe ViewMainWindow

La classe `ViewMainWindow` implementa `IView` tramite una finestra. Eredita e specializza la classe `QMainWindow` di Qt®. Raccoglie i comandi utente e ne delega l'esecuzione al controller.

Classe ResultWidgetBase

La classe astratta `ResultWidgetBase` rappresenta un generico widget integrabile in una interfaccia grafica che sfrutta le classi del Framework Qt® per la visualizzazione dell'output dell'analisi.

Classe ResultTableWidget

La classe `ResultTableWidget` implementa la classe base astratta `ResultWidgetBase` mostrando l'output dell'analisi in forma tabellare, dedicando una riga a ciascun file trovato.

3.6.6 Pacchetto core

Il pacchetto si occupa della realizzazione dell'analisi e di tutti gli algoritmi di business del tool. Utilizza il Design Pattern *Strategy* per l'organizzazione delle classi contenute.

Interfaccia ICoreMultiplatformFileAnalyzer

`ICoreMultiplatformFileAnalyzer` rappresenta l'algoritmo di business del programma, ovvero quello che effettua l'analisi e riempie un'istanza della classe `ResultData` con il risultato.

Classe QtCoreMultiplatformFileAnalyzer

La classe `QtCoreMultiplatformFileAnalyzer` implementa l'interfaccia `ICoreMultiplatformFileAnalyzer` utilizzando il Framework Qt® per esplorare il file system.

3.7 Codifica

Tutto il codice del tool è stato pubblicato sulla piattaforma GitHub®, accedibile tramite il seguente indirizzo: <https://github.com/Mauxx91/Multiplatform-File-A>



3.8. USO PRATICO

nalyzer. Tutto il codice è disponibile gratuitamente sotto licenza GNU GPL v3³

Tutto il codice è stato scritto in lingua inglese, compresi i commenti, dei quali si è cercato di scriverne il più possibile. La codifica ha seguito alcune convenzioni della famosa notazione Ungherese. Di seguito sono elencati i formalismi utilizzati nel codice:

- **prefissi:**

- **m**: usato per le variabili membro (esempio: `m_keyName`);
- **p**: usato per le variabili puntatore (esempio: `m_pkeyNameList`);
- **o**: usato per le variabili input alla funzione che sono usate come output (esempio: `o_foundKeys`);
- **I**: usato per le classi interfacce (esempio: `IObserver`).

- **suffissi:**

- **Base**: usato per le classi astratte (esempio: `ResultWidgetBase`);

- **capitalizzazione:**

- **variabili e parametri**: iniziano sempre in minuscolo e usano una lettera minuscola per ogni parola (esempio: `m_parentWidget`, `returnValue`);
- **macro ed enum**: completamente in maiuscolo con le diverse parole separate dal carattere ‘_’ (esempio: `UPDATE_CODE`);
- **classi, funzioni e metodi**: iniziano sempre in maiuscolo e ogni parola diversa parte in maiuscolo (esempio: `FileOccurrence`);

3.8 Uso pratico

In questa sezione si vuol presentare degli screenshot di uso pratico di Multiplatform File Analyzer.

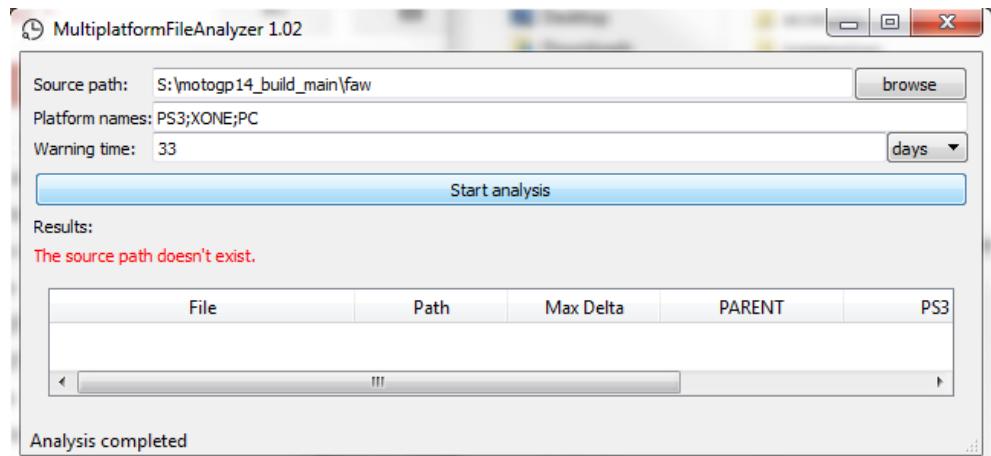


figura 3.6: Screenshot di Multiplatform File Analyzer nel quale si può osservare la segnalazione di errori sui dati in input all’analisi

³Un approfondimento sulla licenza può essere trovato al seguente indirizzo: <http://www.gnu.org/copyleft/gpl.html>.



3.9. CONCLUSIONI

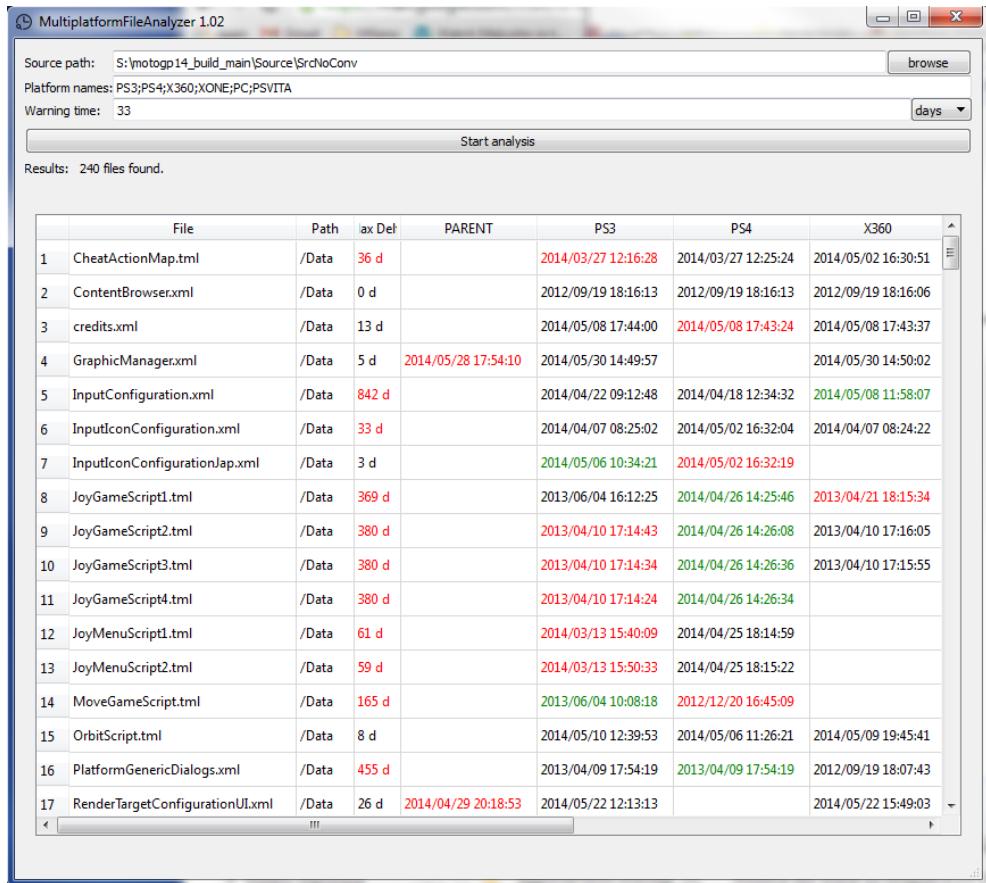


figura 3.7: Screenshot di Multiplatform File Analyzer nel quale si può osservare la presentazione dei risultati dell'analisi

3.9 Conclusioni

Tutti i requisiti sono stati soddisfatti come pianificato e nei tempi prestabiliti. La realizzazione del tool ha messo di fronte lo studente a una problematica reale presente durante lo sviluppo di giochi multiplattaforma. Contemporaneamente gli ha permesso di interfacciarsi ai comuni problemi legati alla manutenzione dei moltissimi file presenti nei grandi progetti, lasciandogli la libertà di progettare e realizzare una soluzione.

Lo studente ha inoltre approfondito le tematiche della creazione di interfacce usabili e degli algoritmi per l'esplorazione del file system.

Capitolo 4

XML Editor

In questo capitolo è presente una analisi completa del tool XML Editor

4.1 Introduzione

4.1.1 Premessa

L'engine di un videogioco, se sviluppato secondo una buona architettura, legge da file di configurazione la maggior parte delle informazioni piuttosto di averle scritte direttamente nel codice¹. I file di configurazione possono essere poi specializzati per una specifica piattaforma come già visto nel Multiplatform File Analyzer. Il contenuto di questi file può essere estremamente vario, ad esempio sono utilizzati per indicare i percorsi delle pagine dei menu, la descrizione degli oggetti grafici in relazione alle sotto-componenti e alle animazioni. Gli standard prevedono che tali file di configurazione vengano scritti mediante il linguaggio XML.

4.1.2 Lo scopo

Vista l'eterogeneità dei file di configurazione ed del team di sviluppo di un gioco, è possibile che questi siano scritti anche da persone non specializzate (esempio: artisti). Inoltre, è possibile inserire relazioni tra elementi presenti in questi file. Ad esempio, per un oggetto grafico si specifica quali sono gli oggetti attaccati allo scheletro di animazione, elementi che sono definiti solitamente in un altro file di configurazione. Essendo questi caricati a run-time, il debugging diviene decisamente non banale, in quanto non è semplice capire da dove l'errore proviene essendo moltissime le variabili in gioco.

Il tool si pone l'obiettivo di rendere la scrittura e il debugging dei file di configurazione XML più semplice e veloce. Innanzitutto fornirà un editor visuale in cui sarà possibile editare in tutti gli aspetti l'XML. Verrà inoltre fornita la possibilità di specificare le relazioni ed eventuali altri file coinvolti, dopodiché il programma sarà in grado di verificare le relazioni nei file aperti e di crearne di nuove tramite un semplice drag & drop dei nodi destinatari della relazione.

4.2 Casi d'uso

Per meglio capire e tracciare l'esperienza d'uso che un developer di un gioco avrà con il tool sono stati creati dei diagrammi di casi d'uso gerarchici.

¹La pratica di scrivere nel gioco le informazioni è comunemente chiamata *Hard-coding*.



4.2. CASI D'USO

I diagrammi di casi d'uso fanno parte della famiglia dei diagrammi UML e descrivono le funzionalità offerte dal prodotto così come sono percepite dagli attori che interagiscono con il sistema.

Ogni caso d'uso ha un codice univoco gerarchico, nella forma:

UC[codice univoco del padre].[codice progressivo di livello]

Il codice progressivo può includere diversi livelli di gerarchia separati da un punto.

4.2.1 UC 1: Caso d'uso principale

- **Attori:** developer.
- **Descrizione:** un developer deve avere tutte le funzionalità a disposizione per editare ed verificare uno o più file XML.
- **Precondizione:** il developer abbia lanciato il tool sotto un sistema Windows 7 o superiore.
- **Flusso principale degli eventi:**
 1. *l'utente ha la possibilità di: creazione di un nuovo file XML (UC 1.1);*
 2. *l'utente ha la possibilità di: apertura di un file XML preesistente (UC 1.2);*
 3. *l'utente ha la possibilità di: editing del file XML correntemente aperto (UC 1.3);*
 4. *l'utente ha la possibilità di: salvare le modifiche di un file modificato (UC 1.4);*
 5. *l'utente ha la possibilità di: editing del filtro sul nome dell'attributo da visualizzare (UC 1.5);*
 6. *l'utente ha la possibilità di: editing dei file associati (UC 1.6);*
 7. *l'utente ha la possibilità di: editing delle relazioni (UC 1.7);*
 8. *l'utente ha la possibilità di: importazione delle relazioni da file (UC 1.8);*
 9. *l'utente ha la possibilità di: esportazione delle relazioni su file (UC 1.9);*
 10. *l'utente ha la possibilità di: avvio del check sulle relazioni (UC 1.10);*
 11. *l'utente ha la possibilità di: visualizzazione del risultato del check sulle relazioni (UC 1.11);*
 12. *l'utente ha la possibilità di: visualizzazione dell'about del programma (UC 1.12).*
- **Estensioni**
 1. *Visualizzazione degli errori occorsi durante l'apertura del file (UC 1.13);*
 2. *Visualizzazione degli errori occorsi durante il salvataggio del file (UC 1.14);*
 3. *Visualizzazione degli errori occorsi durante l'importazione (UC 1.15);*
 4. *Visualizzazione degli errori occorsi durante l'esportazione (UC 1.16).*
- **Postcondizione:** il sistema ha erogato le funzionalità richieste dal developer.



4.2. CASI D'USO

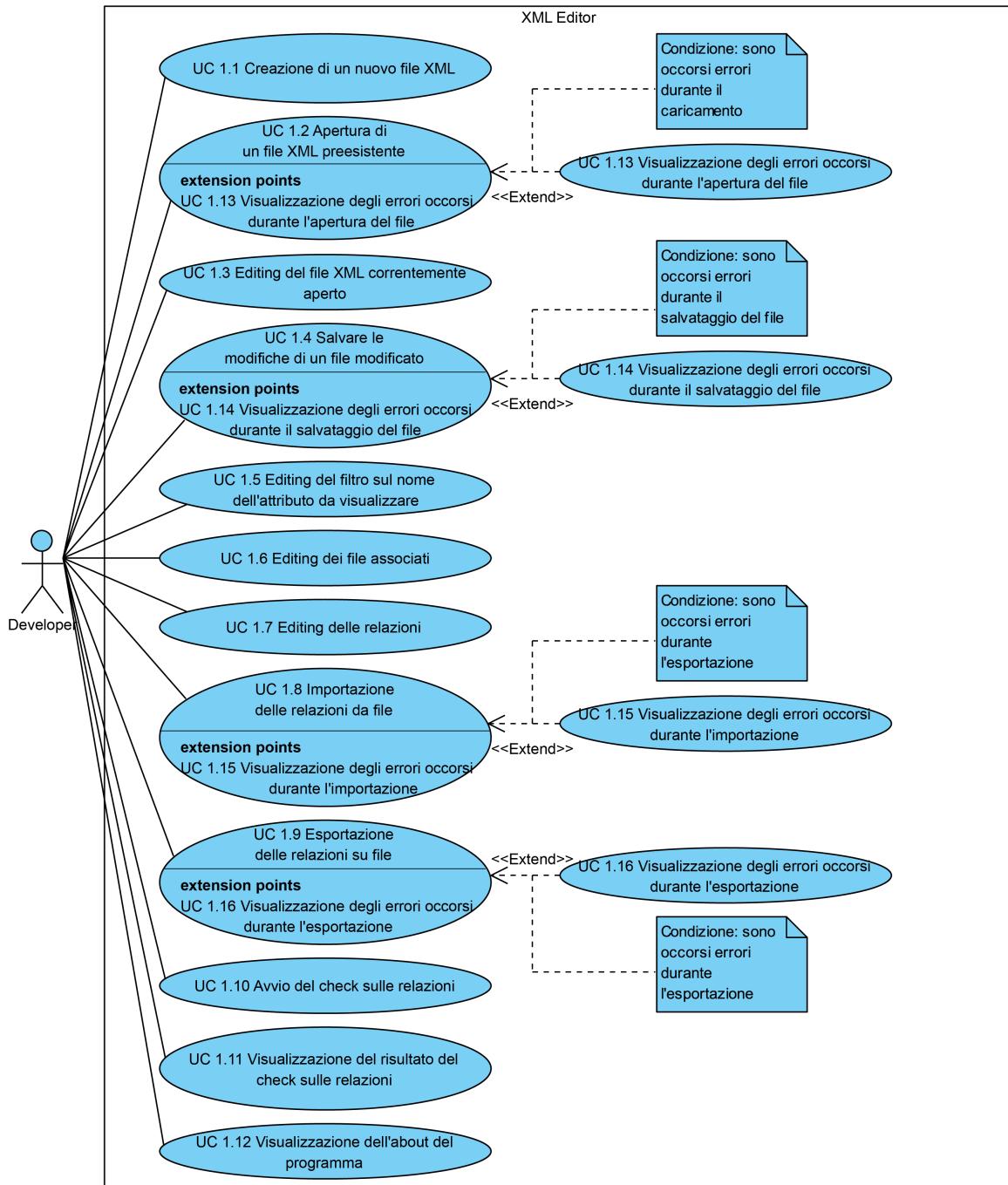


figura 4.1: Use Case - UC 1: Caso d'uso principale

4.2.2 UC 1.1: Creazione di un nuovo file XML

- **Attori:** developer.
- **Descrizione:** un developer deve poter creare un nuovo file XML.
- **Precondizione:** il developer ha lanciato il tool sotto un sistema Windows 7 o superiore.
- **Flusso principale degli eventi:**



4.2. CASI D'USO

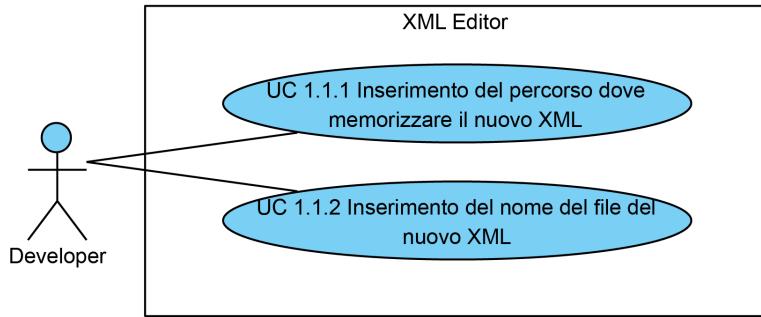


figura 4.2: Use Case - UC 1.1: Creazione di un nuovo file XML

1. *Inserimento del percorso dove memorizzare il nuovo XML* (UC 1.1.1);
2. *Inserimento del nome del file del nuovo XML* (UC 1.1.2).

- **Postcondizione:** il sistema ha creato un nuovo file XML in memoria, esso sarà salvato su disco solo nel momento in cui l'utente lo salverà. Il sistema chiude il lavoro corrente e apre il nuovo file creato.

4.2.3 UC 1.1.1: Inserimento del percorso dove memorizzare il nuovo XML

- **Attori:** developer.
- **Descrizione:** un developer deve poter inserire il percorso dove memorizzare su disco il nuovo file XML.
- **Precondizione:** il developer abbia selezionato l'azione per creare un nuovo file XML.
- **Scenario principale:** il developer sceglie il percorso dove memorizzare il nuovo file XML.
- **Postcondizione:** il sistema ha memorizzato il percorso dove salvare il nuovo file XML.

4.2.4 UC 1.1.2: Inserimento del nome del file del nuovo XML

- **Attori:** developer.
- **Descrizione:** un developer deve poter inserire il nome del nuovo file XML.
- **Precondizione:** il developer abbia selezionato l'azione per creare un nuovo file XML.
- **Scenario principale:** il developer inserisce il nome del nuovo file XML.
- **Postcondizione:** il sistema ha memorizzato il nome del nuovo file XML.

4.2.5 UC 1.2: Apertura di un file XML preesistente

- **Attori:** developer.
- **Descrizione:** un developer deve poter aprire un file XML preesistente.
- **Precondizione:** il developer abbia selezionato l'azione per aprire un file XML.



4.2. CASI D'USO

- **Scenario principale:** il developer inserisce il percorso ed il nome del file XML da aprire.
- **Postcondizione:** il sistema ha chiuso il lavoro precedentemente aperto e l'ha sostituito con il file XML selezionato. Se il file era stato precedentemente aperto ed erano stati settati alcuni file associati allora anche quei file saranno aperti.

4.2.6 UC 1.3 Editing del file XML correntemente aperto

- **Attori:** developer.
- **Descrizione:** un developer deve poter editare il file XML correntemente visualizzato.
- **Precondizione:** il developer abbia aperto con successo almeno un file XML.
- **Flusso principale degli eventi:**
 1. *l'utente ha la possibilità di: visualizzazione del nodo riferito dal nodo selezionato (UC 1.3.1);*
 2. *l'utente ha la possibilità di: aggiunta di un nuovo nodo figlio al nodo selezionato (UC 1.3.3);*
 3. *l'utente ha la possibilità di: editing del nodo selezionato (UC 1.3.4);*
 4. *l'utente ha la possibilità di: duplicazione del nodo selezionato (UC 1.3.5);*
 5. *l'utente ha la possibilità di: copia del nodo selezionato (UC 1.3.6);*
 6. *l'utente ha la possibilità di: incollare il nodo precedentemente copiato (UC 1.3.7);*
 7. *l'utente ha la possibilità di: rimozione del nodo selezionato (UC 1.3.8);*
 8. *l'utente ha la possibilità di: rimozione di tutti i nodi figli del nodo selezionato (UC 1.3.9);*
 9. *l'utente ha la possibilità di: istanziazione di una nuova relazione (UC 1.3.10);*
 10. *l'utente ha la possibilità di: annullamento dell'ultima modifica eseguita (UC 1.3.12);*
 11. *l'utente ha la possibilità di: ripristino dell'ultima modifica annullata (UC 1.3.13).*
- **Estensioni**
 1. *Visualizzazione degli errori occorsi durante la ricerca del nodo riferito (UC 1.3.2);*
 2. *Visualizzazione degli errori occorsi durante l'istanziazione di una nuova relazione (UC 1.3.11).*
- **Postcondizione:** il sistema ha modificato la versione caricata in memoria del file XML modificato.

4.2. CASI D'USO

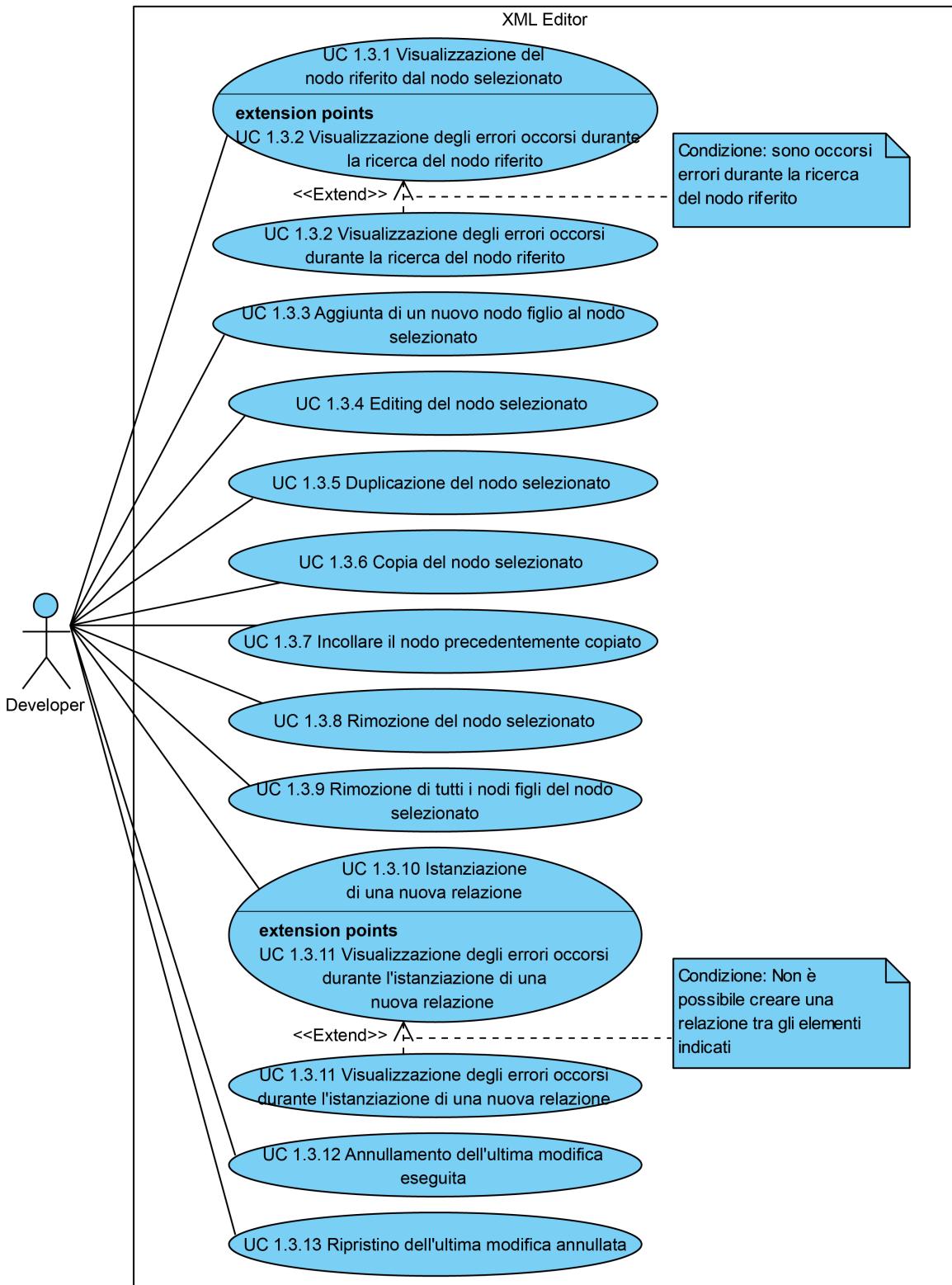


figura 4.3: Use Case - UC 1.3 Editing del file XML correntemente aperto



4.2.7 UC 1.3.1 Visualizzazione del nodo riferito dal nodo selezionato

- **Attori:** developer.
- **Descrizione:** un developer visualizza il nodo destinatario della relazione che ha il nodo selezionato come punto di partenza.
- **Precondizione:** il developer abbia selezionato l'azione per seguire la relazione.
- **Scenario principale:** il developer visualizza il nodo destinatario della relazione.
- **Postcondizione:** il sistema ha selezionato ed espanso l'albero fino al nodo destinatario della relazione. Se sono presenti più nodi destinazione o più relazioni da seguire viene sempre usata quella inserita precedentemente nel sistema.

4.2.8 UC 1.3.2 Visualizzazione degli errori occorsi durante la ricerca del nodo riferito

- **Attori:** developer.
- **Descrizione:** si è verificato uno dei seguenti errori durante la ricerca del nodo destinazione:
 - non esiste nessuna relazione per cui il nodo selezionato è riconoscibile come nodo partenza di una relazione;
 - non esiste nessun nodo di destinazione.
- **Precondizione:** il developer ha avviato l'azione per mostrare il nodo destinatario della relazione.
- **Scenario principale:** viene fornita una spiegazione dell'errore commesso e come risolverlo.
- **Postcondizione:** l'errore commesso è stato visualizzato e spiegato, non viene visualizzato nessun nodo destinazione.

4.2.9 UC 1.3.3 Aggiunta di un nuovo nodo figlio al nodo selezionato

- **Attori:** developer.
- **Descrizione:** viene aggiunto un nuovo nodo vuoto come figlio del nodo correttamente selezionato.
- **Precondizione:** il developer abbia selezionato l'azione aggiungere un nuovo nodo come figlio del nodo correttamente visualizzato.
- **Scenario principale:** il developer inserisce il tag name del nuovo nodo.
- **Postcondizione:** il sistema ha selezionato il nodo ed espanso l'albero fino al nuovo nodo creato.

4.2.10 UC 1.3.4 Editing del nodo selezionato

- **Attori:** developer.
- **Descrizione:** un developer deve poter editare il file XML correntemente visualizzato.

4.2. CASI D'USO

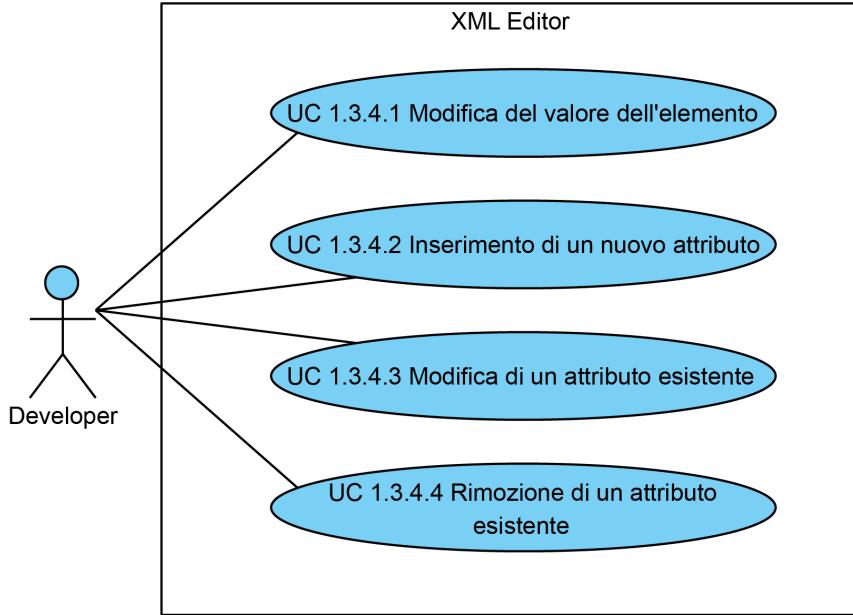


figura 4.4: Use Case - UC 1.3.4 Editing del nodo selezionato

- **Precondizione:** il developer abbia selezionato l'azione per modificare l'elemento correntemente selezionato.
- **Flusso principale degli eventi:**
 1. *l'utente ha la possibilità di: modifica del valore dell'elemento (UC 1.3.4.1);*
 2. *l'utente ha la possibilità di: inserimento di un nuovo attributo (UC 1.3.4.2);*
 3. *l'utente ha la possibilità di: modifica di un attributo esistente (UC 1.3.4.3);*
 4. *l'utente ha la possibilità di: rimozione di un attributo esistente (UC 1.3.4).*
- **Postcondizione:** il sistema ha modificato la versione caricata in memoria del file XML modificato.

4.2.11 UC 1.3.4.1 Modifica del valore dell'elemento

- **Attori:** developer.
- **Descrizione:** il developer può modificare il valore dell'elemento.
- **Precondizione:** il developer abbia selezionato l'azione aggiungere per modificare un elemento.
- **Scenario principale:** il developer inserisce il nuovo value.
- **Postcondizione:** il sistema ha modificato il valore dell'elemento.

4.2.12 UC 1.3.4.2 Inserimento di un nuovo attributo

- **Attori:** developer.
- **Descrizione:** il developer può aggiungere un attributo all'elemento.
- **Precondizione:** il developer abbia selezionato l'azione aggiungere per modificare un elemento.



- **Scenario principale:** il developer inserisce il nome del nuovo attributo ed il corrispettivo valore.
- **Postcondizione:** il sistema ha aggiunto un nuovo attributo con il nome ed il valore inseriti dal developer.

4.2.13 UC 1.3.4.3 Modifica di un attributo esistente

- **Attori:** developer.
- **Descrizione:** il developer può modificare il valore e/o il nome di un attributo precedentemente inserito.
- **Precondizione:** il developer abbia selezionato l'azione aggiungere per modificare un elemento.
- **Scenario principale:** il developer inserisce il nome e/o il valore dell'attributo da modificare.
- **Postcondizione:** il sistema ha modificato l'attributo selezionato con i dati inseriti dal developer.

4.2.14 UC 1.3.4.4 Rimozione di un attributo esistente

- **Attori:** developer.
- **Descrizione:** il developer può eliminare un attributo precedentemente inserito.
- **Precondizione:** il developer abbia selezionato l'azione aggiungere per modificare un elemento.
- **Scenario principale:** il developer seleziona l'attributo da eliminare.
- **Postcondizione:** il sistema ha eliminato l'attributo selezionato dal developer.

4.2.15 UC 1.3.5 Duplicazione del nodo selezionato

- **Attori:** developer.
- **Descrizione:** viene aggiunta al padre del nodo selezionato una copia profonda di questo.
- **Precondizione:** il developer abbia selezionato l'azione di duplicazione di un nodo.
- **Scenario principale:** il developer seleziona il nodo da duplicare.
- **Postcondizione:** il sistema ha aggiunto al padre del nodo selezionato una copia profonda di tale nodo. Inoltre ha espanso l'albero fino al nuovo nodo creato.

4.2.16 UC 1.3.6 Copia del nodo selezionato

- **Attori:** developer.
- **Descrizione:** viene fatta una copia profonda del nodo selezionato e conservata in memoria, pronta per essere incollata.
- **Precondizione:** il developer abbia selezionato l'azione di copia di un nodo.
- **Scenario principale:** il developer seleziona il nodo che intende copiare.
- **Postcondizione:** il sistema ha memorizzato in memoria una copia profonda del nodo selezionato.



4.2. CASI D'USO

4.2.17 UC 1.3.7 Incollare il nodo precedentemente copiato

- **Attori:** developer.
- **Descrizione:** viene aggiunto come figlio del nodo correttamente selezionato il nodo precedentemente copiato.
- **Precondizione:** il developer richiede di copiare il nodo precedentemente copiato.
- **Scenario principale:** il developer seleziona il nodo destinazione che conterrà il nuovo nodo.
- **Postcondizione:** il sistema ha aggiunto come figlio del nodo destinazione il nodo precedentemente copiato. Inoltre ha selezionato ed espanso l'albero fino al nuovo nodo aggiunto.

4.2.18 UC 1.3.8 Rimozione del nodo selezionato

- **Attori:** developer.
- **Descrizione:** viene rimosso il nodo correntemente selezionato.
- **Precondizione:** il developer abbia selezionato l'azione di rimozione di un nodo.
- **Scenario principale:** il developer abbia selezionato il nodo da rimuovere.
- **Postcondizione:** il sistema ha rimosso il nodo selezionato.

4.2.19 UC 1.3.9 Rimozione di tutti i nodi figli del nodo selezionato

- **Attori:** developer.
- **Descrizione:** vengono rimossi tutti i nodi figli del nodo correntemente selezionato.
- **Precondizione:** il developer abbia selezionato l'azione per rimuovere tutti i nodi figli.
- **Scenario principale:** il developer seleziona il nodo di cui si vuole rimuovere tutti i figli.
- **Postcondizione:** il sistema ha rimosso tutti i nodi del nodo attualmente selezionato.

4.2.20 UC 1.3.10 Istanziazione di una nuova relazione

- **Attori:** developer.
- **Descrizione:** vengono creati in automatico tutti i nodi necessari per creare una relazione tra i nodi partenza selezionati e il nodo destinatario scelto.
- **Precondizione:** il developer abbia selezionato l'azione per creare in automatico una relazione.
- **Scenario principale:** il developer seleziona uno o più nodi e successivamente sceglie il nodo che riferirà i nodi precedentemente selezionati.
- **Postcondizione:** il sistema ha creato tutti i nodi necessari ad esplicitare le relazioni richieste.



4.2.21 UC 1.3.11 Visualizzazione degli errori occorsi durante l'istanziazione di una nuova relazione

- **Attori:** developer.
- **Descrizione:** si è verificato uno dei seguenti errori durante l'istanziazione di una relazione:
 - non esiste nessuna relazione per cui il nodo selezionato è riconoscibile come nodo partenza di una relazione;
 - sono stati selezionati come nodi destinazione nodi non riconoscibili come destinazione di una relazione;
 - non esiste nessun percorso di relazioni che lega il nodo partenza con il nodo destinazione selezionati;
 - sono stati selezionati come nodi destinazione nodi di tipi differenti.
- **Precondizione:** il developer ha selezionato l'azione per creare una nuova relazione e specificato gli elementi da coinvolgere.
- **Scenario principale:** viene fornita una spiegazione dell'errore commesso e come risolverlo.
- **Postcondizione:** l'errore commesso è stato visualizzato e spiegato, non viene eseguita nessuna modifica al file XML.

4.2.22 UC 1.3.12 Annullamento dell'ultima modifica eseguita

- **Attori:** developer.
- **Descrizione:** viene annullata l'ultima modifica e ripristinato lo stato precedente del file.
- **Precondizione:** il developer abbia selezionato l'azione per annullare l'ultima modifica ed è stata effettuata almeno una modifica al file XML.
- **Scenario principale:** viene ripristinato lo stato del file precedente alla modifica.
- **Postcondizione:** il sistema ha ripristinato lo stato del file prima dell'ultima azione di modifica.

4.2.23 UC 1.3.13 Ripristino dell'ultima modifica annullata

- **Attori:** developer.
- **Descrizione:** viene ripristinata l'ultima modifica annullata.
- **Precondizione:** il developer abbia selezionato l'azione per ripristinare l'ultima modifica annullata ed è stata annullata almeno una modifica al file XML.
- **Scenario principale:** viene ripristinata l'ultima modifica annullata.
- **Postcondizione:** il sistema ha ripristinato lo stato del file prima dell'ultima azione di annullamento eseguita.



4.2. CASI D'USO

4.2.24 UC 1.4 Salvare le modifiche di un file modificato

- **Attori:** developer.
- **Descrizione:** un developer deve poter salvare le modifiche effettuate su un file XML precedentemente aperto.
- **Precondizione:** il developer abbia selezionato l'azione per salvare un file XML e abbia almeno un file aperto e modificato.
- **Scenario principale:** viene salvato il file XML modificato.
- **Postcondizione:** il sistema ha salvato su disco le modifiche apportate al file XML.

4.2.25 UC 1.5 Editing del filtro sul nome dell'attributo da visualizzare

- **Attori:** developer.
- **Descrizione:** un developer deve poter modificare il filtro che seleziona l'attributo cui valore è mostrato nell'albero rappresentante il file XML.
- **Precondizione:** il developer abbia lanciato il tool sotto un sistema Windows 7 o superiore.
- **Scenario principale:** il developer inserisce il nuovo filtro.
- **Postcondizione:** il sistema ha salvato il nuovo filtro e ha aggiornato la visualizzazione di tutti i file aperti.

4.2.26 UC 1.6 Editing dei file associati

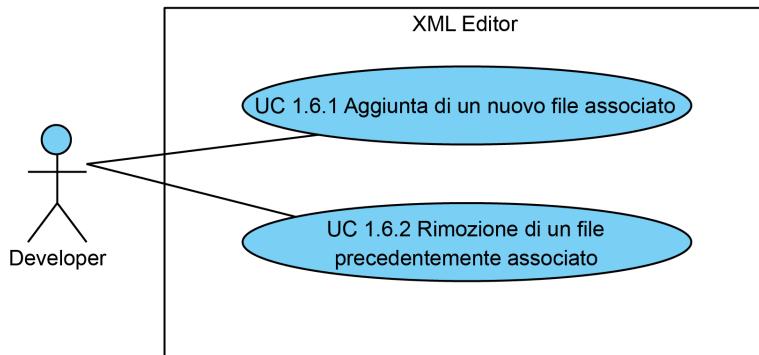


figura 4.5: Use Case - UC 1.6 Editing dei file associati

- **Attori:** developer.
- **Descrizione:** un developer deve poter aggiungere e rimuovere i file associati.
- **Precondizione:** il developer ha caricato correttamente un file XML.
- **Flusso principale degli eventi:**
 1. *l'utente ha la possibilità di: aggiunta di un nuovo file associato (UC 1.6.1);*
 2. *l'utente ha la possibilità di: rimozione di un file precedentemente associato (UC1.6.2).*
- **Postcondizione:** il sistema ha modificato i file associati del file XML principale attualmente aperto.



4.2.27 UC 1.6.1 Aggiunta di un nuovo file associato

- **Attori:** developer.
- **Descrizione:** un developer deve poter aggiungere un nuovo file associato al corrente file principale.
- **Precondizione:** il developer abbia selezionato l'azione di modifica dei file associati.
- **Scenario principale:** il developer inserisce il percorso ed il nome del nuovo file associato.
- **Postcondizione:** il sistema ha salvato il nuovo file associato lo carica.

4.2.28 UC 1.6.2 Rimozione di un file precedentemente associato

- **Attori:** developer.
- **Descrizione:** un developer deve poter rimuovere un file associato dal corrente file principale.
- **Precondizione:** il developer abbia selezionato l'azione di modifica dei file associati.
- **Scenario principale:** il developer seleziona il file da rimuovere.
- **Postcondizione:** il sistema ha rimosso il file selezionato. La copia in memoria del file non è stata eliminata.

4.2.29 UC 1.7 Editing delle relazioni

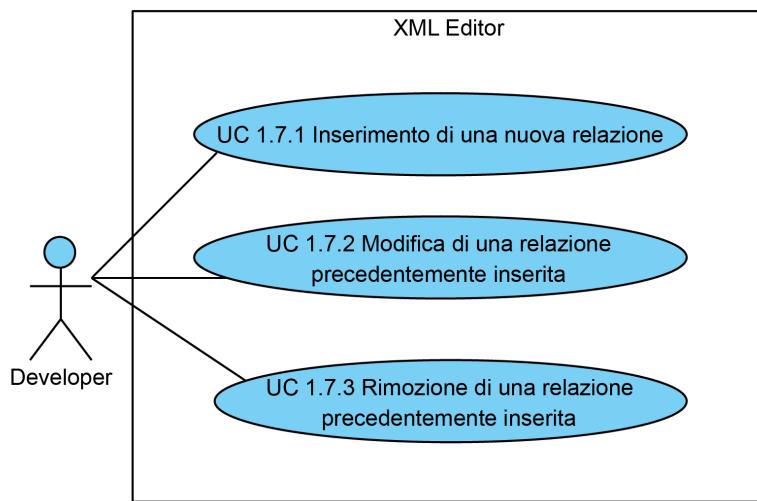


figura 4.6: Use Case - UC 1.7 Editing delle relazioni

- **Attori:** developer.
- **Descrizione:** un developer deve poter editare le relazioni presenti.
- **Precondizione:** il developer ha lanciato il tool sotto un sistema Windows 7 o superiore.



4.2. CASI D'USO

- **Flusso principale degli eventi:**
 1. *l'utente ha la possibilità di: inserimento di una nuova relazione (UC 1.7.1);*
 2. *l'utente ha la possibilità di: modifica di una relazione precedentemente inserita (UC1.7.2);*
 3. *l'utente ha la possibilità di: rimozione di una relazione precedentemente inserita (UC1.7.3).*
- **Postcondizione:** il sistema ha modificato le relazione come desiderato dall'utente.

4.2.30 UC 1.7.1 Inserimento di una nuova relazione

- **Attori:** developer.
- **Descrizione:** un developer deve poter aggiungere una nuova relazione.
- **Precondizione:** il developer abbia selezionato l'azione di modifica delle relazioni.
- **Scenario principale:** il developer inserisce tutte le informazioni necessarie alla creazione di una relazione.
- **Postcondizione:** il sistema ha memorizzato la nuova relazione.

4.2.31 UC 1.7.2 Modifica di una relazione precedentemente inserita

- **Attori:** developer.
- **Descrizione:** un developer deve poter modificare una relazione precedentemente inserita.
- **Precondizione:** il developer abbia selezionato l'azione di modifica delle relazioni e la relazione da modificare.
- **Scenario principale:** il developer inserisce tutte le informazioni che vuole modificare della relazione.
- **Postcondizione:** il sistema ha memorizzato i cambiamenti nella relazione selezionata.

4.2.32 UC 1.7.3 Rimozione di una relazione precedentemente inserita

- **Attori:** developer.
- **Descrizione:** un developer deve poter rimuovere una relazione.
- **Precondizione:** il developer abbia selezionato l'azione di modifica delle relazioni.
- **Scenario principale:** il developer seleziona la relazione da rimuovere.
- **Postcondizione:** il sistema ha rimosso la relazione selezionata.



4.2.33 UC 1.8 Importazione delle relazioni da file

- **Attori:** developer.
- **Descrizione:** un developer deve poter importare e sostituire le relazioni correnti con le relazione contenute in un file di configurazione precedentemente esportato.
- **Precondizione:** il developer ha selezionato l'azione per importare le relazioni da un file.
- **Scenario principale:** il developer inserisce il percorso ed il nome del file contenente le relazioni.
- **Postcondizione:** il sistema ha rimosso tutte le precedenti relazioni e le ha sostituite con tutte quelle presenti nel file.

4.2.34 UC 1.9 Esportazione delle relazioni su file

- **Attori:** developer.
- **Descrizione:** un developer deve poter esportare le relazioni attualmente inserite su file.
- **Precondizione:** il developer ha selezionato l'azione per esportare le relazioni su file.
- **Scenario principale:** il developer inserisce il percorso ed il nome del file dove esportare le relazioni.
- **Postcondizione:** il sistema esportato le relazioni nel file selezionato.

4.2.35 UC 1.10 Avvio del check sulle relazioni

- **Attori:** developer.
- **Descrizione:** un developer deve poter verificare la consistenza delle relazioni presenti nel file principale e in tutti i file associati sulla base delle relazioni inserite.
- **Precondizione:** il developer ha selezionato l'azione per verificare la consistenza delle relazioni. Almeno un file XML deve essere stato aperto con successo.
- **Scenario principale:** il developer seleziona l'azione per effettuare il check delle relazioni.
- **Postcondizione:** il sistema ha verificato le relazioni nel file principale e in tutti gli associati.

4.2.36 UC 1.11 Visualizzazione del risultato del check sulle relazioni

- **Attori:** developer.
- **Descrizione:** un developer deve visualizzare l'esito del check sulle relazioni.
- **Precondizione:** il developer ha avviato il check sulle relazioni e quest'ultimo è terminato.
- **Flusso principale degli eventi:**
 1. *l'utente ha la possibilità di: visualizzazione degli errori (UC 1.11.1);*

4.2. CASI D'USO

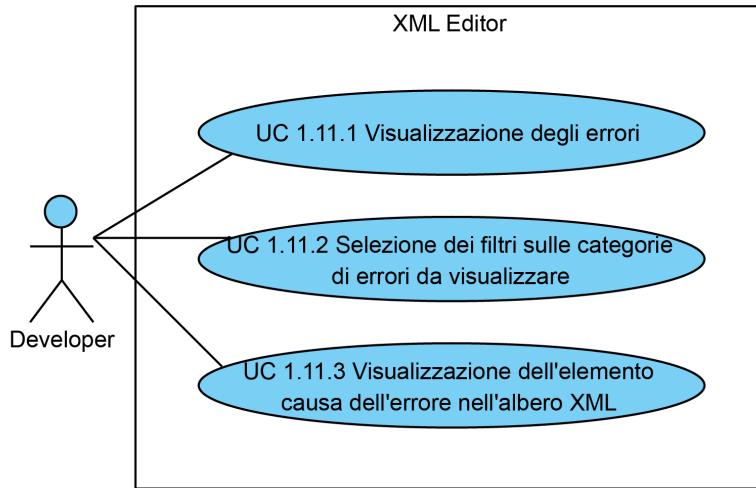


figura 4.7: Use Case - UC 1.11 Visualizzazione del risultato del check sulle relazioni

2. l'utente ha la possibilità di: selezione dei filtri sulle categorie di errori da visualizzare (UC1.11.2);
3. l'utente ha la possibilità di: visualizzazione dell'elemento causa dell'errore nell'albero XML (UC1.11.3).

- **Postcondizione:** il sistema ha permesso la visualizzazione dei risultati del check sulle relazioni.

4.2.37 UC 1.11.1 Visualizzazione degli errori

- **Attori:** developer.
- **Descrizione:** un developer deve poter l'esito della verifica delle relazioni.
- **Precondizione:** il developer ha avviato il check sulle relazioni e quest'ultimo è terminato.
- **Scenario principale:** viene visualizzato per ogni errore: la severità, la tipologia, la descrizione, il nome del file in cui l'errore è stato trovato.
- **Postcondizione:** il sistema ha visualizzato l'output della verifica delle relazioni.

4.2.38 UC 1.11.2 Selezione dei filtri sulle categorie di errori da visualizzare

- **Attori:** developer.
- **Descrizione:** un developer deve poter selezionare i filtri sulla severità degli errori.
- **Precondizione:** il developer ha avviato il check sulle relazioni e quest'ultimo è terminato.
- **Scenario principale:** il developer seleziona per ogni categoria di severità se desidera visualizzare gli errori di quella categoria.
- **Postcondizione:** la visualizzazione viene aggiornata e vengono mostrati solo gli errori per il quale corrispettivo filtro di severità è attivo.



4.2.39 UC 1.11.3 Visualizzazione dell'elemento causa dell'errore nell'albero XML

- **Attori:** developer.
- **Descrizione:** un developer deve poter visualizzare l'elemento causa dell'errore nel suo albero XML di appartenenza.
- **Precondizione:** il developer ha avviato il check sulle relazioni seleziona l'azione per mostrare l'elemento causa dell'errore.
- **Scenario principale:** il developer seleziona l'azione per visualizzare il nodo causa dell'errore nel suo albero di appartenenza.
- **Postcondizione:** il sistema ha visualizzato il nodo nel contesto del suo albero di appartenenza.

4.2.40 UC 1.12 Visualizzazione dell'about del programma

- **Attori:** developer.
- **Descrizione:** un developer deve poter visualizzare la finestra about del programma.
- **Precondizione:** il developer abbia selezionato l'azione per visualizzare la finestra about del programma.
- **Scenario principale:** il developer visualizza l'about sul programma.
- **Postcondizione:** il sistema ha mostrato la finestra about del programma.

4.2.41 UC 1.13 Visualizzazione degli errori occorsi durante l'apertura del file

- **Descrizione:** si è verificato uno dei seguenti errori durante l'apertura di un file XML:
 - il percorso inserito non è valido;
 - il file indicato non esiste;
 - il file indicato non è leggibile;
 - si è verificato un errore non gestito durante l'apertura del file;
 - il parser XML ha trovato un errore sintattico nel file XML.
- **Precondizione:** il developer ha selezionato l'azione per aprire un file esistente e abbia inserito il percorso ed il nome del file.
- **Scenario principale:** viene fornita una spiegazione dell'errore commesso e come risolverlo.
- **Postcondizione:** l'errore commesso è stato visualizzato e spiegato, il file XML non è stato aperto. Il precedente lavoro è stato chiuso.



4.2. CASI D'USO

4.2.42 UC 1.14 Visualizzazione degli errori occorsi durante il salvataggio del file

- **Descrizione:** si è verificato uno dei seguenti errori durante il salvataggio su disco di un file XML:
 - il file indicato non è scrivibile;
 - si è verificato un errore non gestito durante l'apertura in scrittura del file.
- **Precondizione:** il developer ha selezionato l'azione per il salvataggio del file modificato.
- **Scenario principale:** viene fornita una spiegazione dell'errore commesso e come risolverlo.
- **Postcondizione:** l'errore commesso è stato visualizzato e spiegato, il file XML non è stato salvato. Il precedente lavoro non è stato perso.

4.2.43 UC 1.15 Visualizzazione degli errori occorsi durante l'importazione

- **Descrizione:** si è verificato uno dei seguenti errori durante l'importazione delle relazioni:
 - il percorso inserito non è valido;
 - il file indicato non esiste;
 - il file indicato non è leggibile;
 - si è verificato un errore non gestito durante l'apertura del file;
 - il parser XML ha trovato un errore sintattico nel file.
- **Precondizione:** il developer ha selezionato l'azione per l'importazione delle relazioni da file ed ha inserito un percorso ed il nome del file.
- **Scenario principale:** viene fornita una spiegazione dell'errore commesso e come risolverlo.
- **Postcondizione:** l'errore commesso è stato visualizzato e spiegato, le relazioni non sono state importate. Le precedenti relazioni non sono state eliminate.

4.2.44 UC 1.16 Visualizzazione degli errori occorsi durante l'esportazione

- **Descrizione:** si è verificato uno dei seguenti errori durante l'esportazione delle relazioni su un nuovo file:
 - il file indicato non è scrivibile;
 - si è verificato un errore non gestito durante l'apertura in scrittura del file.
- **Precondizione:** il developer ha selezionato l'azione per l'esportazione delle relazioni su file.
- **Scenario principale:** viene fornita una spiegazione dell'errore commesso e come risolverlo.
- **Postcondizione:** l'errore commesso è stato visualizzato e spiegato, le relazioni non sono state esportate.



4.3. TRACCIAMENTO DEI REQUISITI

4.3 Tracciamento dei requisiti

Partendo dai casi d'uso si è provveduto a stilare una precisa analisi dei requisiti per il tool in questione. I requisiti trovati sono stati inoltre tracciati in relazione al caso d'uso di origine.

Di seguito vengono riportati tutti i requisiti individuati. Per essere più leggibili verranno separati in tabelle a seconda della loro categoria. Di ogni requisito verranno indicati: tipologia, importanza e provenienza.

I requisiti dovranno essere classificati per tipo e importanza e utilizzeranno la seguente sintassi:

R[Importanza][Tipo][Codice]

- **Importanza:** può assumere solo uno fra i seguenti valori:

- 0: requisito obbligatorio;
- 1: requisito desiderabile;
- 2: requisito opzionale.

- **Tipo:** può assumere solo uno fra i seguenti valori:

- F: funzionale;
- Q: di qualità;
- P: prestazionale;
- V: vincolo.

- **Codice:** è il codice gerarchico univoco di ogni vincolo espresso in numeri (esempio: 1.3.2).

Per ogni requisito vengono inoltre specificati:

- **descrizione:** breve ma completa ed il meno ambigua possibile;

- **fonte:** può essere soltanto una o più tra le seguenti:

- *caso d'uso*: il requisito è stato estrappolato da un caso d'uso. In questo caso va indicato il codice univoco del caso d'uso. È possibile indicare come fonte più di un caso d'uso.
- *interno*: è stato ritenuto giusto aggiungere questo requisito per completezza.
- *tutor aziendale*: il requisito è stato espressamente richiesto dal tutor aziendale.

4.3.1 Requisiti funzionali

Requisito	Descrizione	Fonti
R0F1	Un utente può creare un nuovo file XML.	UC 1.1
R0F1.1	La creazione di un nuovo file XML richiede l'inserimento del percorso dove memorizzarlo.	UC 1.1.1
R0F1.2	La creazione di un nuovo file XML richiede l'inserimento del nome del file.	UC 1.1.2



4.3. TRACCIAMENTO DEI REQUISITI

R0F1.3	Se il file inserito è già presente nel sistema, il contenuto verrà sovrascritto solo nel momento del primo salvataggio.	Interno
R0F2	Un utente può aprire un file XML preesistente.	UC 1.2
R0F2.1	L'apertura di file XML richiede l'inserimento del percorso dove recuperarlo.	UC 1.2
R0F2.2	L'apertura di file XML richiede l'inserimento del nome del file da aprire.	UC 1.2
R0F2.3	Se prima dell'apertura di un file XML l'utente ha delle modifiche non salvate su qualche file, il sistema, per ogni file, chiede all'utente se cancellare l'azione, salvare o scartare le modifiche al lavoro precedente.	UC 1.2
R0F3	Un utente può editare un file XML correntemente aperto e visualizzato.	UC 1.3
R0F3.1	Un utente può selezionare e visualizzare l'elemento riferito da un elemento.	UC 1.3.1
R0F3.1.1	Deve esistere una relazione nel database ed essere istanziata correttamente nell'XML.	Interno
R0F3.2	Il programma mostra gli eventuali errori occorsi durante la ricerca del nodo destinazione della relazione.	UC 1.3.2
R0F3.2	Il programma mostra gli eventuali errori occorsi durante la ricerca del nodo destinazione della relazione.	UC 1.3.2
R0F3.3	Un utente può inserire un nuovo nodo figlio ad un elemento.	UC 1.3.3
R0F3.3.1	Il programma apre automaticamente la schermata di editing del nuovo nodo inserito.	Tutor interno
R0F3.4	Un utente può modificare un nodo dell'albero XML.	UC 1.3.4
R0F3.4.1	Un utente può modificare il valore dell'elemento.	UC 1.3.4.1
R0F3.4.2	Un utente può appendere un nuovo attributo all'elemento.	UC 1.3.4.2
R0F3.4.3	Un utente può modificare un attributo esistente.	UC 1.3.4.3



4.3. TRACCIAMENTO DEI REQUISITI

R0F3.4.4	Un utente può rimuovere un attributo esistente.	UC 1.3.4.4
R0F3.4.5	Il programma blocca il salvataggio se l'utente ha inserito due attributi con lo stesso nome.	Interno
R0F3.5	Un utente può duplicare un nodo, inserendo la copia come fratello del nodo copiato.	UC 1.3.5
R0F3.6	Un utente può copiare un nodo e mantenerlo in memoria per essere incollato in un momento successivo.	UC 1.3.6
R0F3.6.1	Il programma mantiene un solo nodo copiato alla volta.	Interno
R0F3.6.2	Ogni nuovo nodo copiato sostituisce quelli precedenti.	Interno
R0F3.6.3	Se il nodo origine della copia è viene modificato, la copia rimane immutata.	Interno
R0F3.7	Un utente può incollare il nodo precedentemente copiato.	UC 1.3.7
R0F3.8	Un utente può rimuovere un nodo.	UC 1.3.8
R0F3.8	Quando il programma rimuove un nodo, vengono rimossi anche tutti i nodi discendenti.	Interno
R0F3.9	Un utente può rimuovere tutti i discendenti di un nodo.	UC 1.3.9
R0F3.10	Un utente può istanziare un nuova relazione in automatico, senza dover creare manualmente gli elementi necessari.	UC 1.3.10
R0F3.10.1	Affinché l'operazione avvenga con successo è necessario che sia presente un percorso di relazioni che consente di effettuare l'operazione.	Interno
R0F3.10.2	Se il percorso di relazioni contiene più di un elemento, allora il programma creerà dei nodi intermedi per completare l'operazione.	Interno
R0F3.11	Il programma mostra gli eventuali errori occorsi durante l'istanziazione automatica di una relazione.	UC 1.3.11
R0F3.12	Un utente può annullare l'ultima modifica effettuata.	UC 1.3.12



4.3. TRACCIAMENTO DEI REQUISITI

R0F3.13	Un utente può ripristinare l'ultima modifica annullata.	UC 1.3.13
R0F3.14	Quando viene effettuata una modifica ad un file XML, viene aggiunto il carattere '*' alla fine del nome per indicare all'utente il nuovo stato del file.	Interno
R0F4	Un utente può salvare le modifiche effettuata ad un file XML.	UC 1.4
R0F4.1	Il programma permette di salvare solo un file su cui sono state apportate modifiche.	UC 1.4
R0F5	Un utente può modificare il filtro utilizzato per selezionare l'attributo il cui valore viene mostrato nell'albero per rappresentare l'elemento.	UC 1.5
R0F5.1	Il valore inserito dall'utente viene memorizzato nel registro di sistema e ripristinato ad ogni apertura del programma.	UC 1.5
R0F6	Un utente può modificare i file associati al file correntemente aperto.	UC 1.6
R0F6.1	Un utente può aggiungere un nuovo file associato.	UC 1.6.1
R0F6.2	Un utente può rimuovere un file associato precedentemente inserito.	UC 1.6.2
R0F6.3	Il programma, in fase di salvataggio delle modifiche applicate ai file associati, non permette il salvataggio se tutti i file associati inseriti non sono tutti differenti fra loro.	Interno
R0F6.4	Il programma quando salva le modifiche ai file associati, chiude automaticamente quelli rimossi e apre i nuovi inseriti.	Interno
R0F7	Un utente può editare il database dei tipi di relazioni conosciute dal programma.	UC 1.7
R0F7.1	Un utente può inserire un nuovo tipo di relazione.	UC 1.7.1
R0F7.1.1	L'aggiunta di una nuova relazione richiede l'inserimento del nome del tag origine della relazione.	UC 1.7.1



4.3. TRACCIAMENTO DEI REQUISITI

R0F7.1.2	L'aggiunta di una nuova relazione richiede l'inserimento del nome del tag figlio origine della relazione, elemento in cui in un attributo sarà contenuta la chiave dell'elemento destinazione.	UC 1.7.1
R0F7.1.3	L'aggiunta di una nuova relazione richiede l'inserimento del nome dell'attributo in cui è contenuta la chiave dell'elemento destinazione.	UC 1.7.1
R0F7.1.4	L'aggiunta di una nuova relazione richiede l'inserimento del nome del tag destinatario della relazione.	UC 1.7.1
R0F7.1.5	L'aggiunta di una nuova relazione richiede l'inserimento del nome dell'attributo, nell'elemento destinazione, in cui è presente la chiave identificativa.	UC 1.7.1
R0F7.2	Un utente può modificare tutti i parametri di una relazione precedentemente inserita.	UC 1.7.2
R0F7.3	Un utente può eliminare una relazione precedentemente inserita.	UC 1.7.3
R0F7.4	Il programma salva automaticamente, su file di configurazione XML, le modifiche apportate al database di relazioni.	Tutor interno
R0F7.5	Il programma ricarica automaticamente, da file di configurazione XML, il set di relazioni precedentemente presenti nell'ultimo utilizzo del programma.	Tutor interno
R0F8	Un utente può importare un set di relazioni presenti in un file di configurazione.	UC 1.8
R0F8.1	L'importazione richiede che l'utente inserisca il percorso ed il nome del file di configurazione contenente i dati desiderati.	UC 1.8
R0F8.1	Il programma importa le relazioni contenute, eliminando tutte quelle precedentemente presenti nel database.	Interno
R0F8.1	Il programma salva le nuove relazioni nel file di configurazione sostituendo tutte le precedenti.	Interno
R0F9	Un utente può esportare tutte le relazioni attualmente inserite in un file di configurazione esterno.	UC 1.9



4.3. TRACCIAMENTO DEI REQUISITI

R0F9.1	L'esportazione richiede che l'utente inserisca il percorso ed il nome del file di configurazione destinazione.	UC 1.9
R0F9.1.1	Se il file di destinazione non esiste verrà creato.	Interno
R0F10	Un utente può avviare il controllo di consistenza delle relazioni sulla base del set di relazioni conosciute dal programma su tutti i file correntemente aperti.	UC 1.10
R0F11	Un utente può visualizzare i risultati del controllo di consistenza delle relazioni effettuato sulla base del set di relazioni conosciute dal programma.	UC 1.11
R0F11.1	Un utente può visualizzare l'elenco degli errori riscontrati durante l'analisi.	UC 1.11.1
R0F11.1.1	Il programma mostra per ogni errore la tipologia di questo.	UC 1.11.1
R0F11.1.2	Il programma mostra per ogni errore la severità di questo.	UC 1.11.1
R0F11.1.3	Il programma mostra per ogni errore una descrizione testuale di questo.	UC 1.11.1
R0F11.1.4	Il programma mostra per ogni errore il nome del file nel quale è stato riscontrato.	UC 1.11.1
R0F11.2	Un utente può selezionare un filtro che permette di nascondere o mostrare ogni categoria di errori.	UC 1.11.2
R0F11.3	Un utente può decidere di visualizzare, all'interno dell'albero XML, l'elemento a cui l'errore si riferisce.	UC 1.11.3
R0F12	Un utente può visualizzare la finestra di about sul programma.	UC 1.12
R0F13	Il programma mostra gli eventuali errori occorsi durante l'apertura di un file XML.	UC 1.13
R0F14	Il programma mostra gli eventuali errori occorsi durante il salvataggio di un file XML.	UC 1.14
R0F15	Il programma mostra gli eventuali errori occorsi durante l'importazione di un set di relazioni.	UC 1.15



4.4. TECNOLOGIE E STRUMENTI

R0F16	Il programma mostra gli eventuali errori occorsi durante l'esportazione di un set di relazioni.	UC 1.16
-------	---	---------

tabella 4.1: Requisiti funzionali di XML Editor

4.3.2 Requisiti di qualità

Requisito	Descrizione	Fonti
R0Q1	Viene fornito insieme al programma un l'help con la guida alla compilazione e al deploy per sistemi Windows.	Tutor interno

tabella 4.2: Requisiti di qualità di XML Editor

4.3.3 Requisiti di vincolo

Requisito	Descrizione	Fonti
R0V1	Il programma deve funzionare su Windows 7 SP1 32 e 64 bit.	Tutor interno
R0V2	Il programma deve essere scritto utilizzando il linguaggio C++ 98.	Tutor interno

tabella 4.3: Requisiti di vincolo di XML Editor

4.4 Tecnologie e strumenti

Di seguito viene fornita una panoramica delle tecnologie e strumenti utilizzati per lo sviluppo di XML Editor.

4.4.1 Qt® Framework

Per lo sviluppo del tool è stato usato, come per Multiplatform File Analyzer, Qt® Framework versione 5.3.1.

In particolare è stata usata la libreria per costruire le interfacce grafiche, la libreria per accedere al file system e leggere/scrivere file e la libreria per effettuare il parsing dei file XML.

4.4.2 Qt® Creator

Come IDE per lo sviluppo ci si è affidati a Qt® Creator 3.1.2. IDE semplice e performante che si integra perfettamente con il mondo Qt®, offrendo addirittura la



4.5. CICLO DI VITA DEL SOFTWARE

consultazione veloce della documentazione direttamente nell'IDE.

4.5 Ciclo di vita del software

XML Editor è stato sviluppato seguendo un modello di ciclo di vita incrementale. È stato deciso di fornire due incrementi. Nel primo sono state inserite tutte le funzionalità per la visualizzazione dei file, la gestione dei file associati e le relazioni (inclusa la verifica). Nel secondo incremento sono state aggiunte alla base già consolidate le funzionalità di editing dei file. Ad ogni incremento è seguito un rilascio del software.

4.6 Progettazione

Questo software a differenza di Multiplatform File Analyzer, è stato sviluppato secondo il pattern Model-View (todo bibliografica <http://qt-project.org/doc/qt-5/model-view-programming.html>) ideato dagli ingegneri di Qt® e sul quale si basa tutto il loro framework. La scelta è stata fatta ragionando sui risultati ottenuti durante lo sviluppo del precedente tool. In esso le esigenze erano diverse, ad esempio la possibilità di poterlo usare da riga comando forzava la scelta verso MVC e il maggior disaccoppiamento possibile dalle classi di Qt®. Essendo invece questo un editor visuale, che si sarebbe andato a basare proprio sul meccanismo di model-view per gestire ad esempio l'albero XML, si è deciso di sperimentare questo pattern per l'intero sistema. Per lo stagista, questa è la prima volta che si affaccia a questo pattern architetturale ed il suo utilizzo porterà sicuramente una maggior esperienza nella progettazione, permettendogli di valutare pro e contro rispetto al classico MVC.

La strutturazione generale di alto livello segue quindi il design presentato dal pattern model-view, puntando ad inserire nelle view di competenza le logiche di gestione che, nel mondo MVC, sarebbero appartenute al controller.

I pacchetti in cui il software è stato partizionato sono i seguenti:

- **command**: contenente le classi che partecipano all'omonimo design pattern;
- **core**: contenente le classi che implementano il pattern *Strategy* per la realizzazione del controllo sulle relazioni;
- **data**: a cui appartengono le classi di gestione dei dati;
- **observer**: implementazione dell'omonimo design pattern;
- **view**: le classi di visualizzazione del sistema che contengono anche le logiche di controllo.

Per permettere una facile e veloce connessione tra gli strati è stato adottato il Design Pattern *Observer*, implementato nell'omonimo pacchetto *observer*.

È stato scelto di utilizzare il Design Pattern *Strategy* per rendere modificabile facilmente l'implementazione dell'algoritmo che effettua l'analisi, all'interno del pacchetto *core*.

A differenza del precedente tool, qui il framework Qt® è stato usato a tutto tondo ovunque, senza circoscriverne l'uso.

4.6.1 Diagrammi delle classi

Vista la vastità del software è stato impossibile creare un diagramma che contenesse contemporaneamente tutte le classi e le relazioni. Sono stati quindi creati due diagramma che permettono di visualizzare due punti importanti dell'architettura. Tutti i diagrammi presenti rispettano il formalismo UML 2.0.



Data-Model-View diagram

In questo diagramma è possibile visualizzare l'integrazione delle classi model-view di Qt[®] utilizzate per la visualizzazione dell'albero XML a partire dalla classe di dati. È inoltre presente la logica per il costante aggiornamento della visualizzazione al cambiamento dei dati.

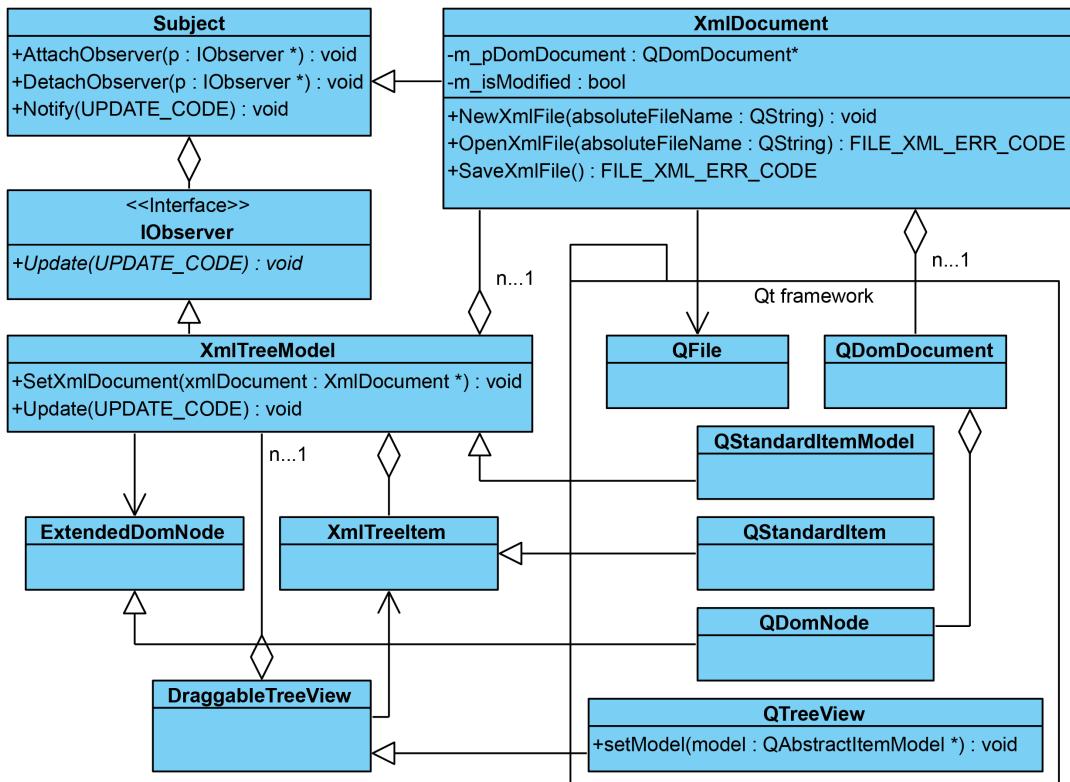


figura 4.8: Diagramma dell'integrazione delle classi model-view di Qt[®] per la visualizzazione dell'albero XML.

XML editing diagram

In questo diagramma è possibile visualizzare l'implementazione del design pattern *command* usato per gestire l'editing dei file XML. È possibile notare la presenza del design pattern *composite*, inserito allo scopo di permettere la creazione di azioni che si comportassero come singole unità, ma contenessero una aggregazione di azioni base. Un esempio di utilizzo è l'istanziazione automatica di una nuova relazione, la quale crea una singola azione `XmlEditCommandAggregator` che però esegue tutti i passi intermedi necessari. È a questo punto possibile annullare l'istanziazione con un'unica invocazione del metodo `Undo` dell'invoker.



4.6. PROGETTAZIONE

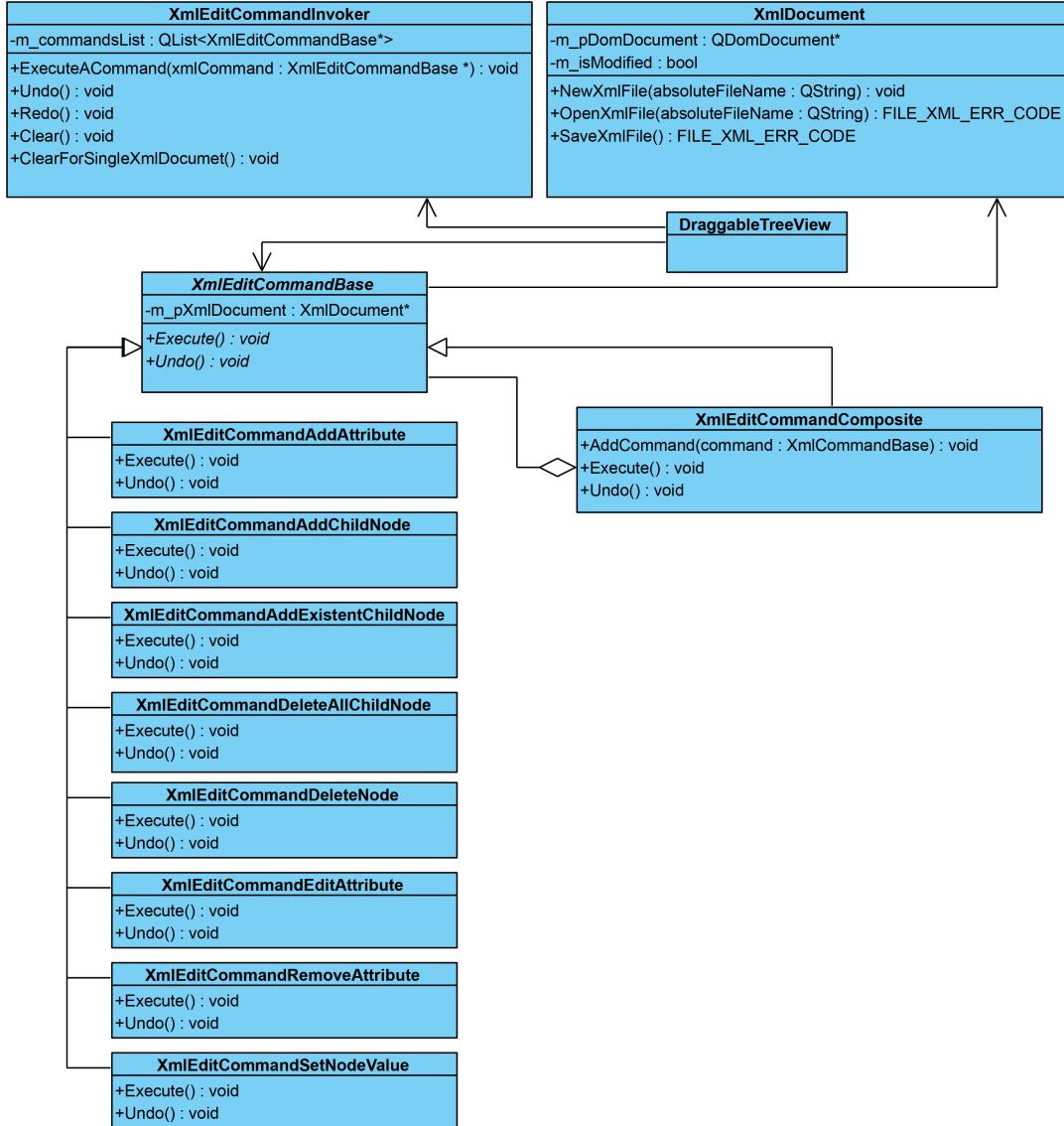


figura 4.9: Diagramma delle classi usate per l'editing dei file XML.

4.6.2 Pacchetto observer

Questo pacchetto rappresenta l'implementazione del Design Pattern *Observer* ed è stato utilizzato per tenere aggiornata la view al cambiamento dei dati.

Interfaccia IObserver

I^observer è l'interfaccia che rappresenta gli oggetti che osservano le modifiche di altri oggetti. Essa contiene il metodo virtuale puro *Update* che i soggetti osservati invocano per segnalare il fatto che sono stati modificati e che quindi un aggiornamento è necessario. Le classi concrete che avranno la necessità di osservare un soggetto deriveranno da I^observer implementando il metodo *Update*.

Classe Subject

La classe *Subject* rappresenta un soggetto che può essere osservato. Essa contiene il codice necessario per notificare tutti gli osservatori dell'avvenuto cambiamento.



4.6. PROGETTAZIONE

Mantiene inoltre una lista degli osservatori da notificare, i quali possono iscriversi se desiderano ricevere la notifica, oppure rimuoversi se non è più necessario ricevere aggiornamenti. Le classi che devono essere osservate deriveranno da questa.

4.6.3 Pacchetto data

Il pacchetto `data` contiene tutte le classi che rappresentano i dati di business del programma. Come classi aggregazione di base sono state scelte quelle offerte dal framework Qt® per la loro maggiore integrazione e l'ampio set di funzionalità offerto.

Classe AssociatedFiles

La classe `AssociatedFiles` raccoglie tutti i nomi ed i percorsi dei file associati ad un main file. È sua la responsabilità di mantenere la persistenza di queste informazioni tramite la lettura e scrittura nel registro di sistema.

Classe AttributeNameTagCollapse

La classe `AttributeNameTagCollapse` contiene il filtro sull'attributo da visualizzare nell'albero XML. È sua la responsabilità di mantenere la persistenza di questa informazioni tramite la lettura e scrittura nel registro di sistema.

Classe XmlDocument

La classe `XmlDocument` rappresenta un file XML aperto nel programma. Ne gestisce la lettura/scrittura su disco e l'effettiva implementazione dell'editing dell'XML.

Classe XmlRelation

La classe `XmlRelation` rappresenta un prototipo di relazione, contenendo tutti i dati necessari.

Classe XmlRelationCollection

La classe `XmlRelationCollection` funge da contenitore per la gestione del set di relazioni cui il programma è a conoscenza.

Classe XmlRelationError

La classe `XmlRelationError` rappresenta un errore trovato durante l'analisi di consistenza delle relazioni sulla base dei prototipi conosciuti. Contiene tutte le informazioni dell'errore, le quali saranno mostrate all'utente. È qui presente la logica che compone la stringa descrittiva dell'errore.

Classe XmlTreeModel

La classe `XmlTreeModel` rappresenta un modello osservabile da una classe view del framework Qt®. Estende le classi `QStandardItemModel` e `Subject`. Essa osserva tramite il pattern *observer* la classe `XmlDocument` e trasforma il documento XML in un modello visualizzabile dalla classe `DraggableTreeView`. La classe è un'implementazione del design pattern *Adapter*.

Classe XmlTreeItem

La classe `XmlTreeItem` estende un classico item del model Qt® (`QStandardItem`) inserendo i dati aggiuntivi necessari alla corretta visualizzazione.



4.6. PROGETTAZIONE

4.6.4 Core

Il pacchetto si occupa della realizzazione dell’analisi e verifica delle relazioni. Utilizza il Design Pattern *Strategy* per l’organizzazione delle classi contenute.

Classe XmlRelationCheckerCoreBase

`XmlRelationCheckerCoreBase` rappresenta l’algoritmo di che effettua l’analisi delle relazioni e ritorna una collazione di `XmlRelationError` con gli errori trovati.

Classe XmlRelationCheckerCoreImpl

La classe `XmlRelationCheckerCoreImpl` concretizza la base astratta `XmlRelationCheckerCoreBase` implementando gli algoritmi che effettuano l’analisi dei file verificando la consistenza delle relazioni.

4.6.5 Pacchetto command

Il pacchetto si occupa della gestione delle operazioni di editing dei file XML. Utilizza il Design Pattern *Command* per l’organizzazione delle classi contenute.

Classe XmlEditCommandBase

`XmlEditCommandBase` è una classe astratta che rappresenta una operazione eseguibile su un file XML. Mantiene il puntatore al documento XML da editare che verrà usato dalle sottoclassi concrete.

Classe XmlEditCommandInvoker

`XmlEditCommandInvoker` è la classe che esegue realmente le operazioni mantenendo uno storico di ciò che è stato effettuato offrendo quindi le funzionalità di `Undo` e `Redo`.

Classe XmlEditCommandAggregator

`XmlEditCommandAggregator` è la classe che implementa il design pattern *Command* permettendo l’uso di più comandi come fossero uno singolo.

Classe XmlEditCommandAddAttribute

`XmlEditCommandAddAttribute` è la classe che implementa il comando di aggiunta di un attributo. Essa mantiene lo stato allo scopo di poter eseguire l’azione di `Undo`. Per la reale implementazione dell’operazione che essa rappresenta si base sui metodi esposti dalla classe `XmlDocument`.

Classe XmlEditCommandAddChildNode

`XmlEditCommandAddChildNode` è la classe che implementa il comando di aggiunta di nuovo nodo vuoto. Essa mantiene lo stato allo scopo di poter eseguire l’azione di `Undo`. Per la reale implementazione dell’operazione che essa rappresenta si base sui metodi esposti dalla classe `XmlDocument`.

Classe XmlEditCommandAddExistentChildNode

`XmlEditCommandAddExistentChildNode` è la classe che implementa il comando di aggiunta di un nodo a partire dalla copia di un nodo precedentemente esistente. Essa mantiene lo stato allo scopo di poter eseguire l’azione di `Undo`. Per la reale implementazione dell’operazione che essa rappresenta si base sui metodi esposti dalla classe `XmlDocument`.



Classe XmlEditCommandDeleteAllChildNode

`XmlEditCommandDeleteAllChildNode` è la classe che implementa il comando di rimozione di tutti i nodi discendenti di un elemento. Essa mantiene lo stato allo scopo di poter eseguire l'azione di Undo. Per la reale implementazione dell'operazione che essa rappresenta si base sui metodi esposti dalla classe `XmlDocument`.

Classe XmlEditCommandDeleteNode

`XmlEditCommandDeleteNode` è la classe che implementa il comando di eliminazione di un attributo e tutti i suoi discendenti. Essa mantiene lo stato allo scopo di poter eseguire l'azione di Undo. Per la reale implementazione dell'operazione che essa rappresenta si base sui metodi esposti dalla classe `XmlDocument`.

Classe XmlEditCommandEditAttribute

`XmlEditCommandEditAttribute` è la classe che implementa il comando di editing di un attributo. Essa mantiene lo stato allo scopo di poter eseguire l'azione di Undo. Per la reale implementazione dell'operazione che essa rappresenta si base sui metodi esposti dalla classe `XmlDocument`.

Classe XmlEditCommandRemoveAttribute

`XmlEditCommandRemoveAttribute` è la classe che implementa il comando di rimozione di un attributo. Essa mantiene lo stato allo scopo di poter eseguire l'azione di Undo. Per la reale implementazione dell'operazione che essa rappresenta si base sui metodi esposti dalla classe `XmlDocument`.

Classe XmlEditCommandSetnodeValue

`XmlEditCommandSetnodeValue` è la classe che implementa il comando di editing del valore di un elemento. Essa mantiene lo stato allo scopo di poter eseguire l'azione di Undo. Per la reale implementazione dell'operazione che essa rappresenta si base sui metodi esposti dalla classe `XmlDocument`.

4.6.6 Pacchetto view

Il pacchetto si occupa del display dell'interfaccia utente e della gestione della logica di controllo del software.

Classe MainWindowView

La classe `MainWindowView` eredita e specializza la classe `QMainWindow` di Qt[®]. Essa è il cuore della logica del programma. Raccoglie e organizza i comandi utente di base, delegando l'esecuzione alle altri componenti del pacchetto.

Classe DialogAddModifyRelation

La classe `DialogAddModifyRelation` è una dialog creata allo scopo di permettere l'inserimento dei parametri che caratterizzano una relazione da parte dell'utente.

Classe DialogAssociatedFilesBase

La classe `DialogAssociatedFilesBase` è una base astratta di una finestra di dialog che permette all'utente di gestire i file associati al main file correntemente aperto.



4.6. PROGETTAZIONE

Classe DialogAssociatedFilesForm

La classe `DialogAssociatedFilesForm` concretizza la base astratta `DialogAssociatedFilesBase` creando un’interfaccia atta alla gestione dei file associati.

Classe DialogEditAttributeNameTagCollapseBase

La classe `DialogEditAttributeNameTagCollapseBase` è una base astratta di una finestra di dialog che permette all’utente di gestire il filtro sull’attributo da visualizzare nell’albero XML.

Classe DialogEditAttributeNameTagCollapse

La classe `DialogEditAttributeNameTagCollapse` concretizza la base astratta `DialogEditAttributeNameTagCollapseBase` creando un’interfaccia atta alla modifica del filtro sull’attributo da visualizzare nell’albero XML.

Classe DialogEditNodeBase

La classe `DialogEditNodeBase` è una base astratta di una finestra di dialog che permette all’utente di editare un nodo XML.

Classe DialogEditNodeTable

La classe `DialogEditNodeTable` concretizza la base astratta `DialogEditNodeBase` creando un’interfaccia atta all’editing del valore del nodo e di tutti i suoi attributi.

Classe DialogInputFileNameBase

La classe `DialogInputFileNameBase` è una base astratta di una finestra di dialog che permette all’utente di inserire il percorso ed il nome di un file.

Classe DialogInputFileNameForm

La classe `DialogInputFileNameForm` concretizza la base astratta `DialogInputFileNameBase` creando un’interfaccia atta all’inserimento di un percorso ed un nome di file. Offre la possibilità di esplorare il file system per un più agevole inserimento. Questa classe è usata per acquisire il file da dove importare o verso dove esportare le relazioni.

Classe DialogInputStringBase

La classe `DialogInputStringBase` è una base astratta di una finestra di dialog che permette all’utente di inserire una stringa.

Classe DialogInputStringForm

La classe `DialogInputStringForm` concretizza la base astratta `DialogInputStringBase` creando un’interfaccia atta all’inserimento di una stringa da parte dell’utente.

Classe DialogXmlRelationManagementBase

La classe `DialogXmlRelationManagementBase` è una base astratta di una finestra di dialog che permette all’utente di gestire il database di relazioni.

Classe DialogXmlRelationManagementTable

La classe `DialogXmlRelationManagementTable` concretizza la base astratta `DialogXmlRelationManagementBase` creando un’interfaccia atta alla gestione delle relazioni, visualizzate sotto forma tabellare.



Classe ViewXmlRelationErrorsBase

La classe `ViewXmlRelationErrorsBase` è una base astratta di una finestra di dialog che permette all'utente di visualizzare il risultato del controllo di consistenza delle relazioni.

Classe ViewXmlRelationErrorsTable

La classe `ViewXmlRelationErrorsTable` concretizza la base astratta `ViewXmlRelationErrorsBase` creando un'interfaccia tabellare per la visualizzazione dei risultati sul check delle relazioni.

Classe DraggableTreeView

La classe `DraggableTreeView` deriva dalla classe `QTreeView` del framework Qt®. Essa è quindi in grado di visualizzare la classe model `XmlTreeModel` nella quale abilita il drag & drop per l'istanziazione automatica. È sua responsabilità l'invocazione dei command tramite l'invoker.

Classe XmlViewBase

La classe `XmlViewBase` è una base astratta di un widget per la visualizzazione di file XML.

Classe XmlViewTree

La classe `XmlViewTree` concretizza la base astratta `XmlViewBase` creando un widget che mostra un file XML sotto forma di albero.

4.7 Codifica

Tutto il codice del tool è stato pubblicato sulla piattaforma GitHub®, accedibile tramite il seguente indirizzo: <https://github.com/Mauxx91/XML-Editor>. Tutto il codice è disponibile gratuitamente sotto licenza GNU GPL v3²

Tutto il codice è stato scritto in lingua inglese, compresi i commenti, dei quali si è cercato di scriverne il più possibile. La codifica ha seguito alcune convenzioni della famosa notazione Ungherese. Di seguito sono elencati i formalismi utilizzati nel codice:

- **prefissi:**

- **m**: usato per le variabili membro (esempio: `m_keyName`);
- **p**: usato per le variabili puntatore (esempio: `m_pkeyNameList`);
- **o**: usato per le variabili input alla funzione che sono usate come output (esempio: `o_foundKeys`);
- **I**: usato per le classi interfacce (esempio: `IObserver`).

- **suffissi:**

- **Base**: usato per le classi astratte (esempio: `ResultWidgetBase`);

- **capitalizzazione:**

- **variabili e parametri**: iniziano sempre in minuscolo e usano una lettera minuscola per ogni parola (esempio: `m_parentWidget`, `returnValue`);

²Un approfondimento sulla licenza può essere trovato al seguente indirizzo: <http://www.gnu.org/copyleft/gpl.html>.



4.8. USO PRATICO

- **macro ed enum:** completamente in maiuscolo con le diverse parole separate dal carattere ‘_’ (esempio: UPDATE_CODE);
- **classi, funzioni e metodi:** iniziano sempre in maiuscolo e ogni parola diversa parte in maiuscolo (esempio: FileOccurrence);

4.8 Uso pratico

In questa sezione si vuol presentare degli screenshot di uso pratico di XML Editor.

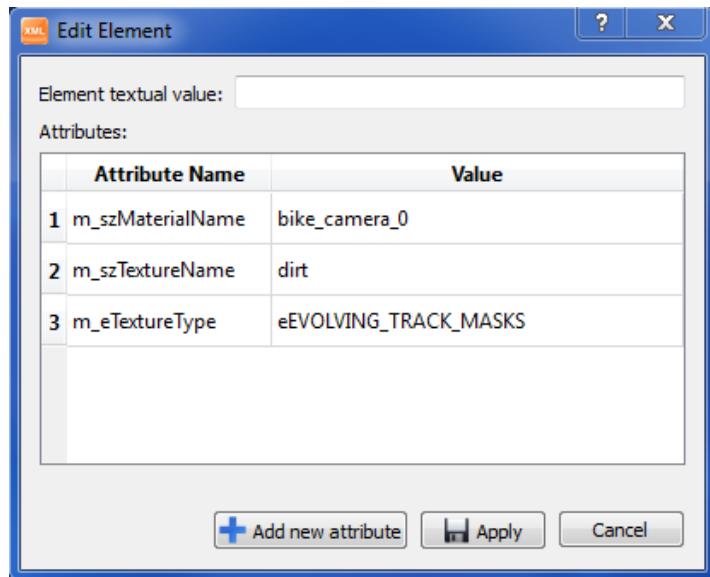


figura 4.10: Screenshot di XML Editor nel quale si può osservare la dialog di edit di un nodo dell'albero XML



4.8. USO PRATICO

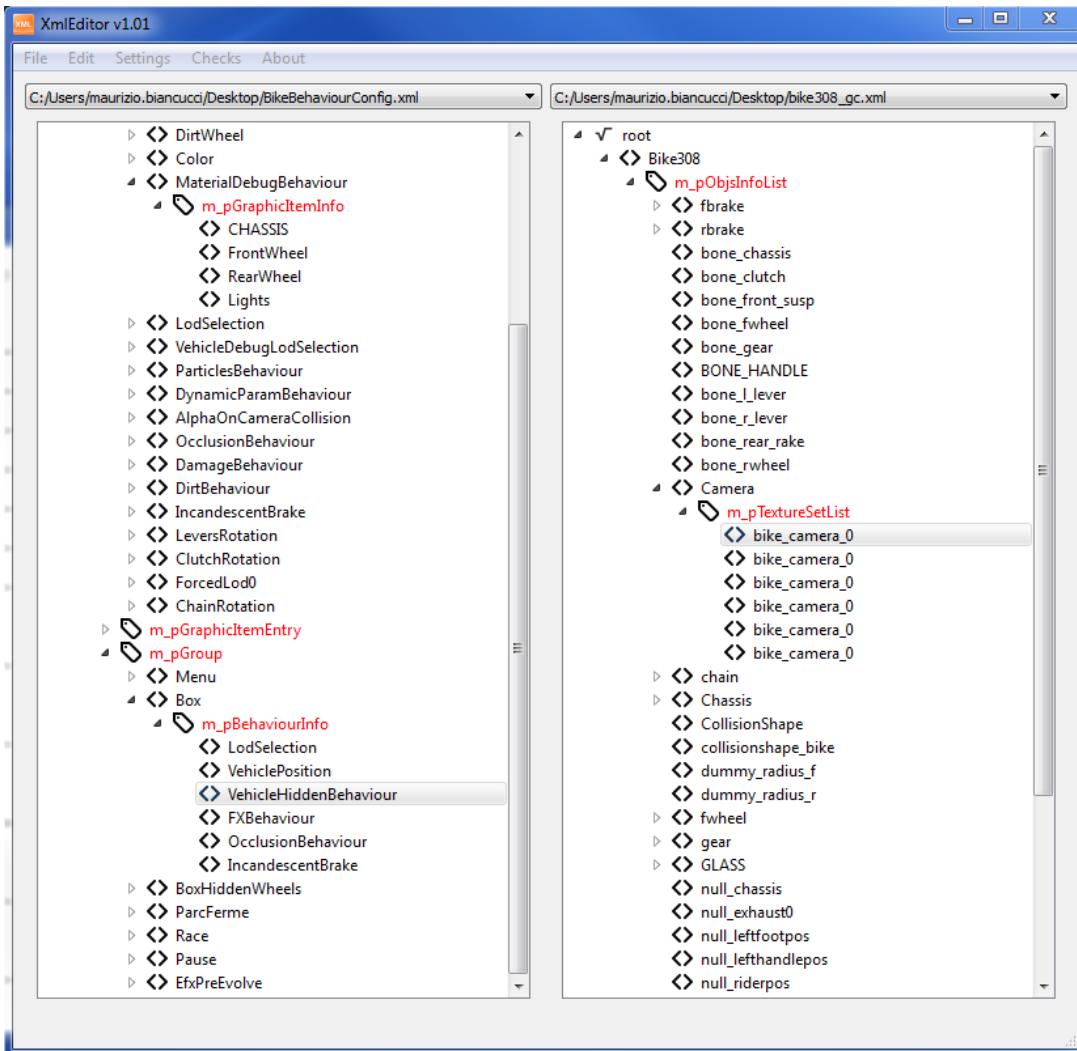


figura 4.11: Screenshot di XML Editor nel quale si può osservare la visualizzazione dei file XML come alberi



4.9. CONCLUSIONI

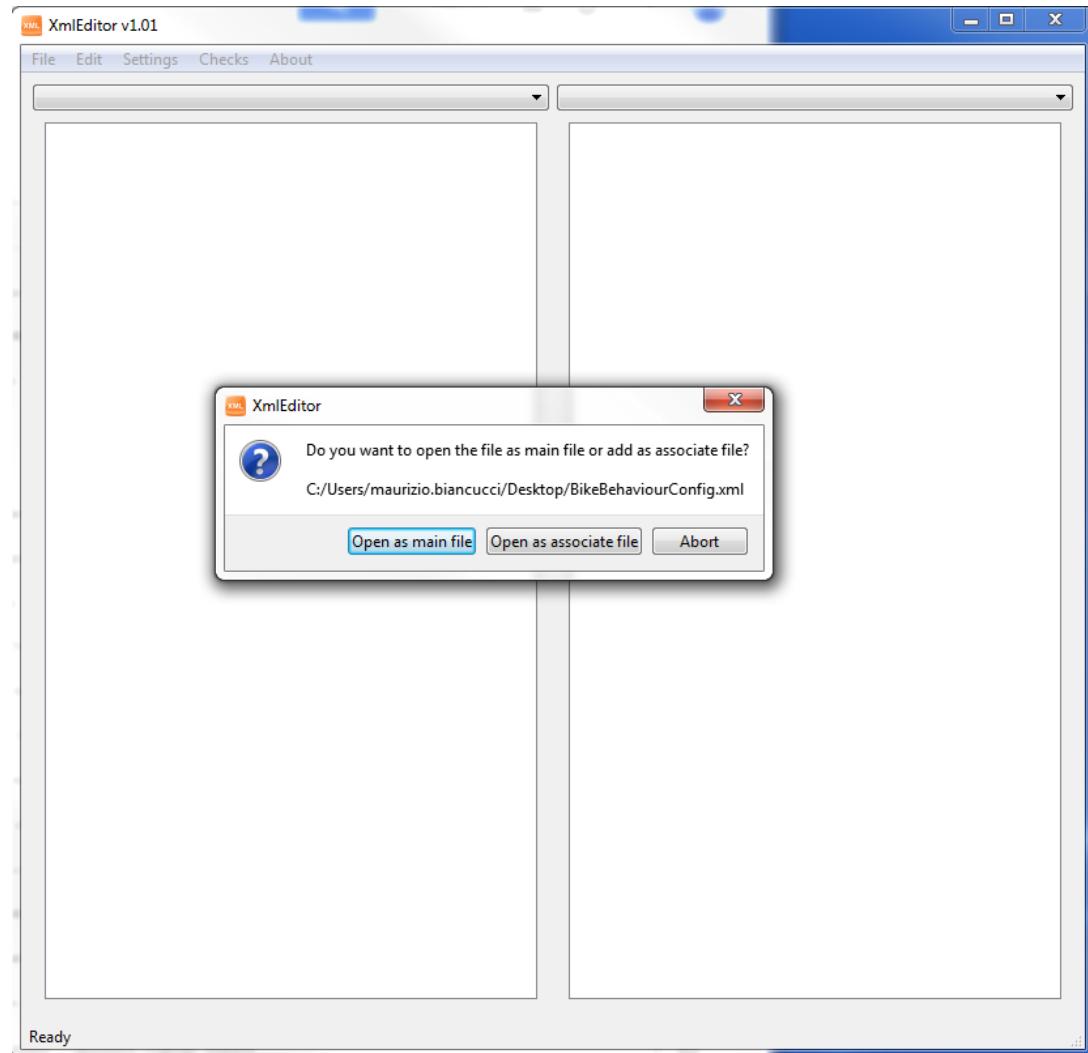


figura 4.12: Screenshot di XML Editor nel quale si può osservare la dialog che chiede all'utente come procedere dopo il drag & drop di un file da Windows

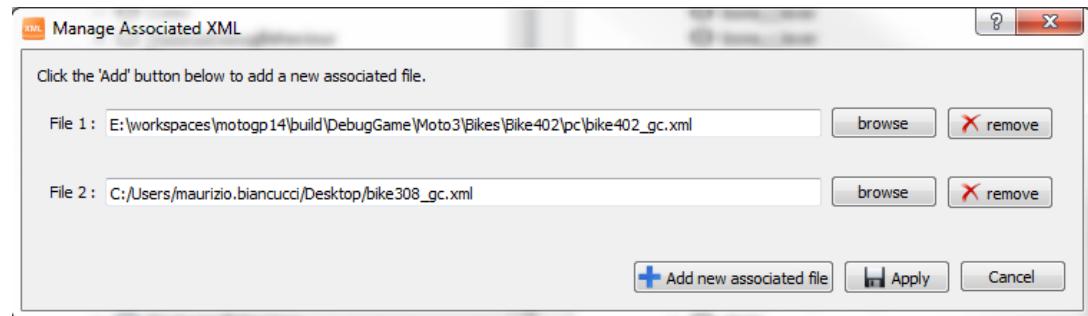


figura 4.13: Screenshot di XML Editor nel quale si può osservare la dialog che permette l'editing dei file associati

4.9 Conclusioni

Tutti i requisiti sono stati soddisfatti come pianificato ma, il secondo incremento comprendente le funzionalità di editing è stata completata con 10 ore di ritardo sulla



4.9. CONCLUSIONI

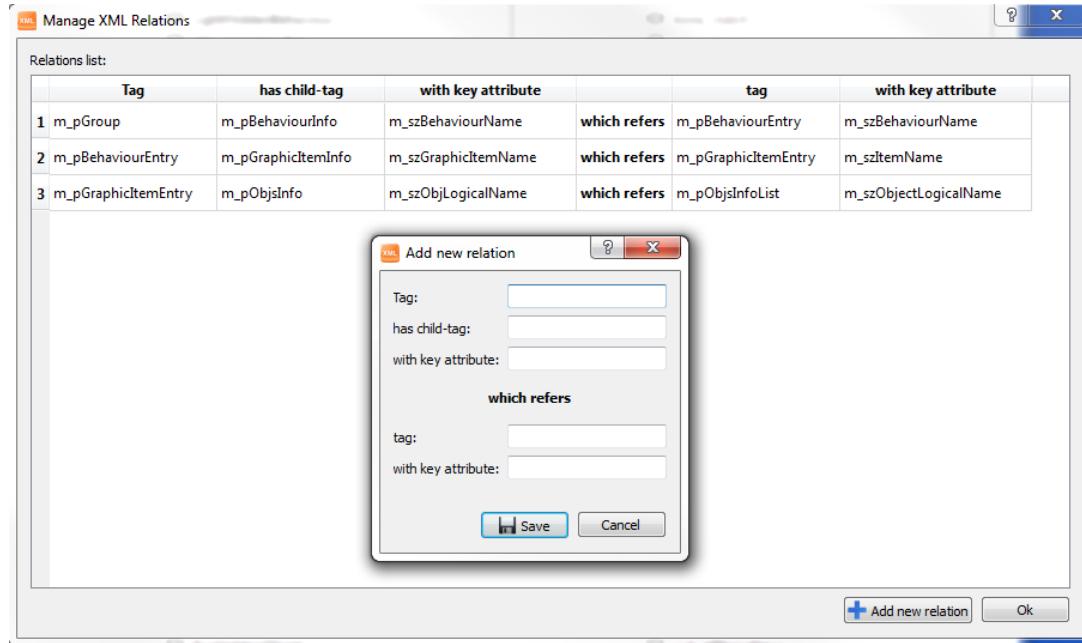


figura 4.14: Screenshot di XML Editor nel quale si può osservare la dialog che permette l'editing delle relazioni

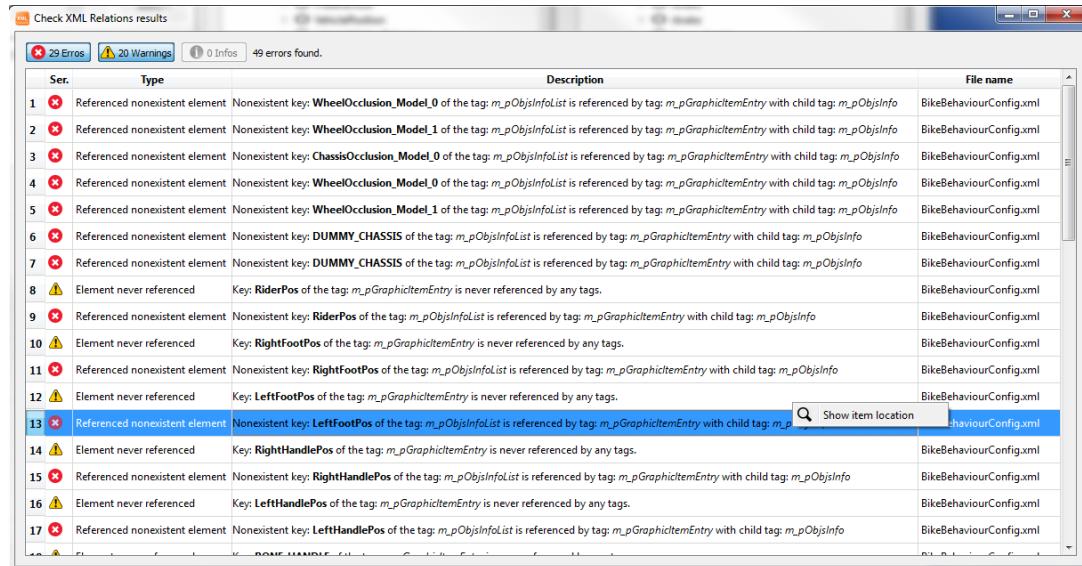


figura 4.15: Screenshot di XML Editor nel quale si può osservare la dialog che permette la visualizzazione dell'esito del risultato del check delle relazioni

pianificazione iniziale. la realizzazione di questo tool ha permesso allo studente di conoscere davvero approfonditamente il framework Qt® ed implementare un tool di buona utilità anche all'esterno dell'azienda. L'utilizzo del nuovo design pattern architettonurale model-view si è rivelato davvero adatto allo sviluppo di sistemi di questa tipologia, dimostrandosi una valida alternativa a MVC.

Capitolo 5

Funzionalità di disegno di elementi grafici 3D a scopo di debug

Lo studente è stato chiamato ad effettuare un refactor¹ ed implementare nuove funzionalità di disegno di oggetti 3D in game a scopo di debug. Le modifiche sono state apportate al gioco MotoGP 14, delle quali alcune sono state portate a MXGP HD.

Si è reso necessario dapprima un’analisi del codice allo scopo di individuare i flussi di interesse e localizzare le vecchie implementazioni su cui effettuare un refactor e sfruttare come base per l’implementazione delle nuove.

5.1 Stampa dei livelli di LOD di moto e piloti

In gioco esistono dei moduli grafici che implementano logiche di gestione degli actor di gioco. In particolare esistono in gioco due moduli che gestiscono il calcolo dei livelli di $LOD_{|g|}$ per le moto ed i piloti. Entrambi i moduli sono istanze della classe `LodSelectionModule`

Gli algoritmi più utilizzati per calcolare i livelli di LOD degli oggetti in una scena 3D sono principalmente due. Il primo e più semplice è quello che calcola semplicemente la distanza fra la camera e l’oggetto 3D e scala i livelli secondo una generica funzione (la quale può cambiare a seconda delle tipologie di oggetti o particolari esigenze). Il secondo metodo invece consiste nel calcolare la quantità di pixel occupati dall’oggetto in relazione alla finestra di disegno. Essendo l’oggetto 3D potenzialmente formato da migliaia di poligoni, il calcolo preciso della superficie occupata richiederebbe molto più tempo di quello guadagnato scalando i livelli di dettaglio, si usa quindi un’approssimazione, ad esempio calcolando la superficie della bounding sphere.

Il secondo metodo risulta quasi necessario in giochi racing come MotoGP poiché esistono casi in cui il primo algoritmo fallisce molto più spesso del secondo. Si immagini ad esempio un grosso edificio in lontananza visibile dalla pista. Con il primo metodo, l’oggetto in questione, essendo distante, avrà il minimo livello di dettaglio disponibile anche se, essendo anche molto grande occuperà molto spazio su schermo, risultando sgradevole alla vista.

¹Per effettuare refactor sono state seguite le linee guida spiegate in [MF99].



5.1. STAMPA DEI LIVELLI DI LOD DI MOTO E PILOTI



figura 5.1: Screenshot di MotoGP 14: con il primo metodo la torre sarebbe al lod più basso, invece con il secondo continua ad essere gradevole alla vista

Come già accennato in MotoGP 14 i moduli che gestiscono i livelli di LOD per moto e piloti sono due istanze della classe `LodSelectionModule`. Essa fornisce, attivabile da Beholder, la funzionalità di stampa dei livelli di LOD correnti degli oggetti per i quali gestisce il livello di dettaglio, in questo caso di moto e piloti presenti al momento in pista. Il problema in questo modulo, era la stampa dei livelli di LOD in posizioni errate, non permettendo di associare il LOD stampato con il suo oggetto.

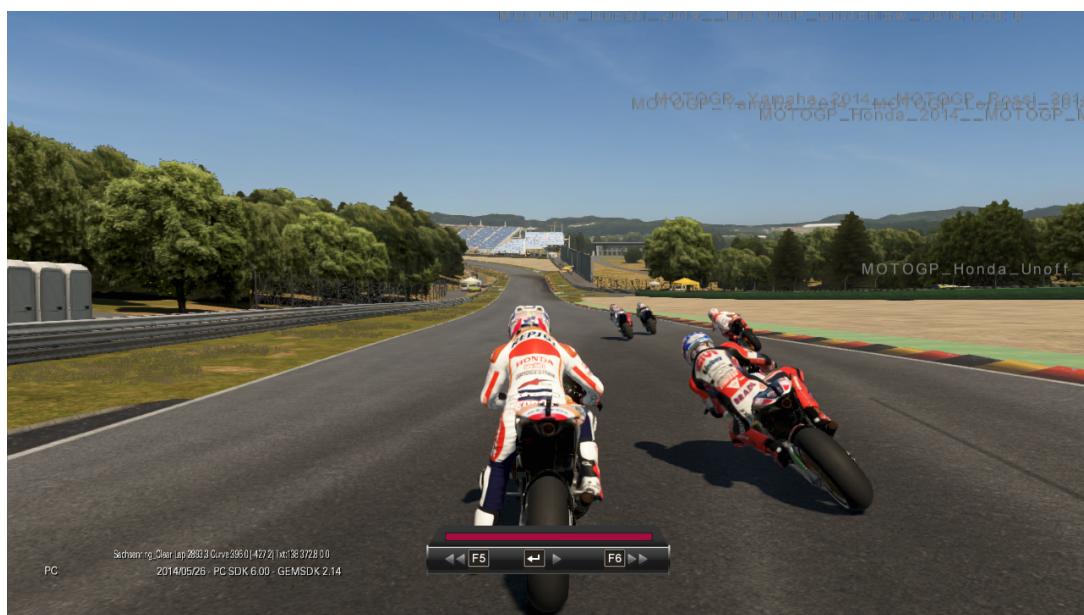


figura 5.2: Screenshot di MotoGP 14 nel quale si può osservare la vecchia stampa dei livelli di LOD

L'obiettivo era quindi quello di avere la stampa dell'informazione affiancata all'oggetto, allo scopo di rendere più agevole e veloce il debug e il tuning per ogni



5.1. STAMPA DEI LIVELLI DI LOD DI MOTO E PILOTI

piattaforma dell'algoritmo di scelta del LOD.

Si parla di tuning dell'algoritmo per ciascuna piattaforma poiché, avendo caratteristiche hardware diverse, le varie piattaforme potranno gestire differenti combinazioni di oggetti con LOD alti. Esiste ad esempio un parametro che indica il massimo numero di bike con LOD 1². È compito del team 3D trovare i giusti parametri per permettere il miglior livello grafico possibile per ogni piattaforma.

Per risolvere il problema è stata inizialmente stata quindi creata una funzione che trasformasse il punto 3D indicante la posizione dello scene actor, in un punto 2D relativo alle coordinate della finestra di gioco. Punto con il quale si procedeva con la stampa dell'informazione. Viene di seguito riportata la funzione creata.

Listing 5.1: Metodo per la trasformazione di un punto 3D in un punto 2D della finestra di gioco rispetto la telecamera corrente

```
void GetScreenPosition(const CCamera* i_pCamera, GVector3& o_position) const
{
    //Transform to the homogeneous clip space

    //World transformation it's needed
    o_position.TransformPoint(poCamera->GetViewMatrix());
    o_position.TransformPoint(poCamera->GetProjMatrix());

    //Now the point is in the range [-1,1] for the x and y axis

    //Transform to the windows space
    o_position.x = (o_position.x + 1) * poCamera->GetViewport().m_uiWidth;
    o_position.y = (o_position.y + 1) * poCamera->GetViewport().m_uiHeight;

    //The x coordinate of o_position is simply ignored
}
```

La funzione GetScreenPosition procede a calcolare le stesse trasformazioni che esegue la GPU per trasformare ogni pixel dell'oggetto nella scena 3D in un'immagine 2D³. Si ottengono quindi le coordinate x e y dello schermo nel range [-1, 1]. Si procede quindi a trasformarle in coordinate relative allo spazio dell'intera finestra di gioco (ad esempio 1920x1080px).

Con un'analisi più approfondita del codice si è successivamente scoperta la presenza di un metodo della classe CCamera che produceva esattamente gli stessi risultati. Si è quindi provveduto a scartare questa implementazione (seppur corretta) ed usare quella già presente allo scopo di evitare inutili duplicazioni di codice. Di seguito è presente la funzione della classe LodSelectionModule che lancia il disegno del LOD attivo. Questa funzione è lanciata per ogni oggetto gestito dalla classe.

Listing 5.2: Metodo per la stampa del LOD corrente

```
void LodSelectionModule::PrintLodDebugInformation(CWorld* iWorld,
    GraphicComponentInfo* i_pComponentInfo )
{
    //Foreach active camera
```

²LOD 1 è il massimo livello di dettaglio.

³Il lettore potrà trovare un approfondimento in [Lun12, chp. 5].



5.1. STAMPA DEI LIVELLI DI LOD DI MOTO E PILOTI

```
for (GUInt camIdx = 0; camIdx < m_pWorld->GetNumberOfCameras();  
    ++camIdx)  
{  
    //Get the string with the lod information and the name of  
    //the 3D object  
    FixedString256 text;  
    i_pComponentInfo->LodToString(camIdx, text);  
  
    //Use the old function that calculates the screen point  
    GVector2 screenPosition;  
    iWorld->GetCurrentCamera(camIdx)->ProjectPoint(  
        i_pComponentInfo->m_pBehavioutInput->GetMatrix().  
        GetTranslation(), screenPosition);  
  
    //Draw the LOD information  
    DebugTextDrawer::GetInstance()->AppendLog(text.c_str(),  
        screenPosition.x, screenPosition.y);  
}  
}
```

Come si può facilmente capire dal codice l'invocazione del metodo `LodToString` compone la stringa che verrà stampata affianco all'oggetto 3D. Si è modificato tale metodo in modo tale che scrivesse come prima informazione il LOD e poi il nome dello scene actor. Questo perché se l'oggetto è nella parte destra della finestra e ha un nome lungo si corre il rischio che l'informazione più importante (il LOD) finisca fuori dal bordo e non sia visibile.



figura 5.3: Screenshot di MotoGP 14 nel quale si può osservare la nuova stampa dei livelli di LOD per le moto

Si è poi provveduto ad esporre nel Beholder una variabile membro per ogni istanza della classe `LodSelectionModule` che permette di attivare e disattivare la stampa di del LOD attivo a run-time.



5.2. RIFATTORIZZAZIONE DELLA GERARCHIA DI STAMPA

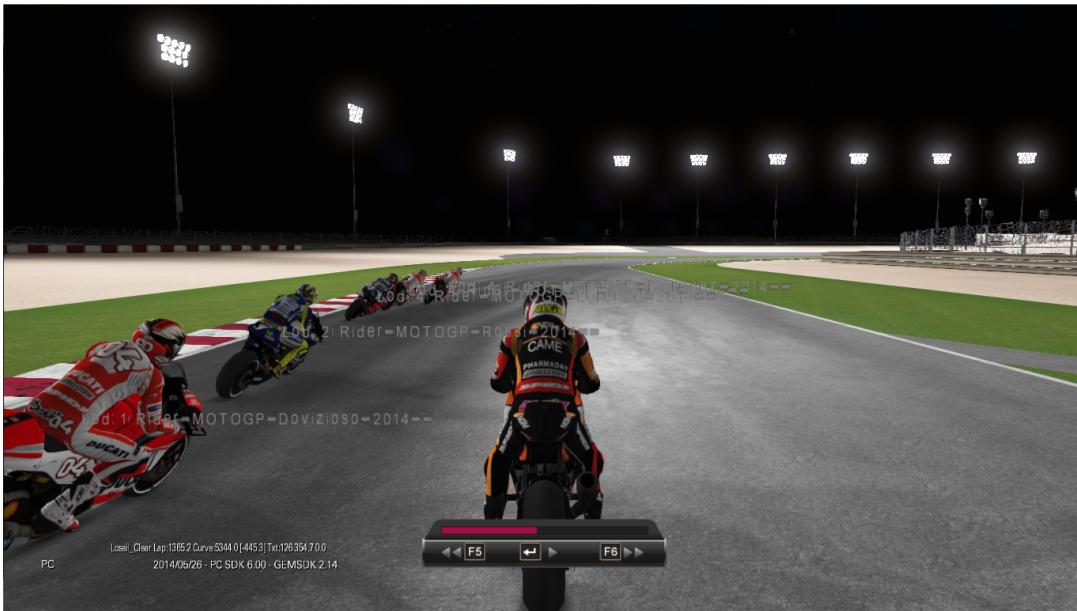


figura 5.4: Screenshot di MotoGP 14 nel quale si può osservare la nuova stampa dei livelli di LOD per i piloti

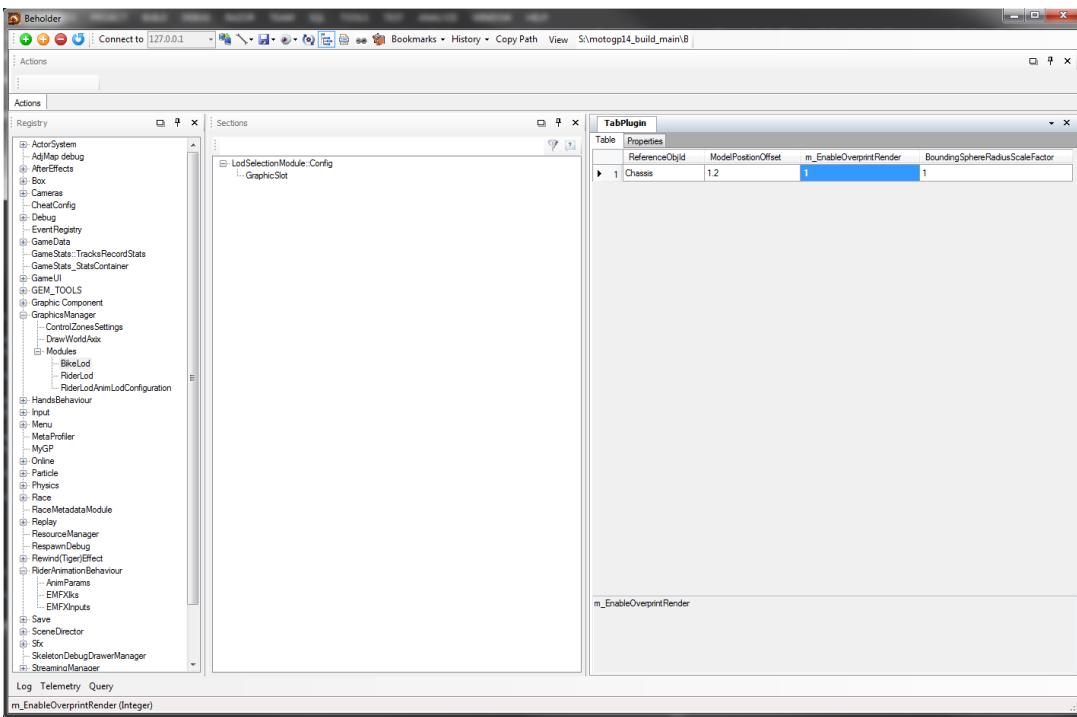


figura 5.5: Screenshot del Beholder nel quale è possibile vedere dove attivare e disattivare la stampa dei LOD delle moto

5.2 Rifattorizzazione della gerarchia di stampa

Prestando attenzione al metodo `PrintLodDebugInformation` presente nel listato 5.2 si può notare che la stampa a video della stringa è affidata alla classe singleton `DebugTextDrawer`. Inizialmente la stampa era affidata in gestione alla classe `LodModule-Debug`, la quale aveva un nome completamente diverso dal compito che svolgeva. Inda-



5.2. RIFATTORIZZAZIONE DELLA GERARCHIA DI STAMPA

gando sulle origini di tale classe si è scoperto che questa era stata originata dal “copia e incolla” della classe `GemScreenLogger`, la quale gestiva la stampa di stringhe in stile terminale, conservando e ristampando le ultime entry inserite e gestendo gli “a capo” e non permettendo sovrapposizioni del testo.

Avendo le due classi la maggior parte del codice in comune, si è provveduto ad effettuare un refactor creando una base astratta `UI_GemScreenTextBase` dalla quale entrambe derivano e concretizzano implementando i metodi astratti. La classe `LodModuleDebug` è stata inoltre rinominata in `DebugTextDrawer` per meglio rispecchiare le sue funzionalità e rendere il codice più comprensibile.

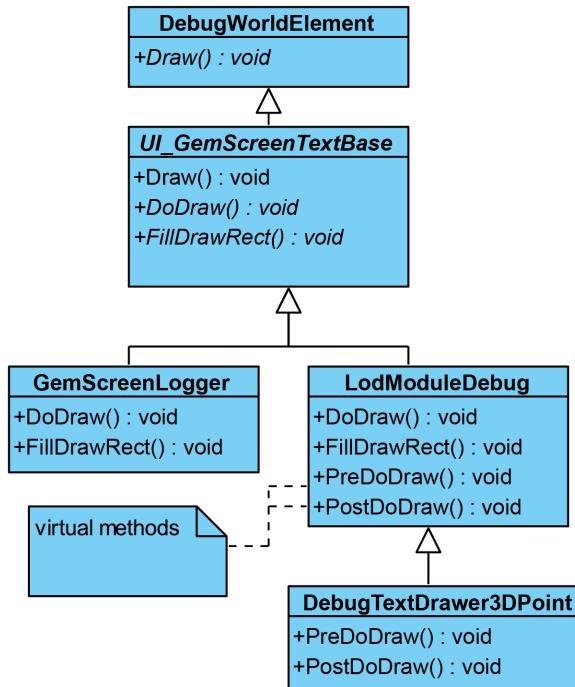


figura 5.6: Diagramma UML della gerarchia di stampa del testo. Sono riportati solo i metodi più significativi

La classe `DebugTextDrawer3DPoint` verrà trattata successivamente nella sezione 5.3.

Le differenze fra le due classi si mostra anche nei loro scenari di utilizzo. `GemScreenLogger` accetta dei log e continua a stamparli indipendentemente dai frame. `DebugTextDrawer` invece, ad ogni frame, stampa tutte le stringhe che possiede e le scarta, non stampandole più al frame successivo. Questo impone quindi che l'inserimento delle scritte sia costante e presente ad ogni frame. Questa caratteristica si adatta perfettamente alla stampa dei LOD, in quanto la posizione su schermo e la stessa informazione varia ad ogni frame e necessita di un costante aggiornamento.

Come si nota dal diagramma (link diagramma) la gerarchia parte da `DebugWorldElement`, classe che rappresenta uno scene actor che può essere inserito in un world. I world sono dei contenitori di oggetti 3D che procedono, se attivi, ad ogni frame a disegnare ogni oggetto presente in essi.

Le istanze delle classi concrete della gerarchia vengono, durante la fasi di inizializzazione, inserite nel mondo di debug. Esso è stato creato appositamente per contenere oggetti di debug. Nel resto del gioco, sono presenti implementazioni più



5.3. STAMPA DEL NOME DEGLI SPAZI DI RIFERIMENTO

specifiche di world, le quali permettono quindi di fare assunzioni sugli oggetti presenti e conseguentemente implementare forti ottimizzazioni. In MotoGP 14 ad esempio sono presenti il `MenuWorld` ed il `GameWorld`, che gestiscono rispettivamente il mondo visibile nel menu di gioco ed il mondo visibile durante una gara, permettendone un disegno efficiente.

Di seguito è presente la sezione di codice che inizializza gli oggetti della gerarchia in fase di avvio del gioco:

Listing 5.3: Inizializzazione delle classi responsabili della stampa di testo

```
GemScreenLogger::Create();
debugWorld->AddElement( UI::GemScreenLogger::GetInstance() );

DebugTextDrawer::Create();
DebugTextDrawer::Config config;
config.m_fontName = "Linotype Univers 430 Regular20";
config.m_alphaValue = 255;
config.m_timeBeforeFade = 500;
config.m_includeContext = false;
config.m_includeSourceFile = false;
config.m_timestamp = false;

DebugTextDrawer::GetInstance()->Init();
DebugTextDrawer::GetInstance()->SetConfig( config );
debugWorld->AddElement( DebugTextDrawer::GetInstance() );

DebugTextDrawer3DPoint::Create();
DebugTextDrawer3DPoint::GetInstance()->Init();
config.m_timeBeforeFade = 999999999;
DebugTextDrawer3DPoint::GetInstance()->SetConfig( config );
debugWorld->AddElement( DebugTextDrawer3DPoint::GetInstance() );
```

5.3 Stampa del nome degli spazi di riferimento

Una funzionalità già presente è la stampa degli assi di riferimento degli oggetti presenti nella scena 3D e dell'asse origine del mondo. L'obbiettivo quindi è stato l'inserimento della stampa del nome dell'oggetto affianco al disegno dell'asse allo scopo di poterli riconoscere fra loro quando se ne disegnano più d'uno. Per realizzare questa funzionalità si è proceduto alla creazione della classe `DebugTextDrawer3DPoint`. Essa deriva da `DebugTextDrawer` della quale specializza semplicemente l'inserimento della stringa, allo scopo di poter continuare a disegnare il testo nella corretta posizione (che può cambiare) senza dover chiamare il disegno della stringa ogni frame. Il disegno degli assi è attivato dal Beholder da diversi contesti.

`DebugTextDrawer` provvede ad ogni frame, tramite un puntatore alla camera attiva e una reference alla matrice della posizione, che può quindi essere aggiornata, ad effettuare il calcolo del punto 3D a partire dal punto 2D e lanciare la stampa ad ogni frame. L'appesa di un log a questa classe richiede quindi della camera attiva, del testo e della matrice che contiene la posizione 3D. In questo modo è possibile disegnare il nome dell'asse, e continuare a visualizzare il nome ad ogni frame, senza dover appendere il log ad ogni frame come con `DebugTextDrawer`. Questo si è reso necessario in quanto il submit degli oggetti 3D che rappresentano gli assi avviene una volta soltanto, rendendo quindi necessaria l'utilizzo di un meccanismo simile anche per il testo correlato.



5.3. STAMPA DEL NOME DEGLI SPAZI DI RIFERIMENTO

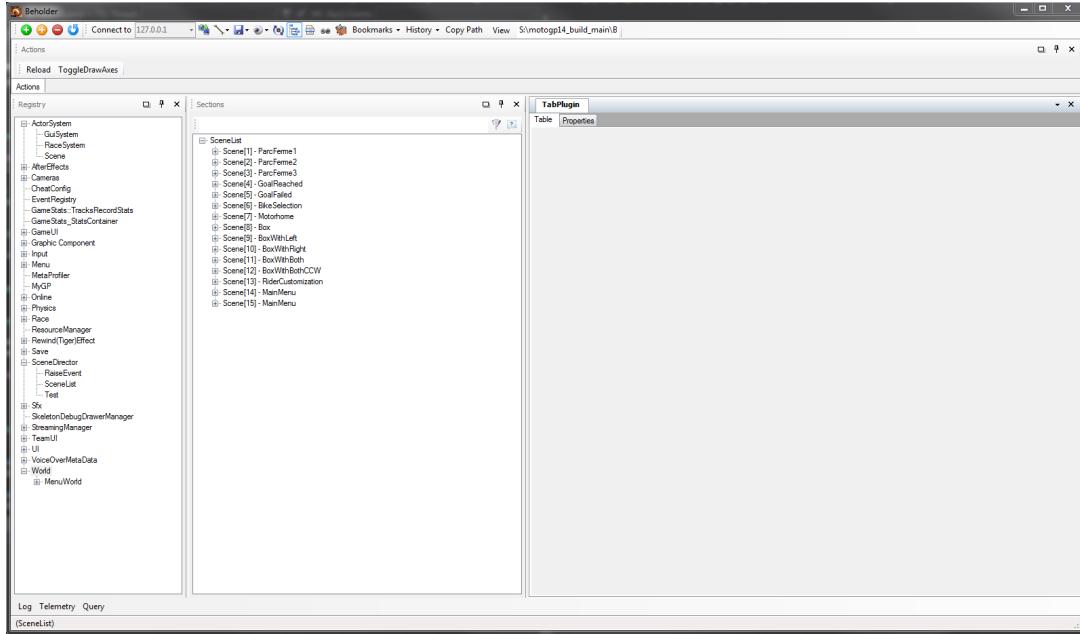


figura 5.7: Screenshot del Beholder nel quale è possibile lanciare il disegno degli assi tramite l'azione “ToggleDraeAxes” dello “SceneDirector” visibile in alto a sinistra

Lo studente ha provveduto quindi a individuare i punti del codice in cui erano presenti le chiamate di disegno degli assi allo scopo di implementare la nuova funzionalità, scoprendo che il codice per disegnarli era duplicato molteplici volte.

Si è proceduto quindi ad eseguire un refactor creando una classe specializzata nel disegno di assi con l'etichetta del nome dell'oggetto, la classe `DebugAxisDrawer`.

Essa provvede a incapsulare la funzione che si occupa del disegno degli assi e, eseguendo il codice necessario a ottenere la telecamera attiva, a lanciare la stampa del nome dell'oggetto. Come accennato, la funzionalità di disegno degli assi è lanciata durante un particolare stato in cui le geometrie disegnate, una volta eseguito il submit, vengono renderizzate ogni frame senza doverne richiedere la rappresentazione ad ogni frame. Si procede quindi allo stesso modo per il testo associato grazie alla classe `DebugTextDrawer3DPoint`.

Di seguito si riporta il codice della classe `DebugAxisDrawer`.

Listing 5.4: `DebugAxisDrawer.h`

```
#ifndef DEBUG_AXIS_DRAWER_H
#define DEBUG_AXIS_DRAWER_H

class DebugAxisDrawer
{
private:
    static DebugAxisDrawer* s_pInstance;

    const GMatrix m_matrix;

    vector<GMatrix> m_matrixCollection;

protected:
```



5.3. STAMPA DEL NOME DEGLI SPAZI DI RIFERIMENTO

```
DebugAxisDrawer();

public:

    static DebugAxisDrawer* GetInstance();

    void DrawAxis(CEngine* i_pEngine, const GMatrix& i_matrixPos,
                  const GString& i_text, GUIInt32 i_radius, GUIInt32 i_scale =
                  1);

    inline const GMatrix& GetIdentityMatrix();

};

inline const GMatrix& DebugAxisDrawer::GetIdentityMatrix()
{
    return m_matrix;
}

#endif
```

Listing 5.5: DebugAxisDrawer.cpp

```
#include "DebugAxisDrawer.h"

DebugAxisDrawer* DebugAxisDrawer::s_pInstance(NULL);

DebugAxisDrawer::DebugAxisDrawer():
    m_matrix(GMatrix::IDENTITY)
{
}

DebugAxisDrawer* DebugAxisDrawer::GetInstance()
{
    if(s_pInstance != NULL)
    {
        s_pInstance = new DebugAxisDrawer();
    }

    return s_pInstance;
}

void DebugAxisDrawer::DrawAxis(CEngine* i_pEngine, const GMatrix&
    i_matrixPos, const GString& i_text, GUIInt32 i_radius, GUIInt32
    i_scale)
{
    bool isMatrixToStore = &i_matrixPos == &m_matrix && i_scale !=
    1;

    //Resolve the flickering of the Word origin name
    if(isMatrixToStore)
    {
        m_matrixCollection.push_back(const_cast<GMatrix&>(
            i_matrixPos));
        m_matrixCollection.back().ApplyScale(GVector3(i_scale,
            i_scale, i_scale));
    }
}

static const GColor RED(255,0,0,255);
```



5.4. VISTA GERARCHICA DEI WORLD NEL BEHOLDER

```
static const GColor GREEN(0,255,0,255);
static const GColor BLUE(0,0,255,255);

//Draw the axis
CML_DrawVector::DrawVector(i_matrixPos.GetAxis(), i_matrixPos.
    GetTranslation(), RED, i_radius);
CML_DrawVector::DrawVector(i_matrixPos.GetYAxis(), i_matrixPos.
    GetTranslation(), GREEN, i_radius);
CML_DrawVector::DrawVector(i_matrixPos.GetAxis(), i_matrixPos.
    GetTranslation(), BLUE, i_radius);

//Draw the object name near the axis
CWorldManager& worldManager = i_pEngine->GetWorldManager();
for( CWorld*& const* iter = worldManager.begin(); iter !=
    worldManager.end(); ++iter)
{
    const CWorld& world = **iter;

    if( (world.GetClassId() == CLSID_MENUWORLD || world.
        GetClassId() == CLSID_GAMEWORLD)
        &&
        (world.GetEnabled() == eWORLD_ENABLE_RENDER || world.
            GetEnabled() == eWORLD_ENABLE_ALL) )
    {
        for (GUInt camIdx = 0; camIdx < world.GetNumberOfCameras
            (); ++camIdx)
        {
            if(isMatrixToStore)
            {
                DebugTextDrawer3DPoint::GetInstance()->AppendLog(i_text.
                    c_str(), world.GetCurrentCamera(camIdx), m_matrixCollection.back
                    ());
            }
            else
            {
                DebugTextDrawer3DPoint::GetInstance()->AppendLog(i_text.
                    c_str(), world.GetCurrentCamera(camIdx), i_matrixPos);
            }
        }
    }
}
```

5.4 Vista gerarchica dei world nel Beholder

Nel Beholder è possibile visualizzare tutte le risorse grafiche presenti nel ResourcesManager. Questa classe gestisce il caricamento e lo scaricamento in memoria di tutte le risorse grafiche utilizzate per costruire le scene 3D del gioco. Essa utilizza la strategia di reference counting per sapere quando un oggetto diviene inutilizzato e può essere quindi scaricato. Essa contiene quindi una collezione di world in cui sono presenti le varie risorse (mesh, texture, animazioni, ecc..). Per ogni tipo di risorsa presente nel world nel Beholder viene presentata come lista piatta. È possibile invece che queste risorse siano imparentate fra di loro, formando ad esempio delle gerarchie di oggetti 3D.

Si è proceduto quindi a creare una versione ad albero per la categoria degli Object3D presenti nel world.



5.4. VISTA GERARCHICA DEI WORLD NEL BEHOLDER

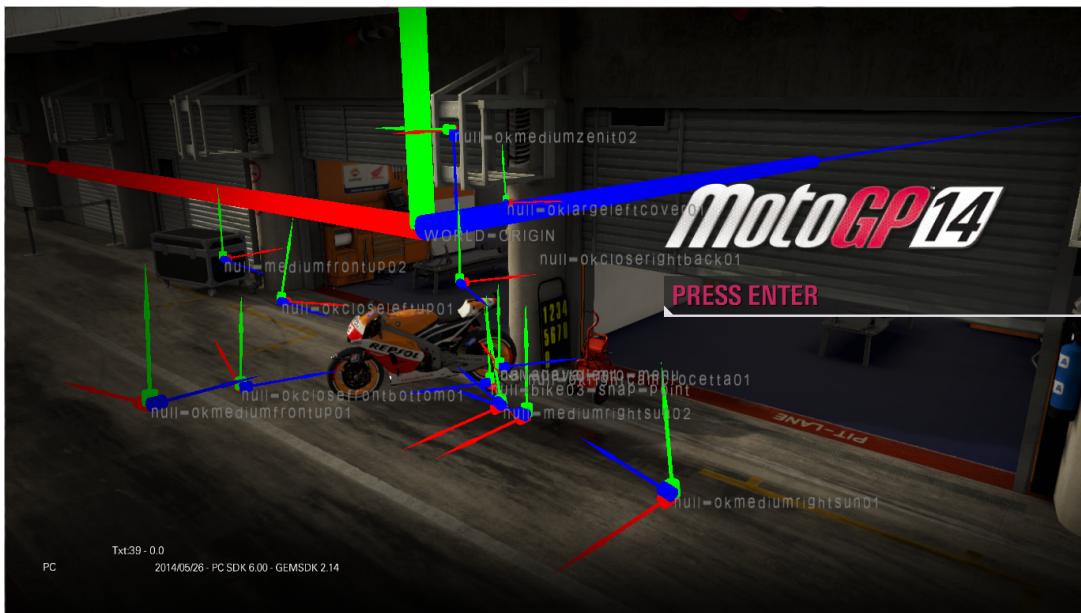


figura 5.8: Screenshot di MotoGP 14 in cui nel main menu è possibile vedere gli assi dei punti in cui viene ancorata la camera e la stampa del nome corrispondente

Il contenuto del `ResourcesManager` è esposto nel Beholder tramite la classe `ResourcesManagerDebug` in cui si concentrano le modifiche effettuate per poter offrire la nuova funzionalità.

Di seguito vengono riportati alcune sezioni del codice della classe significative per la nuova funzionalità inserita.

Listing 5.6: ResourcesManagerDebug.h

```
#ifndef RESOURCES_MANAGER_DEBUG
#define RESOURCES_MANAGER_DEBUG

class ResourcesManagerDebug
{
public:
    ENABLE_METADATA;

    ResourcesManagerDebug();
    ~ResourcesManagerDebug();

    GBool Init(ResourcesManager* i_pResourcesManager);
    GBool Uninit();

    void Update() const;

    GBool IsInitialized() const;

    static GBool ResourcesManagerDebugEnabled();

private:
    void RegisterToBeholder();
    void UnregisterFromBeholder();

    void UpdateGroupResources() const;
}
```



5.4. VISTA GERARCHICA DEI WORLD NEL BEHOLDER

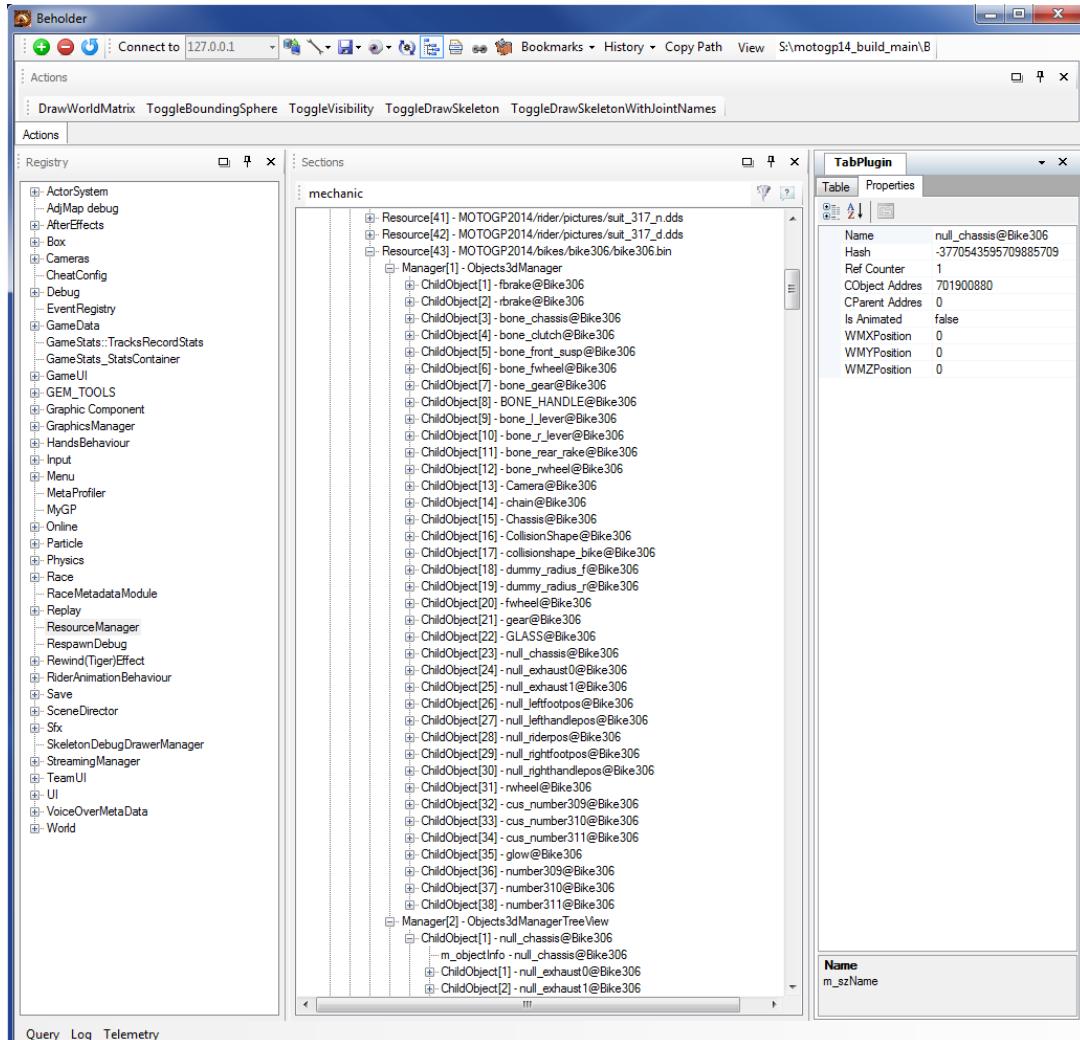


figura 5.9: Screenshot del Beholder in cui è possibile visualizzare la versione a lista piatta sopra e la versione ad albero sotto per la bike 306

```
ResourcesManagerDebug(const ResourcesManagerDebug& i_that);
ResourcesManagerDebug& operator=(const ResourcesManagerDebug&
    i_that);

private:

    static const FixedString64 s_szMetadataName;

public:

    struct ObjectInfo
    {
        ENABLE_METADATA;

        ObjectInfo();

        GUIInt64      m_uiHash;
        char          m_szName[256];
        uint_t        m_uiRefCount;
```



5.4. VISTA GERARCHICA DEI WORLD NEL BEHOLDER

```
    GUIInt64      m_uiCObjectAddres;
    GUIInt64      m_uiCParentAddres;

    void DrawWorldMatrix();
    void ToggleBoundingSphere();
    void ToggleVisibility();
    void ToggleDrawSkeleton();
    void ToggleDrawSkeletonWithJointNames();

    void ToggleFlag(uint32_t flag);

    void RealDrawSkeleton(GBool i_bDrawJointNames);

    float m_WMXPosition;
    float m_WMYPosition;
    float m_WMZPosition;

    GUIInt32 m_uiAxesScale;
    GReal    m_fAxesRadius;
    GBool    m_bIsAnimated;
};

private:

    struct ObjectInfoContainer
    {
        ENABLE_METADATA;

        char          m_szName[256];
        ObjectInfo    m_objectInfo;
        vector<ObjectInfoContainer> m_aListOfChildObjects;
    };

    struct ManagerInfo
    {
        ENABLE_METADATA;

        char          m_szManagerName[256];
        vector<ObjectInfoContainer> m_aListOfObjects;
    };

    struct ResourceInfo
    {
        ResourceInfo() { }
        ResourceInfo(const GString& i_szFileNameBin, const CWorld*
                     i_pWorld, GBool i_bIsHierarchicalViewEnabled);

        ENABLE_METADATA;

        char          m_szFileNameBin[256];
        GUIInt       m_uiRefCount;
        GUIInt64     m_uiWorldAddres;
        ManagerInfo  m_managers[7];
    };

    template<typename T>
    static void FillObjectInfoContainer(ObjectInfoContainer&
                                         i_infoContainer, T* i_pRes);

    static void FillObjectInfoContainer(ObjectInfoContainer&
```



5.4. VISTA GERARCHICA DEI WORLD NEL BEHOLDER

```
i_infoContainer, CObject3d* i_pRes);

template<typename T>
static void GeneralFillObjectInfoContainer(ObjectInfoContainer&
    i_infoContainer, T* i_pRes);

template<typename T>
static void FillResourcesHelper(const TManager<T>& i_manager,
    vector<ResourcesManagerDebug::ObjectInfoContainer>&
    o_aListOfResources);

static void FillResourcesHelperAsTree(const CObjects3dManager&
    i_manager, vector<ResourcesManagerDebug::ObjectInfoContainer
    >& o_aListOfRootContainer);

struct ResourcesGroup
{
    ENABLE_METADATA;

    char m_szGroupName[256];
    vector<ResourceInfo> m_aListOfResources;
};

struct DebugData
{
    ENABLE_METADATA;

    vector<ResourcesGroup> m_aListOfGroupResources;
};

ResourcesManager* m_pResourcesManager;

GBool m_bBeholderRegistered;
GBool m_bInitialized;
GBool m_bIsHierarchicalViewEnabled;

mutable CMutex m_oMutex;
mutable DebugData m_oDebugData;

const DebugData& UpdateResources() const { Update(); return
    m_oDebugData; }

public:
    void ReloadDescriptors();

    void ToggleHierarchicalView();
};

inline GBool ResourcesManagerDebug::IsInitialized() const
{
    CMutexAutoLock oMutexAutoLock(m_oMutex);
    return m_bInitialized;
}

#endif
```



5.4. VISTA GERARCHICA DEI WORLD NEL BEHOLDER

Listing 5.7: ResourcesManagerDebug.cpp

```
template<typename T>
void ResourcesManagerDebug::FillObjectInfoContainer(
    ObjectInfoContainer& i_infoContainer, T* i_pRes)
{
    GeneralFillObjectInfoContainer(i_infoContainer, i_pRes);

    i_infoContainer.m_objectInfo.m_WMXPosition = 0.0f;
    i_infoContainer.m_objectInfo.m_WMYPosition = 0.0f;
    i_infoContainer.m_objectInfo.m_WMZPosition = 0.0f;

    i_infoContainer.m_objectInfo.m_bIsAnimated = GFALSE;
}

void ResourcesManagerDebug::FillObjectInfoContainer(
    ObjectInfoContainer& i_infoContainer, CObject3d* i_pRes)
{
    GeneralFillObjectInfoContainer(i_infoContainer, i_pRes);

    CObject3d* pObj = (CObject3d*)(uintptr_t)(i_infoContainer.
        m_objectInfo.m_uiCObjectAddres);
    i_infoContainer.m_objectInfo.m_uiCParentAddres = (GUlnt64)(pObj
        ->GetParent());

    const GVector3& objectWordPosition = pObj->GetWorldMatrix().
        GetTranslation();
    i_infoContainer.m_objectInfo.m_WMXPosition = objectWordPosition.
        x;
    i_infoContainer.m_objectInfo.m_WMYPosition = objectWordPosition.
        y;
    i_infoContainer.m_objectInfo.m_WMZPosition = objectWordPosition.
        z;

    i_infoContainer.m_objectInfo.m_bIsAnimated = pObj->GetClassId()
        == CLSID_ANIMATEDMODEL3D;
}

template<typename T>
void ResourcesManagerDebug::GeneralFillObjectInfoContainer(
    ObjectInfoContainer& i_infoContainer, T* i_pRes)
{
    sprintf_s(i_infoContainer.m_szName, 256, "%s", i_pRes->GetId().
        DebugGetName().c_str());
    sprintf_s(i_infoContainer.m_objectInfo.m_szName, 256, "%s",
        i_pRes->GetId().DebugGetName().c_str());
    i_infoContainer.m_objectInfo.m_uiHash = i_pRes->GetId().
        GetNameHash();
    i_infoContainer.m_objectInfo.m_uiCObjectAddres =
        reinterpret_cast<GUlnt64>(i_pRes);
    i_infoContainer.m_objectInfo.m_uiRefCounter = i_pRes->
        GetRefCounter();
}

template<typename T>
void ResourcesManagerDebug::FillResourcesHelper(const TManager<T>&
    i_manager, vector<ResourcesManagerDebug::ObjectInfoContainer>&
    o_aListOfResources)
{
    size_t uiResNum = i_manager.size();
```



5.4. VISTA GERARCHICA DEI WORLD NEL BEHOLDER

```
o_aListOfResources.clear();
o_aListOfResources.reserve(uiResNum);

for(GUInt uiIndex = 0; uiIndex < uiResNum; ++uiIndex)
{
    ObjectInfoContainer new_infoContainer;
    FillObjectInfoContainer(new_infoContainer, i_manager.at(
        uiIndex));

    o_aListOfResources.push_back(new_infoContainer);
}
}

void ResourcesManagerDebug::FillResourcesHelperAsTree(const
    CObjects3dManager& i_manager, vector<ResourcesManagerDebug::
    ObjectInfoContainer>& o_aListofRootContainer)
{
    const size_t uiResNum = i_manager.size();

    o_aListofRootContainer.clear();
    o_aListofRootContainer.reserve(uiResNum);

    vector<GBool> inserted;
    inserted.resize(uiResNum, GFALSE);

    GBool bFinished = GFALSE;

    while (!bFinished)
    {
        bFinished = GTRUE;

        for(GUInt uiIndex = 0; uiIndex < uiResNum; ++uiIndex)
        {
            if( !inserted[uiIndex] )
            {
                ObjectInfoContainer new_infoContainer;
                FillObjectInfoContainer(new_infoContainer, i_manager
                    .at(uiIndex));

                //If it's a parent object
                if(new_infoContainer.m_objectInfo.m_uiCParentAddres
                    == 0)
                {
                    o_aListofRootContainer.push_back(
                        new_infoContainer);
                    inserted[uiIndex] = GTRUE;
                }
                else
                {
                    //Search the parent object where append this
                    //node
                    for(GUInt i = 0; i < o_aListofRootContainer.size
                        (); ++i)
                    {
                        if(o_aListofRootContainer[i].m_objectInfo.
                            m_uiCObjectAddres == new_infoContainer.
                            m_objectInfo.m_uiCParentAddres)
                        {
                            o_aListofRootContainer[i].
```



5.4. VISTA GERARCHICA DEI WORLD NEL BEHOLDER

```
        m_aListOfChildObjects.push_back(
            new_infoContainer);
        inserted[uiIndex] = GTRUE;
        bFinished = GFALSE;
        break;
    }

}

}

}

}

//Add the spare (root) elements that hasn't the parent pointer =
0
for(GUInt uiIndex = 0; uiIndex < uiResNum; ++uiIndex)
{
    if( !inserted[uiIndex] )
    {
        ObjectInfoContainer new_infoContainer;
        FillObjectInfoContainer(new_infoContainer, i_manager.at(
            uiIndex));

        GBool bIsRoot = GTRUE;
        for(GUInt i = 0; i < o_aListOfRootContainer.size(); ++i)
        {
            if(o_aListOfRootContainer[i].m_objectInfo.
                m_uiCObjectAddres == new_infoContainer.
                m_objectInfo.m_uiCParentAddres)
            {
                o_aListOfRootContainer[i].m_aListOfChildObjects.
                    push_back(new_infoContainer);
                bIsRoot = GFALSE;
                break;
            }
        }

        if(bIsRoot)
        {
            o_aListOfRootContainer.push_back(new_infoContainer);
        }
    }
}

ResourcesManagerDebug::ResourceInfo::ResourceInfo(const GString&
    i_szFileNameBin, const CWorld* i_pWorld, GBool
    i_bIsHierarchicalViewEnabled )
{
    sprintf_s(m_szFileNameBin, 256, "%s", i_szFileNameBin.c_str());
    m_uiWorldAddres = reinterpret_cast<GUInt64>(i_pWorld);
    m_uiRefCounter = int_cast<GUInt>(i_pWorld->GetRefCounter())-1;

    sprintf_s(m_managers[0].m_szManagerName, 256, "%s", "
        Objects3dManager");
    sprintf_s(m_managers[1].m_szManagerName, 256, "%s", "
        Objects3dManagerTreeView");
    sprintf_s(m_managers[2].m_szManagerName, 256, "%s", "MeshManager
        ");
```



5.4. VISTA GERARCHICA DEI WORLD NEL BEHOLDER

```
sprintf_s(m_managers[3].m_szManagerName, 256, "%s", "MaterialManager");
sprintf_s(m_managers[4].m_szManagerName, 256, "%s", "TextureManager");
sprintf_s(m_managers[5].m_szManagerName, 256, "%s", "AnimationManager");
sprintf_s(m_managers[6].m_szManagerName, 256, "%s", "AnimatorManager");

FillResourcesHelper(i_pWorld->GetObjects3dManager(),
    m_managers[0].m_aListOfObjects);
if (i_bIsHierarchicalViewEnabled)
{
    FillResourcesHelperAsTree(i_pWorld->GetObjects3dManager(),
        m_managers[1].m_aListOfObjects);
}
FillResourcesHelper(i_pWorld->GetMeshManager(),
    m_managers[2].m_aListOfObjects);
FillResourcesHelper(i_pWorld->GetMaterialManager(),
    m_managers[3].m_aListOfObjects);
FillResourcesHelper(i_pWorld->GetTextureManager(),
    m_managers[4].m_aListOfObjects);
FillResourcesHelper(i_pWorld->GetAnimationManager(),
    m_managers[5].m_aListOfObjects);
FillResourcesHelper(i_pWorld->GetAnimatorManager(),
    m_managers[6].m_aListOfObjects);
}

void ResourcesManagerDebug::ToggleHierarchicalView()
{
    m_bIsHierarchicalViewEnabled = !m_bIsHierarchicalViewEnabled;

    Update();
}

ResourcesManagerDebug::ObjectInfo::ObjectInfo():
    m_uiAxesScale(1.0f),
    m_fAxesRadius(0.05f),
    m_bIsAnimated(FALSE)
{
    m_WMXPosition = 0;
    m_WMYPosition = 0;
    m_WMZPosition = 0;
}

void ResourcesManagerDebug::ObjectInfo::DrawWorldMatrix()
{
    CObject3d* pObj = (CObject3d*)(uintptr_t)(m_uiCObjectAddres);

    const GMatrix& matrix = pObj->GetWorldMatrix();

    DebugAxisDrawer::GetInstance()->DrawAxis(g_pEngine, matrix,
        m_szName, m_fAxesRadius);
}

void ResourcesManagerDebug::ObjectInfo::ToggleFlag(uint32_t flag)
{
    CObject3d* pObj = (CObject3d*)(uintptr_t)(m_uiCObjectAddres);

    if(pObj->GetFlags() & flag)
```



5.4. VISTA GERARCHICA DEI WORLD NEL BEHOLDER

```
{  
    p0bj->SetFlags(0, (uint32_t)~flag);  
}  
else  
{  
    p0bj->SetFlags(flag);  
}  
}  
  
void ResourcesManagerDebug::ObjectInfo::ToggleDrawSkeleton()  
{  
    RealDrawSkeleton(false);  
}  
  
void ResourcesManagerDebug::ObjectInfo::  
    ToggleDrawSkeletonWithJointNames()  
{  
    RealDrawSkeleton(true);  
}  
  
void ResourcesManagerDebug::ObjectInfo::RealDrawSkeleton(GBool  
    i_bDrawJointNames)  
{  
    if(m_bIsAnimated)  
    {  
        GBool bInserted = SkeletonDebugDrawerManager::GetInstance()  
            ->AddObject((CAnimatedModel3d*)(uintptr_t)(  
                m_uiCObjectAddres), i_bDrawJointNames);  
  
        if( !bInserted )  
        {  
            SkeletonDebugDrawerManager::GetInstance()->RemoveObject  
                ((CAnimatedModel3d*)(uintptr_t)(m_uiCObjectAddres));  
        }  
    }  
}
```

Listing 5.8: ResourcesManagerDebug_Metadata.cpp

```
START_ACTIONS(ResourcesManagerDebug::ObjectInfo)  
    ACTION("DrawWorldMatrix", &ResourcesManagerDebug::ObjectInfo::  
        DrawWorldMatrix),  
    ACTION("ToggleBoundingSphere", &ResourcesManagerDebug::  
        ObjectInfo::ToggleBoundingSphere),  
    ACTION("ToggleVisibility", &ResourcesManagerDebug::ObjectInfo::  
        ToggleVisibility),  
    ACTION("ToggleDrawSkeleton", &ResourcesManagerDebug::ObjectInfo::  
        ::ToggleDrawSkeleton),  
    ACTION("ToggleDrawSkeletonWithJointNames", &  
        ResourcesManagerDebug::ObjectInfo::  
        ToggleDrawSkeletonWithJointNames)  
END_ACTIONS;  
  
START_ACTIONS(ResourcesManagerDebug)  
    ACTION("ReloadDescriptors", &ResourcesManagerDebug::  
        ReloadDescriptors),  
    ACTION("ToggleHierarchicalView", &ResourcesManagerDebug::  
        ToggleHierarchicalView),  
END_ACTIONS;
```



5.4. VISTA GERARCHICA DEI WORLD NEL BEHOLDER

```
// ResourceManagerDebug::Info's members Metadata definition
START_META("ResourceManager::Info",ResourcesManagerDebug::ObjectInfo
)
    META_PROPERTY(m_szName),
    META_PROPERTY(m_uiHash),
    META_PROPERTY(m_uiRefCounter),
    META_PROPERTY(m_uiCObjectAddres),
    META_PROPERTY(m_uiCParentAddres),
    META_PROPERTY(m_bIsAnimated),
    META_PROPERTY(m_WMXPosition),
    META_PROPERTY(m_WMYPosition),
    META_PROPERTY(m_WMZPosition),
END_META;

// ResourceManagerDebug::ResourceInfo's members Metadata definition
START_META("ResourceManager::ResourceInfo",ResourcesManagerDebug::
    ResourceInfo)
    META_PROPERTY(m_szFileNameBin),
    META_PROPERTY(m_uiRefCounter),
    META_PROPERTY(m_uiWorldAddres),
    META_NAMED_PROPERTY("Manager",m_managers)
END_META;

// ResourceManagerDebug::ResourcesGroup's members Metadata
// definition
START_META("ResourceManager::ResourcesGroup",ResourcesManagerDebug::
    ResourcesGroup)
    META_PROPERTY(m_szGroupName),
    META_NAMED_PROPERTY("Resource", m_aListOfResources)
END_META;

START_META("ResourceManager::DebugData",ResourcesManagerDebug::
    DebugData)
    META_NAMED_PROPERTY("Group", m_aListOfGroupResources)
END_META;

// ResourceManagerDebug::DebugData's members Metadata definition
START_META("ResourceManager",ResourcesManagerDebug)
    META_NAMED_GETTER("ResourcesGroup",UpdateResources),
    META_PROPERTY(m_oDebugData)
END_META;

// ResourceManagerDebug::ManagerInfo's members Metadata definition
START_META("ResourceManager::ManagerInfo",ResourcesManagerDebug::
    ManagerInfo)
    META_PROPERTY(m_szManagerName),
    META_NAMED_PROPERTY("ChildObject", m_aListOfObjects)
END_META;

// ResourceManagerDebug::ManagerInfo's members Metadata definition
START_META("ResourceManager::ObjectInfoContainer",
    ResourcesManagerDebug::ObjectInfoContainer)
    META_PROPERTY(m_szName),
    META_PROPERTY(m_objectInfo),
    META_NAMED_PROPERTY("ChildObject", m_aListOfChildObjects)
END_META;
```



5.5. DISEGNO DEGLI SCHELETRI DI ANIMAZIONE

5.5 Disegno degli scheletri di animazione

Allo scopo di permettere agli artisti di affinare le animazioni scheletrali dei personaggi e permettere al team 3D un più rapido debug, è stato deciso di aggiungere al gioco la funzionalità di disegno dello scheletro di animazione degli oggetti animati.

Come visibile in figura 5.10 lo scheletro di animazione è composto da un'insieme di bone (ossa) organizzati gerarchicamente: se il padre viene mosso, il movimento è trasmesso a tutta la prole. Un modo efficiente di memorizzare lo scheletro è tramite una paletta di matrici⁴ che rappresentano la posizione dei joint (estremi esterni di un bone) che compongono lo scheletro. È inoltre necessaria una lista che permetta di sapere come è organizzata la gerarchia dei joint per permettere la ricostruzione dello scheletro⁵.

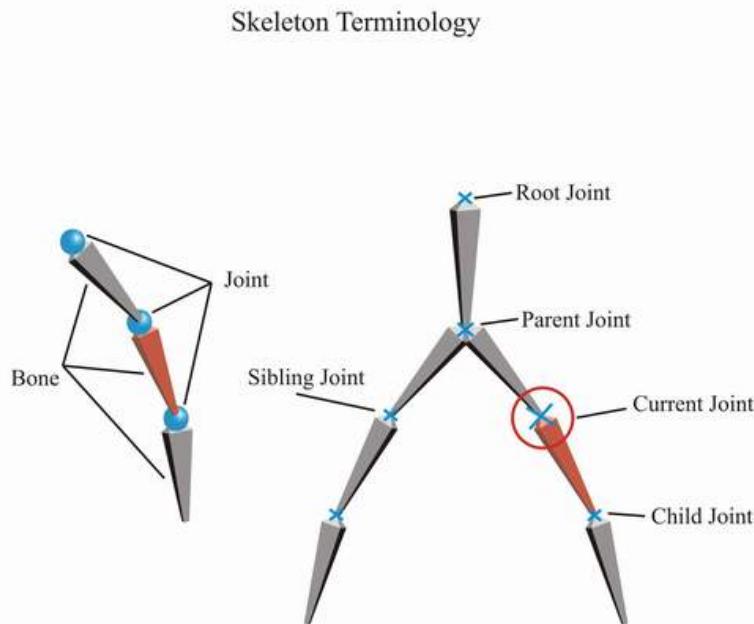


figura 5.10: Sintesi della terminologia usata nelle animazioni scheletrali. Tratto da [Sitf]

Si procede quindi ad animare⁶ lo scheletro di animazione tramite la fisica o con delle pose fisse precalcolate dagli artisti, oppure con il blend tree calcolato tramite il middleware Emotion Fx.

Il passo successivo è modificare la mesh dell'oggetto per rispecchiare l'animazione dello skeleton, questa procedura è detta Skinning. Un modo semplice per effettuarla è assegnare ad ogni vertice della mesh un massimo di 4 bone che potranno influenzarne la posizione. Si procede quindi (solitamente su GPU durante la fase di Vertex Shader) a calcolare un'interpolazione sulla trasformazione della posizione del vertice effettuando una media pesata fra i bone assegnati. La trasformazione risultante viene quindi

⁴Nella grafica 3D una matrice 4x4 è un modo efficiente di memorizzare una posizione oppure una trasformazione in termini di traslazione, rotazione e scale. Un approfondimento è disponibile in [Lun12, chp. 3]

⁵Esistono metodi di memorizzazione più efficienti utilizzando strutture dati avanzate ma, visto che solitamente lo scheletro viene sempre scorso linearmente e i bone vengono raramente rimossi o aggiunti, questa è una delle implementazioni più usate nella computer grafica.

⁶Si consiglia al lettore appassionato [Lun12, chp. 24, 25] in cui è presente una breve introduzione alla matematica dei quaternioni, l'animazione scheletrale e lo skinning.



5.5. DISEGNO DEGLI SCHELETRI DI ANIMAZIONE

applicata alla posizione originale del vertice.

L'analisi del codice di gioco ha evidenziato la presenza di una precedente implementazione del disegno dello skeleton dei soli rider durante la corsa. Questa implementazione permetteva l'attivazione e la disattivazione della funzionalità di disegno attraverso una `define`.

La prima modifica è stata l'esposizione nel Beholder di una variabile di membro di quella classe che permettesse di attivare/disattivare il disegno di debug a run-time.

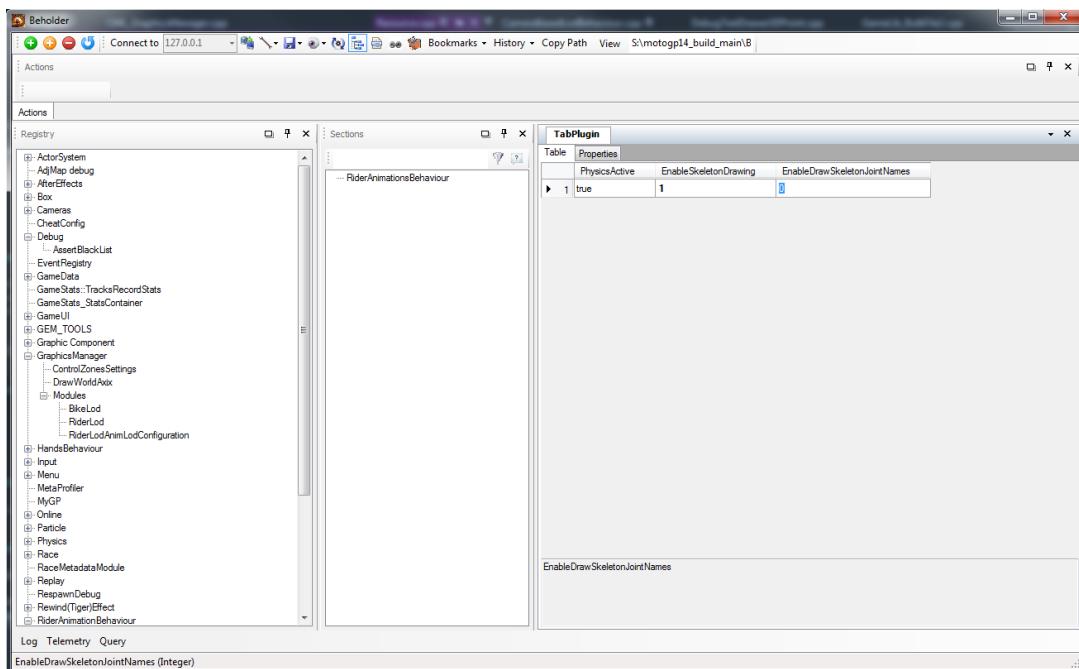


figura 5.11: Screenshot del Beholder in cui è possibile visualizzare dove è permesso attivare e disattivare il disegno dello skeleton del rider

Successivamente si è proceduto ad affinare e ottimizzare l'algoritmo di disegno, estrapolandolo e portandolo in una classe specializzata nel solo disegno di uno skeleton: `SkeletonDebugDrawer`.

Di seguito è presente il codice della classe `SkeletonDebugDrawer`.

Listing 5.9: SkeletonDebugDrawer.h

```
#ifndef SKELETON_DEBUG_DRAWER_H
#define SKELETON_DEBUG_DRAWER_H

class SkeletonDebugDrawer
{
private:

    static GColor GetJointColor( GBool bObjectIsInSkeleton, GVector3
        * i_ParentJointWorldPosition);

    SkeletonDebugDrawer(void);

public:
```



5.5. DISEGNO DEGLI SCHELETRI DI ANIMAZIONE

```
~SkeletonDebugDrawer(void);

static void DrawSkeleton(const CAnimatedModel3d*
    i_pAnimatedModel3d, const GUI32 i_drawJointNames);

static void DrawEMFXSkeleton(
    GBool i_bDrawJointNames,
    const GMATRIX& i_RootMatrix,
    CObject3d* i_pObj,
    const vector<GMATRIX>& i_SkinPalette,
    CSkeleton* i_Skeleton,
    GVector3* i_ParentJointWorldPosition = NULL);

inline static void DrawPoint(const GVector3& i_vPosition, const
    GUI32 i_uiRed, const GUI32 i_uiGreen, const GUI32
    i_uiBlue, const GReal i_fSize);
};

inline void SkeletonDebugDrawer::DrawPoint(const GVector3&
    i_vPosition, const GUI32 i_uiRed, const GUI32 i_uiGreen, const
    GUI32 i_uiBlue, const GReal i_fSize)
{
    CML_DrawVector::DrawPoint(i_vPosition, i_uiRed, i_uiBlue,
        i_uiBlue, i_fSize);
}

#endif
```

Listing 5.10: SkeletonDebugDrawer.cpp

```
#include "SkeletonDebugDrawer.h"

SkeletonDebugDrawer::SkeletonDebugDrawer(void)
{

}

SkeletonDebugDrawer::~SkeletonDebugDrawer(void)
{

}

GColor SkeletonDebugDrawer::GetJointColor( GBool bObjectIsInSkeleton
    , GVector3* i_ParentJointWorldPosition)
{
    static const GColor FUCSIA(255, 0, 255, 255);
    static const GColor GREEN(0, 255, 0, 255);
    static const GColor RED(255, 0, 0, 255);

    GColor jointColor;

    if(bObjectIsInSkeleton)
    {
        if(i_ParentJointWorldPosition == NULL)
        {
            jointColor = GREEN;
        }
        else
        {
            jointColor = FUCSIA;
        }
    }
}
```



5.5. DISEGNO DEGLI SCHELETRI DI ANIMAZIONE

```
        }
```

```
    else
```

```
    {
```

```
        jointColor = RED;
```

```
    }
```

```
    return jointColor;
}
```

```
void SkeletonDebugDrawer::DrawSkeleton(const CAnimatedModel3d*
i_pAnimatedModel3d, const GUIInt32 i_drawJointNames)
{
    const CObject3d* animatedObject3D = i_pAnimatedModel3d;
```

```
    const GMatrix& objectWorldMatrix = animatedObject3D->
        GetWorldMatrix();
```

```
    SkinController *skinCtrl = animatedObject3D->GetControllerById<
        SkinController>(CLSID_SKINCONTROLLER);
    CSkeleton* skeleton = skinCtrl->GetSkeleton(1);
    if(skeleton == NULL)
    {
        skeleton = skinCtrl->GetSkeleton(0);
    }
    GEM_ASSERT(skeleton);
```

```
    BufferController* controller = animatedObject3D->
        GetControllerById<BufferController>(
            g_uiCLSID_BUFFER_CONTROLLER);
    const vector<GMatrix*>* jointsMatrices = controller->
        GetSkinMatrix();
```

```
    CObject3d* p0bj = skeleton->GetRootJoint(0);
    if (p0bj && jointsMatrices)
    {
        SkeletonDebugDrawer::DrawEMFXSkeleton(i_drawJointNames,
            objectWorldMatrix, p0bj, *jointsMatrices, skeleton);
    }
```

```
    //Draw of the object position
    CML_DrawVector::DrawPoint(objectWorldMatrix.GetTranslation(),
        GColor(255,0,0,255), 0.02);
}
```

```
void SkeletonDebugDrawer::DrawEMFXSkeleton(
    GBool i_bDrawJointNames,
    const GMatrix& i_RootMatrix,
    CObject3d* i_p0bj,
    const vector<GMatrix*>& i_SkinPalette,
    CSkeleton* i_Skeleton,
    GVector3* i_ParentJointWorldPosition)
{
    GEM_ASSERT(i_p0bj != NULL);

    //dato l'object3d mi da l'index alla posizione nel vettore dello
    //scheletro
    GInt iJointIndex = vector_signed_index(i_Skeleton->GetJointsList
        (), i_p0bj);

    GBool bObjectIsInSkeleton = (iJointIndex >= 0);
```



5.5. DISEGNO DEGLI SCHELETRI DI ANIMAZIONE

```
GVector3 jointWorldPosition;
GMatrix jointWorldMatrix;
if (bObjectIsInSkeleton)
{
    jointWorldMatrix = RiderAnimationsBehaviour::
        GetJointLocalMatrix(i_Skeleton, iJointIndex,
        i_SkinPalette[iJointIndex]);
}
else
{
    jointWorldMatrix = i_pObj->GetWorldMatrix();
}

jointWorldPosition = (jointWorldMatrix * i_RootMatrix).
    GetTranslation();
GColor jointColor = GetJointColor(bObjectIsInSkeleton,
    i_ParentJointWorldPosition);

const GReal POINT_SIZE = 0.05f;
CML_DrawVector::DrawPoint(jointWorldPosition, jointColor,
    POINT_SIZE);

if(i_bDrawJointNames)
{
    CWorldManager worldManager = g_pEngine->GetWorldManager();

    for( CWorld*const* iter = worldManager.begin(); iter != 
        worldManager.end(); ++iter)
    {
        const CWorld& world = **iter;

        if( (world.GetClassId() == CLSID_MENUWORLD || world.
            GetClassId() == CLSID_GAMEWORLD)
            &&
            (world.GetEnabled() == eWORLD_ENABLE_RENDER || world
                .GetEnabled() == eWORLD_ENABLE_ALL) )
        {
            for (GUInt camIdx = 0; camIdx < world.
                GetNumberOfCameras(); ++camIdx)
            {
                GVector2 screenPosition;
                world.GetCurrentCamera(camIdx)->ProjectPoint(
                    jointWorldPosition, screenPosition);

                const char* name = i_pObj->GetId().DebugGetName
                    () .c_str();
                if(name)
                {
                    GraphicModule::DebugTextDrawer::GetInstance
                        () ->AppendLog(name, screenPosition.x,
                            screenPosition.y);
                }
            }
        }
    }

    if (i_ParentJointWorldPosition)
    {
```



5.5. DISEGNO DEGLI SCHELETRI DI ANIMAZIONE

```
static const GColor YELLOW(255,255,0,255);
static const GReal SEGMENT_SIZE = 0.02f;
CML_DrawVector::DrawSegment(jointWorldPosition, *
    i_ParentJointWorldPosition, YELLOW, SEGMENT_SIZE);
}

CObject3d::Children::const_iterator childIt = i_pObj->
    GetChildren().begin();
CObject3d::Children::const_iterator childEnd = i_pObj->
    GetChildren().end();

while (childIt != childEnd)
{
    //Recursive call
    DrawEMFSkeleton(i_bDrawJointNames, i_RootMatrix, *childIt,
        i_SkinPalette, i_Skeleton, &jointWorldPosition);
    childIt++;
}
}
```

La rappresentazione interna dello scheletro organizzata come spiegato prima, permette quindi di utilizzare lo scheletro efficientemente minimizzando le moltiplicazioni tra matrici. L'algoritmo di disegno procede quindi ricorsivamente, con un approccio top-down, a calcolare l'effettiva posizione del joint corrente applicando la trasformazione del padre. In questo modo si evitano la ripetizione della stessa operazione che sarebbe emersa in un approccio bottom-up.



figura 5.12: Screenshot n. 1 del disegno dello skeleton del rider

Un altro obiettivo era quello di disegnare lo scheletro di animazione non solo del rider corrente ma, di un qualsiasi oggetto animato.

Per soddisfare questo obiettivo si è ricorsi al `ResourceManagerDebug`. Esso contenendo tutti gli oggetti era il posto corretto da cui partire. Per ogni `CObject3d` si è proceduto a valutare se era un oggetto animato e nel caso, permettere all'utente



5.5. DISEGNO DEGLI SCHELETRI DI ANIMAZIONE



figura 5.13: Screenshot n. 2 del disegno dello scheletro del rider

tramite Beholder di lanciare il disegno dello scheletro.

Per capire se un oggetto è animato si è ricorsi ad un dynamic _ cast da CObject3d a CAnimated3dModel, sottoclasse che rappresenta gli oggetti 3d anche animati. Nel codice il dynamic _ cast è stato poi sostituito con un più veloce confronto tra ClassID.

Per gli oggetti animati è stato quindi inserito nel Beholder la possibilità di lanciarne il disegno dello scheletro. Essendo il disegno dello scheletro un’azione da compiere ad ogni frame, vista la sua continua modifica, si è proceduto a creare la classe SkeletonDebugDrawerManager. Essa permette di aggiungere un oggetto il cui scheletro verrà disegnato ogni frame. Permette inoltre di mettere in pausa il disegno oppure di fermare definitivamente il disegno.

Come già accennato tutte queste funzionalità sono disponibili da Beholder, in particolare per ogni oggetto visualizzato nel Beholder tramite il ResourcesManagerDebug è presente una proprietà che indica (true o false) se l’oggetto è animato, e quindi è possibile disegnarne lo scheletro.

In figura 5.9 è visibile sulla destra le proprietà di un oggetto 3D, tra le quali isAnimated. In alto a sinistra invece ci sono le azioni disponibili, tra le quali il disegno dello scheletro. Azione che verrà davvero eseguita solo se l’oggetto è animato.

Di seguito è presente il codice della classe SkeletonDebugDrawerManager.

Listing 5.11: SkeletonDebugDrawerManager.h

```
#ifndef SKELETON_DEBUG_DRAWER_MANAGER_H
#define SKELETON_DEBUG_DRAWER_MANAGER_H

class SkeletonDebugDrawerManager : public DebugWorldElement
{
private:
    SkeletonDebugDrawerManager();
}
```



5.5. DISEGNO DEGLI SCHELETRI DI ANIMAZIONE

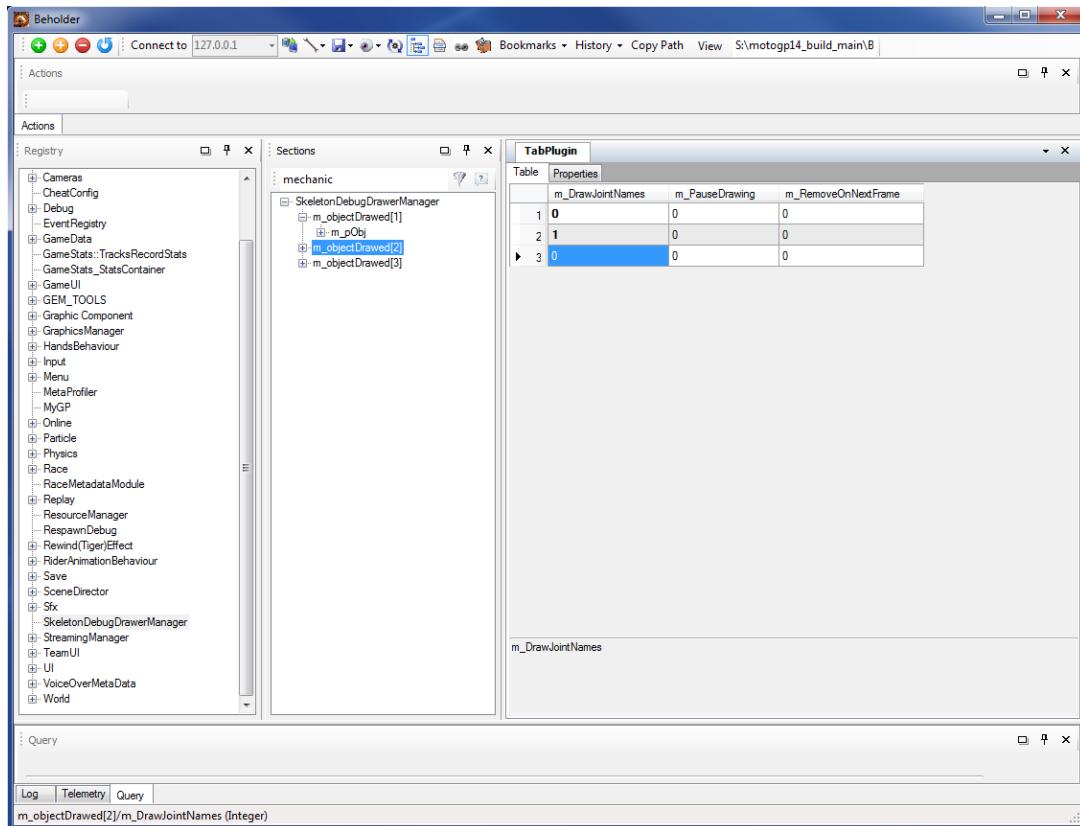


figura 5.14: Screenshot del Beholder in cui è visibile lo SkeletonDebugDrawerManager



figura 5.15: Screenshot di MotoGP 14 in cui è stato attivato il disegno degli scheletri di animazione di 2 meccanici



5.5. DISEGNO DEGLI SCHELETRI DI ANIMAZIONE

```
static SkeletonDebugDrawerManager* s_instance;

struct ObjectDrawedInfo
{
    ENABLE_METADATA;

    CAnimatedModel3d* m_pObj;
    GUIInt32 m_DrawJointNames;
    GUIInt32 m_PauseDrawing;
    GUIInt32 m_RemoveOnNextFrame;
};

vector<ObjectDrawedInfo> m_objectDrawed;

public:

    ENABLE_METADATA;

    static SkeletonDebugDrawerManager* GetInstance();

    void Draw( const DrawParams & params );

    //Return true if the added object hasn't been present yet
    GBool AddObject(CAnimatedModel3d* i_pObj, GUIInt32
        i_DrawJointNames);

    void RemoveObject(CAnimatedModel3d* i_pObj);
};

#endif
```

Listing 5.12: SkeletonDebugDrawerManager.cpp

```
#include "SkeletonDebugDrawerManager.h"

START_META("SkeletonDebugDrawerManager", SkeletonDebugDrawerManager)
    META_PROPERTY(m_objectDrawed)
END_META;

START_META("SkeletonDebugDrawerManager::info",
    SkeletonDebugDrawerManager::ObjectDrawedInfo)
    META_PROPERTY(m_pObj),
    META_PROPERTY(m_DrawJointNames),
    META_PROPERTY(m_PauseDrawing),
    META_PROPERTY(m_RemoveOnNextFrame),
END_META;

SkeletonDebugDrawerManager* SkeletonDebugDrawerManager::s_instance
(0);

SkeletonDebugDrawerManager::SkeletonDebugDrawerManager()
{
    CML_MetadataRegistry::AddEntry("SkeletonDebugDrawerManager",
        this);
}

SkeletonDebugDrawerManager* SkeletonDebugDrawerManager::GetInstance
()
{
    if(!s_instance)
```



5.5. DISEGNO DEGLI SCHELETRI DI ANIMAZIONE

```
{  
    s_instance = GEM_NEW SkeletonDebugDrawerManager();  
}  
return s_instance;  
}  
  
void SkeletonDebugDrawerManager::Draw( const DrawParams & params )  
{  
    for(GUInt32 index = 0; index < m_objectDrawed.size(); ++index)  
{  
        if( !m_objectDrawed[index].m_PauseDrawing )  
        {  
            SkeletonDebugDrawer::DrawSkeleton(m_objectDrawed[index].  
                m_pObj, m_objectDrawed[index].m_DrawJointNames);  
        }  
    }  
  
    vector<ObjectDrawedInfo>::iterator iterator = m_objectDrawed.  
        begin();  
    while( iterator != m_objectDrawed.end() )  
    {  
        vector<ObjectDrawedInfo>::iterator iteratorCurr = iterator;  
        iterator++;  
  
        if(iteratorCurr->m_RemoveOnNextFrame)  
        {  
            if(m_objectDrawed.size() > 1)  
            {  
                m_objectDrawed.erase(iteratorCurr);  
            }  
            else  
            {  
                m_objectDrawed.clear();  
                break;  
            }  
        }  
    }  
}  
  
GBool SkeletonDebugDrawerManager::AddObject(CAnimatedModel3d* i_pObj  
, GUInt32 i_DrawJointNames)  
{  
    vector<ObjectDrawedInfo>::iterator iterator = m_objectDrawed.  
        begin();  
    vector<ObjectDrawedInfo>::iterator iteratorEnd = m_objectDrawed.  
        end();  
  
    GBool bIsPresentYet = false;  
  
    while(iterator != iteratorEnd && !bIsPresentYet)  
    {  
        if(iterator->m_pObj == i_pObj)  
        {  
            bIsPresentYet = true;  
        }  
        iterator++;  
    }  
  
    if( !bIsPresentYet )  
    {  
        ObjectDrawedInfo info;  
        info.m_pObj = i_pObj;  
        info.m_DrawJointNames = i_DrawJointNames;  
        info.m_RemoveOnNextFrame = false;  
        m_objectDrawed.push_back(info);  
    }  
}
```



5.5. DISEGNO DEGLI SCHELETRI DI ANIMAZIONE

```
ObjectDrawedInfo objInfo;
objInfo.m_pObj = i_pObj;
objInfo.m_DrawJointNames = i_DrawJointNames;
objInfo.m_PauseDrawing = 0;
objInfo.m_RemoveOnNextFrame = 0;

m_objectDrawed.push_back(objInfo);

}

return !bIsPresentYet;
}

void SkeletonDebugDrawerManager::RemoveObject(CAnimatedModel3d*
i_pObj)
{
vector<ObjectDrawedInfo>::iterator iterator = m_objectDrawed.
begin();
vector<ObjectDrawedInfo>::iterator iteratorEnd = m_objectDrawed.
end();

while(iterator != iteratorEnd)
{
if(iterator->m_pObj == i_pObj)
{
m_objectDrawed.erase(iterator);
break;
}
iterator++;
}
}
```


Capitolo 6

Conclusioni

In questo capitolo verranno presentate le riflessioni scaturite dall'analisi effettuata a posteriori dell'esperienza di stage. Verranno analizzati gli obiettivi raggiunti e l'esperienza acquisita durante il lavoro, concludendo con una valutazione di come la preparazione accademica si interfacci con il mondo del lavoro.

6.1 Raggiungimento degli obiettivi

Lo studente ha raggiunto tutti gli obiettivi pianificati ad inizio stage, con soddisfazione dell'azienda e del tutor esterno. Questa ha infatti proposto allo studente altri sei mesi di collaborazione per continuare il percorso iniziato e la formazione on-the-job.

I tool sviluppati durante la prima parte dello stage sono infatti stati apprezzati e verranno presto inseriti nel modus operandi aziendale. In essi lo studente ha soddisfatto tutti i requisiti posti in fase di analisi degli stessi, seppur sforando di 4 giorni lavorativi sul tempo pianificato per la conclusione definitiva di XML Editor. Tempo recuperato poi durante lo svolgimento della seconda parte.

Di seguito viene riportata una tabella ed un grafico contenenti il resoconto della suddivisione delle ore per macro attività ed il confronto fra le ore preventivate e quelle a consuntivo. Le ore riportate di seguito sono comprensive del tempo dedicato alla stesura della documentazione.

Attività	Ore a preventivo	Ore a consuntivo	Variazione
Analisi	130	110	-20
Progettazione	45	65	+20
Codifica	85	90	+5
Verifica	60	55	-5
Totale	320	320	0

tabella 6.1: Riepilogo ore a preventivo e consuntivo

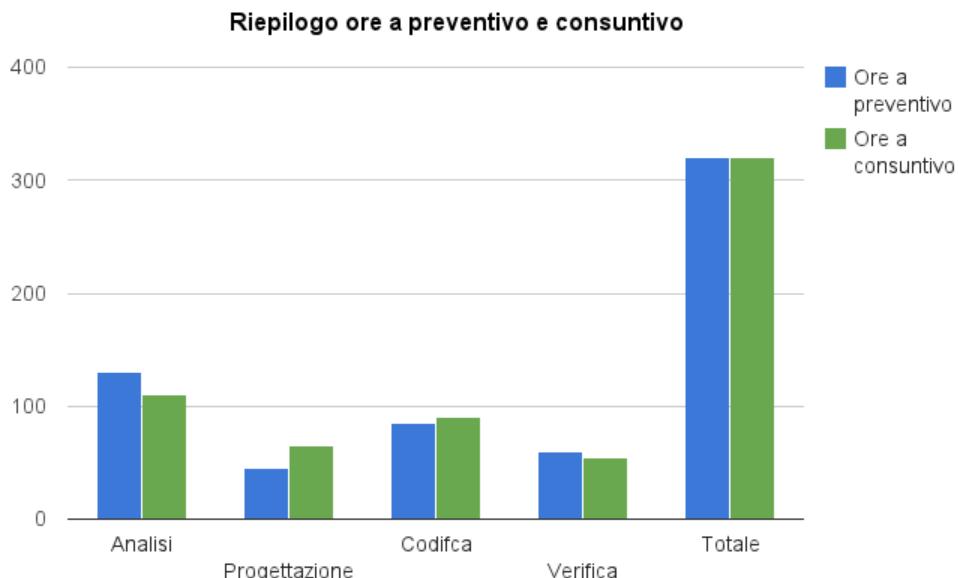


figura 6.1: Istogramma del riepilogo ore a preventivo e consuntivo

Come visibile in tabella 6.1 ed in figura 6.1 si nota che l'attività di analisi e formazione ha occupato 20 ore meno del previsto, le quali sono state quasi tutte reinvestite in progettazione, spiegabile dalla complessità del sistema con il quale lo studente ha interagito.

La codifica invece ha portato via poco in più di ciò preventivato e, vista l'impossibilità di aumentare il budget ore complessivo, si è stati costretti a tagliare sulla verifica.

6.2 Conoscenze acquisite e valutazione della preparazione accademica

Lo studente ha approfondito la sua conoscenza delle librerie Qt[®], soprattutto ha potuto maturare esperienza nella progettazione di un'architettura che si adattasse al meglio con il framework in questione e che al contempo risultasse facilmente comprensibile ed estendibile. Egli ha potuto inoltre mettere in pratica le conoscenze di usabilità nel progettare la user interface dei tool, acquisite nei corsi di Tecnologie Web.

Lo stagista ha inoltre potuto approfondire le proprie conoscenze nella grafica 3D, soprattutto nell'ambito delle animazioni scheletrali. C'è da dire, sotto questo aspetto, che le competenze necessarie per svolgere gli obiettivi della seconda parte sono state acquisite dallo studente in maniera autonoma durante tutto il percorso di studi, utilizzando come basi per comprendere le nuove nozioni ciò studiato durante il percorso accademico.

Tranne le conoscenze di grafica 3D bisogna affermare che la preparazione fornita durante il percorso di studi è stata adeguata alle necessità aziendali. Durante lo sviluppo si è potuto constatare che il metodo di lavoro acquisito si adattava bene a quello aziendale.

Si è constatato inoltre la necessità di autonomia nella ricerca e nell'approfondimento di particolare tematiche di cui lo studente non era a conoscenza con gli strumenti a



6.3. VALUTAZIONE PERSONALE

disposizione (internet e manuali).

Tutto il codice prodotto, soprattutto quello dei tool, è stato documentato nel migliore dei modi per poter permettere a chiunque di comprenderlo ed evolverlo in completa autonomia.

È convinzione dello studente che, al di là degli aspetti teorici e pratici di qualunque tecnologia, un informatico maturo che abbia compreso gli aspetti fondamentali dell'informatica insegnati in questo corso di studio, potrà durante la sua intera carriera acquisire senza troppi problemi tutte le tecnologie di cui avrà bisogno.

6.3 Valutazione personale

Lo studente ritiene che lo stage sia stata un'esperienza decisamente positiva, oltre agli obiettivi raggiunti, l'ambiente in cui è stato inserito si è dimostrato positivo e amichevole. La positività che lo stagista ha sempre dimostrato è stata ricambiata dai colleghi e dalla direzione che ha confermato la collaborazione. Altro motivo di crescita è derivato dal trasferimento a Milano, sede di Milestone S.r.l., per lo svolgimento del tirocinio, che ha permesso allo studente di crescere e maturare anche da questo punto di vista.

Per tutti questi motivi si afferma la buona riuscita del periodo di stage.

Glossario

AAA Nel mondo dei videogiochi i titoli *AAA* (pronunciato “tripla a”) definisco quei titoli di punta sviluppati dalle case più grandi con maggiori budget. 7

Alienbrain® Alienbrain® è un sistema di versionamento proprietario (sito: <http://www.alienbrain.com/>) . Il suo punto di forza è la gestione efficiente dei formati non testuali, quali ad esempio gli asset grafici. 10

Ban In informatica con il termine *ban* si intende l'esclusione, a tempo determinato o non, di un utente da un servizio di qualsivoglia genere. 4

Bug In informatica con il termine *bug* si definisce un problema presente in un codice che porta questo a mostrare comportamenti indesiderati, ad esempio crash improvvisi. 5

Build Con il termine *build* si intende il processo di trasformazione di una qualche risorsa digitale. In questo caso si intende la compilazione/linking del codice e la conversione e compressione degli assetti. 12

Day One Per *day one* si intende il primo giorno in cui un gioco viene reso disponibile per l'acquisto. 4

Debug In informatica con il termine *debug* si intende la pratica dell'analisi del comportamento del codice, soprattutto a run-time, allo scopo di individuare e risolvere bug (difetti). 1

Delta Con *delta* si vuole intendere la differenza fra due oggetti. 10

Engine In informatica con il termine *engine* si intende un software che funge da base per altri. Nello specifico dei videogame, l'engine è spesso scambiato per il solo motore grafico. Invece questo comprende praticamente tutte le funzioni di base che poi vengono specializzate per la creazione di specifici video game. 1

First Playable Nel mondo dei videogiochi con *Vertical Slice* si intende una versione di un game non ancora completa, in cui i livelli sono ancora delle bozze (draft) che rendono l'idea di come sarà il gioco. Lo scopo di queste versioni sono per la maggior parte il test del gameplay e la verifica del design. 7

Hackerate Nel contesto di questa tesi, il termine è stato usato per indicare la pratica di bucare il software delle console o dei videogiochi per poter giocare con copie non autorizzate. 4

Junior In informatica con il termine *junior* si intende una figura professionale agli inizi di carriera senza significative esperienze regresse sul campo di interesse. 1



Glossario

Leaderboard Nei videogiochi online, le *leaderboard* sono delle classifiche sulle prestazioni dei giocatori. 6

LOD Con l'espressione level of detail (LOD), si intende la tecnica utilizzata nelle applicazioni di grafica 3D. Tale tecnica consiste nell'avere multiple versioni degli asset con livelli di dettaglio e quindi spesa computazionale differente e, al variare della visibilità degli oggetti utilizzare versioni progressivamente meno dettagliate. Si riesce in questo modo ad aumentare la qualità grafica complessiva in quanto il tempo di calcolo risparmiato per oggetti distanti e comunque poco visibili viene poi sfruttata per aggiungere dettagli e/o oggetti in posizioni che l'utilizzatore può notare di più. 73

LUA Lua è un linguaggio di programmazione dinamico, riflessivo, imperativo e procedurale, utilizzato come linguaggio di scripting di uso generico. È spesso usato nei videogame. 9

Mesh Nel settore della grafica 3D, il termine *mesh* indica l'insieme dei poligoni (triangoli) che compongono un oggetto tridimensionale. 8

Microsoft Project *Microsoft Project* è un software per la gestione di progetti. Maggiori informazioni sono reperibili al seguente indirizzo: <http://office.microsoft.com/it-it/gestione-di-progetti-e-portfolio-microsoft-project-FX10472268.aspx>. 7

Middleware In informatica con il termine *middleware* si intende un insieme di programmi informatici che fungono da intermediari tra diverse applicazioni e componenti software. Fonte <http://it.wikipedia.org/wiki/Middleware>. 6

Perforce® Perforce® è un sistema di versionamento proprietario (sito: <http://www.perforce.com/>). Il suo punto di forza sono i branch intelligenti (stream). 10

PSN® Il Playstation Network® (PSN®) è il servizio offerto da Sony agli utenti delle proprie console. Esso permette di giocare online con altri giocatori e l'acquisto di contenuti digitali (film, giochi ecc.). 4

Raknet RakNet è un cross-platform C++ and C# game networking engine. Fonte <http://www.jenkinssoftware.com/features.html>. 6

Refactoring In informatica con il termine *refactoring* si indica la pratica di riorganizzazione del codice, allo scopo di migliorarne le priorità. 1

Serializzazione Con *serializzazione* si intende il processo di trasformazione di una risorsa (tipicamente testuale) in formato binario. 11

Sistema di integrazione continua Un *sistema di integrazione continua* permette di eseguire costantemente le build ogni volta che qualche dato viene aggiornato. Questo permette di avere un controllo continuo sul lavoro eseguito. 12

Team In informatica con il termine *team* si intende un gruppo di sviluppatori che collaborano in maniera organizzata per raggiungere un obiettivo comune. 1

Texture Nella conoscenza comune la texture è l'immagine 2D che viene applicata alle mesh per farle sembrare veri oggetti. Nell'ambito invece della programmazione il termine assume invece un contesto più generale, significando una matrice di punti (considerabile come un'immagine) la quale però viene utilizzata per contenere molte altre cose. 8



Vertical Slice Nel mondo dei videogiochi con *Vertical Slice* si intende una versione di un game non ancora completa, ma presenta un assaggio completo di tutti gli strati che lo compongono. L'esempio classico di vertical slice è una demo in cui è permesso al giocatore di giocare un solo livello (sui tanti che comporranno il gioco completo) in maniera completa. 7

XML eXtensible Markup Language (XML) è un linguaggio di markup basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento di testo sia human che machine readable. 8

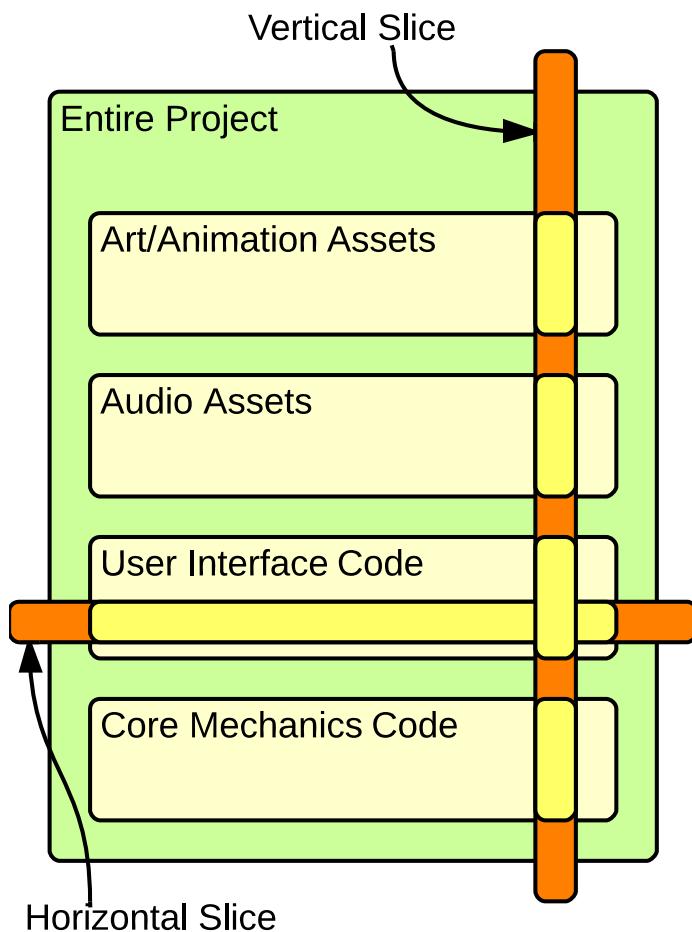


figura 2: Rappresentazione grafica di vertical e horizontal slice

Bibliografia

Riferimenti bibliografici

- [Lun12] Frank D. Luna. «Introduction to 3D Game Programming with DirectX® 11». In: 1st. Dulles: Mercury Learning e Information, 2012. Cap. 3, 5, 24, 25 (cit. alle pp. 75, 93).
- [MF99] John Brant William Opdyke Don Roberts Martin Fowler Kent Beck. «Refactoring: improving the design of existing code». In: 1st. Westford: Addison Wesley Longman, Inc, 1999 (cit. a p. 73).
- [MM13] David Graham Mike McShaffry. «Game Coding Complete». In: 4th. Boston: Course Technology, 2013. Cap. 12 (cit. a p. 9).

Siti Web consultati

- [Sita] *Accordo Namco Bandai*. 2014. URL: http://motogpvideogame.com/wp-content/uploads/MOTO-GP14-announcement_v4.pdf (cit. a p. 3).
- [Sitb] *Approfondimento game testing*. 2014. URL: http://en.wikipedia.org/wiki/Game_testing (cit. a p. 4).
- [Sitc] *Approfondimento TRC*. 2014. URL: <http://research.ncl.ac.uk/game/mastersdegree/workshops/technicalrequirementschecklists/Technical%20Requirements%20Checklist%20Workshop.pdf> (cit. a p. 3).
- [Sited] *Lua SPE paper*. 2014. URL: <http://www.lua.org/spe.html> (cit. a p. 9).
- [Site] *Milestone Home Page*. 2014. URL: <http://milestone.it/azienda/chisiamo/> (cit. a p. 1).
- [Sitf] *Skeletal Animation*. 2010. URL: http://www.wazim.com/Collada_Tutorial_1.htm (cit. a p. 93).