ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA
SCUOLA DI INGEGNERIA E ARCHITETTURA
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

CROSS-PLATFORM GAME DEVELOPMENT

Tesi di Laurea in Mobile Web Design

Relatore:
Dott. Mirko Ravaioli

Presentata da:
Alex Grassi

Sessione III
Anno Accademico 2013-2014

Indice generale

1 I videogiochi.....3
1.1 Caratteristiche.....3
1.2 I principali generi.....3
1.3 Storia ed evoluzione dei videogiochi.....4
1.3.1 Dagli anni cinquanta ad oggi 4
1.3.2 Il giocatore.....5
1.3.3 L'aspetto sociale.....6
1.3.4 Il mercato.....6
1.3.5 L'evoluzione delle piattaforme.....7
1.4 Cosa compone e definisce un videogioco.....8
1.5 Non solo risorse.....10
1.6 Le principali fasi nello sviluppo di un videogioco.....10
2 Studio delle Tecnologie.....15
2.1 La scelta del genere.....15
2.1.1 Confronto fra le piattaforme.....15
2.1.2 Analisi delle risorse.....16
2.1.3 La scelta.....19
2.2 La scelta delle tecnologie.....20
2.2.1 Engines.....20
2.2.2 ShiVa Engine.....28
2.2.3 Modellazione.....34
2.2.4 Character Creation.....35
2.2.5 Altri software utilizzati.....36
3 Concept Doc.....37
3.1 High Concept.....37
3.2 Genere.....37
3.3 Gameplay.....37
3.4 Features.....38
3.5 Settings.....38
3.6 Story.....40
3.7 Target Audience.....41
3.8 Hardware Platform.....42
3.9 Estimated Schedule.....43
3.10 Competitive Analysis.....44

4 Progettazione e sviluppo....47
4.1 Studio dell'ambiente di sviluppo....47
4.2 Analisi delle prestazioni per piattaforma....50
4.3 Scelta della leading platform....50
4.4 Analisi dei requisiti e progettazione....51
4.5 Sviluppo....54
4.5.1 Definizione dei comandi....55
4.5.2 Creazione del corpo fisico del giocatore....56
4.5.3 Gestione della telecamera di gioco....57
4.5.4 Introduzione dell'interazione con gli oggetti....58
4.5.5 Gestione delle variabili dei componenti in gioco....60
4.5.6 Definizione del comportamento degli oggetti....60
4.5.7 Estensione delle funzionalità 2D del motore....62
4.5.8 Creazione delle interfacce 2D....65
4.5.9 Gestione dell'audio....66
4.6 Creazione di un sottogioco....66
4.7 Modellazione ed animazione degli oggetti....71
4.8 Ottimizzazione....76
4.8.1 Ottimizzazione delle facce dei modelli....76
4.8.2 Limitazione delle classi attive....77
4.8.3 Gestione dei modelli extra in gioco....77
4.8.4 Definizione di oggetti come occluder ed occludable....78
4.8.5 Gestione del caricamento degli ambienti di gioco....79
4.8.6 Ottimizzazione in ottica multipiattaforma....80
4.8.7 Modifiche al gameplay....82
4.9 Menu....83
4.10 Testing....83
4.11 Creazione della demo....84
4.12 Screenshots....85
5 Conclusioni....89
5.1 I risultati raggiunti....89
5.1.1 Prestazioni....89
5.2 Riflessioni sullo sviluppo multi-piattaforma....90
5.3 Piano di lavoro post laurea....91
5.3.1 Miglioramento della classe adibita alle cut scene....91

5.3.2 Estensione della classe Comp2dAI..91
5.3.3 Miglioramento della navigabilità dei menu tramite gamepad.......................................92
5.3.4 Sottogiochi multi classe...92
5.3.5 Miglioramento del motore delle collisioni 2D..92
5.3.6 Sistema di salvataggio\caricamento delle partite..93
5.3.7 Introduzione di un secondo metodo per la creazione di sottogiochi bidimensionali......93
5.3.8 Preparazione del gioco alla release sul market Android...94
5.3.9 Modifica del sistema con cui vengono gestite le lightmap..94

Introduzione

Si definisce videogioco un gioco elettronico, in cui l'utente può interagire con esso tramite diversi tipi di user interface e che produce un output sotto forma di feedback video.

La programmazione di videogiochi risulta essere una delle applicazioni dell'informatica che più si discosta dall'utilizzo comune, applicando la logica alla base del codice non per realizzare un compito "pratico", ma per intrattenere l'utente.

I videogiochi sono un medium relativamente nuovo ed in rapida evoluzione. Nati intorno agli anni cinquanta sono passati, in poco più di mezzo secolo, dall'essere dei prototipi realizzati in centri di ricerca informatica ad uno dei passatempi più comuni e praticati nella società moderna. La sempre crescente domanda di nuovi prodotti e differenti contenuti ha continuato ad alimentare la crescita del mercato (attualmente intorno ai 100 miliardi di dollari[1]) favorendo la diversificazione delle piattaforme e dei generi, un mercato, sufficientemente forte da sostenere prodotti il cui sviluppo può arrivare a costare centinaia di milioni di dollari (in pari con le più grosse produzioni Hollywoodiane) ma anche sufficientemente sensibile da notare, supportare e premiare progetti più piccoli ma diversi dai generi più canonici.

La rapidità con cui i videogiochi come medium si stanno evolvendo interessa anche il modo in cui essi vengono distribuiti, in pochi anni infatti, la maggior parte dei prodotti venduti ha abbandonato il supporto fisico in favore del Digital Delivery, questo non ha fatto altro che facilitare ulteriormente gli sviluppatori più indipendenti, permettendogli con estrema facilità di avere il proprio lavoro disponibile sui market di tutto il mondo.

Obiettivo della tesi

L'obiettivo di questa tesi è stato quello di progettare ed avviare lo sviluppo di un videogioco 3D multipiattaforma, nello specifico PC(Windows) e Mobile(Android), realizzandone poi una demo, affrontando così tutte le problematiche che possono sorgere quando si sviluppa un programma per dispositivi profondamente diversi fra loro.

Il videogioco che si è scelto di realizzare è del genere "First Person Adventure", in cui il giocatore deve esplorare la mappa di gioco, ed interagire con i vari elementi presenti al fine di far procedere la storia, il tutto realizzato ad un livello qualitativo sufficiente a permetterne la commercializzazione una volta ultimato.

Sommario

1. I videogiochi

   Nel primo capitolo si descrive brevemente l'evoluzione dei videogiochi e delle loro piattaforme, per poi passare ad elencare gli elementi di cui è composto un videogioco.

2. Studio delle tecnologie

   Nel secondo capitolo si affrontano tutte le problematiche relative alla progettazione di un videogioco, tenendo conto dello stato del mercato e delle risorse disponibili.

3. Concept Doc

   Nel terzo capitolo si produce un documento in linea con ciò che verrebbe realizzato in una azienda per presentare l'idea di un nuovo videogioco.

4. Progettazione e sviluppo

   In questo capitolo si descrive lo sviluppo del videogioco, andando ad elencare le molteplici risorse prodotte per esso, ed evidenziando le principali problematiche sorte dalla sua natura multipiattaforma.

5. Conclusioni

   In quest'ultimo capitolo si analizzano i risultati ottenuti e si descrive come lo sviluppo procederà post tesi.

1 I VIDEOGIOCHI

In questo capitolo viene descritto in cosa consiste un videogioco e quali sono i suoi principali generi, per poi riassumerne brevemente storia ed evoluzione, ed analizzare da quali risorse è composto e definito.

Il capitolo termina con l'elenco dei principali passaggi nello sviluppo di un videogioco.

1.1 Caratteristiche

"Un videogioco può essere definito come un software con cui un giocatore (o più) può interagire, il cui scopo è intrattenere o educare"

I videogiochi sono uno dei passatempi moderni più diffusi, capaci di catturare gli interessi di milioni di persone e sostenere un mercato in continua espansione, nati nei centri di ricerca e nelle università statunitensi negli anni cinquanta, in poco più di sessant'anni sono diventati un prodotto comune, utilizzato in diverse forme dalla maggior parte della popolazione.
C'è chi li usa per passare il tempo al cellulare in attesa di un treno, chi investe somme importanti per avere una piattaforma in grado di supportare le simulazioni più complesse, o chi li usa come passatempo serale con gli amici o la famiglia. In un modo o nell'altro, i videogiochi sono diventati parte della nostra società.

1.2 I principali generi

Anche se la maggior parte dei videogiochi moderni non può essere completamente rappresentata da una sola categoria, essendo spesso un mix di più generi, è comunque sempre possibile individuarne il genere dominante in uno dei seguenti:

- **Action Game:** In questo genere il giocatore si trova ad affrontare prove (sotto forma di ostacoli o antagonisti), in cui vengono testati i suoi riflessi e capacità di prendere decisioni rapidamente.
- **Strategia:**In questa tipologia il giocatore mette alla prova le sue capacità di gestione di grossi quantitativi di unità, a differenza del precedente genere, in questo caso, ciò che viene

premiato non sono i riflessi, ma la capacità di analizzare una grossa mole di informazioni e pianificare una strategia.

- **Avventura:** nei giochi di avventura, il giocatore si trova a confrontarsi con enigmi, la cui risoluzione fa procedere la storia, fulcro dell'intero gioco.
- **Role-Playing Game (RPG):** In questa tipologia di titoli il giocatore si trova a guidare uno o più personaggi in una serie di missioni, il fulcro del gioco è il prosieguo della storia e l'evoluzione dei personaggi gestiti.
- **Simulazione:** Da sempre un prodotto di nicchia, questi giochi cercano di emulare il comportamento nel mondo reale di macchinari complessi quali aerei, carriarmati, elicotteri etc.
- **Sport:** Questo genere permette di sperimentare la maggior parte degli sport, impersonando un giocatore o il manager di una squadra.
- **Casual:** La tipologia in maggiore espansione dall'avvento dei dispositivi mobile come piattaforma di gioco, questa categoria racchiude in se tutti quei giochi in cui la storia e la complessità del gameplay passano in secondo piano, a favore di un intrattenimento di breve durata ma immediato.
- **Educational:** Questa tipologia di videogiochi ha come obiettivo quello di educare l'utente, fino ad alcuni anni fa, la quasi totalità di questo genere era indirizzata ad un pubblico estremamente giovane, ma negli ultimi anni questa tendenza sta cambiando.

1.3 Storia ed evoluzione dei videogiochi

Nati negli anni cinquanta nei centri di ricerca informatica e nelle università degli Stati Uniti, in appena sessant'anni i videogiochi sono diventati uno dei medium più diffusi della nostra società.

Di seguito verranno analizzati i principali fattori di questa evoluzione.

1.3.1 Dagli anni cinquanta ad oggi

Il primo prototipo di gioco elettronico nacque nel 1947 ad opera di Thomas Goldsmith Jr. e Estle Ray Mann, in un periodo in cui il concetto di hardware e software non era ancora stato introdotto questi due ricercatori brevettarono il primo antenato dei moderni videogiochi, “Cathode-ray tube

amusement device”, il gioco lasciava all'utente un limitato periodo di tempo per “mirare” attraverso delle manopole un obiettivo e, nel caso in cui la mira fosse stata corretta, il giocatore avrebbe vinto. Il sistema risultò troppo costoso per essere commercializzato, ma nonostante questo, il primo videogioco era nato.

Nel decennio successivo vennero sviluppati altri prototipi ma fu solo nel 1961 che un videogioco, per la prima volta, fu largamente distribuito, si trattava di Spacewar!, programmato da un gruppo di studenti del MIT su un computer DEC PDP-1, il gioco fu così apprezzato che venne incluso in tutti i nuovi esemplari di computer DEC, sancendone così la distribuzione su larga scala.

Gli anni settanta videro la nascita della prima e della seconda generazione di console, il cabinato forse più conosciuto della storia, “Pong”, e dell'iconico gioco “Space Invaders”. Ormai un nuovo medium era entrato nella società e, forte di un mercato fiorente, stava crescendo rapidamente.

Nei trent'anni successivi i videogiochi hanno continuato ad evolversi favorendo inoltre l'avanzamento delle piattaforme che li hanno abbracciati, il mercato dei videogiochi, nonostante alcuni alti e bassi ha continuato a crescere fino a raggiungere dimensioni paragonabili ad altri medium quali il cinema o la musica.

Al giorno d'oggi la produzione di un videogioco di qualità può facilmente costare decine di milioni di dollari e richiedere team di cinquanta o più persone, la continua ricerca da parte delle case di produzione di sviluppare un prodotto sempre più coinvolgente ha dato vita alla nascita di figure professionali specifiche quali il Game Writer e lo ha ulteriormente avvicinato ad un medium quale il cinema, al punto da permettere lo scambio di figure professionali fra i due[2].

1.3.2 Il giocatore

L'evoluzione della natura del videogioco ha portato ad un cambiamento anche nel suo utilizzatore medio. Questa evoluzione può essere suddivisa in tre fasi:

1. **Gli albori (anni 50):** Inizialmente un passatempo o esercizio di stile per pochi studenti o ricercatori. I primi videogiochi erano realizzati all'interno dei centri di ricerca informatici e delle università americane e rappresentavano i primi tentativi di dare vita ad una nuova tipologia di passatempo.

2. **La nascita del mercato (70-90):** Una volta che il progresso tecnologico permise la produzione su larga scala di dispositivi elettronici quali cabinet, personal computer e

console, il videogioco divenne un prodotto per le masse, andando ad interessare principalmente due tipologie di giocatori, i giovani (grazie ai cabinet e alle console) ed una nicchia dei primi utilizzatori di personal computer.

3. **Il giocatore moderno (dal 2000 ad oggi):** Al giorno d'oggi, grazie alla sua evoluzione da semplice passatempo a medium di maggior spessore, il videogioco ha creato una base di utenza estremamente eterogenea in età (il giocatore medio ha 30 anni e gioca da 12) e sesso (il 42% dei giocatori è femminile).

1.3.3 L'aspetto sociale

Conseguentemente al cambio di utenza si è avuto un mutamento anche nel modo in cui il videogioco viene percepito socialmente, in passato visto maggiormente come un passatempo da solitari (da condividere unicamente con la ristretta cerchia di persone aventi lo stesso interesse), mentre ora come un evento da condividere con amici e familiari (un terzo dei genitori gioca regolarmente tutte le settimane con i proprio figli almeno ad un videogioco[3]).

1.3.4 Il mercato

Tutto questo non ha fatto altro che alimentare la crescita del mercato dei videogiochi e diversificare le piattaforme su cui se ne usufruisce, includendo ora non solo dispositivi "fissi" quali personal computer o console, ma anche mobile (attualmente, il numero di utenti che gioca regolarmente con cellulari e tablet ha raggiunto i 48 milioni [4])

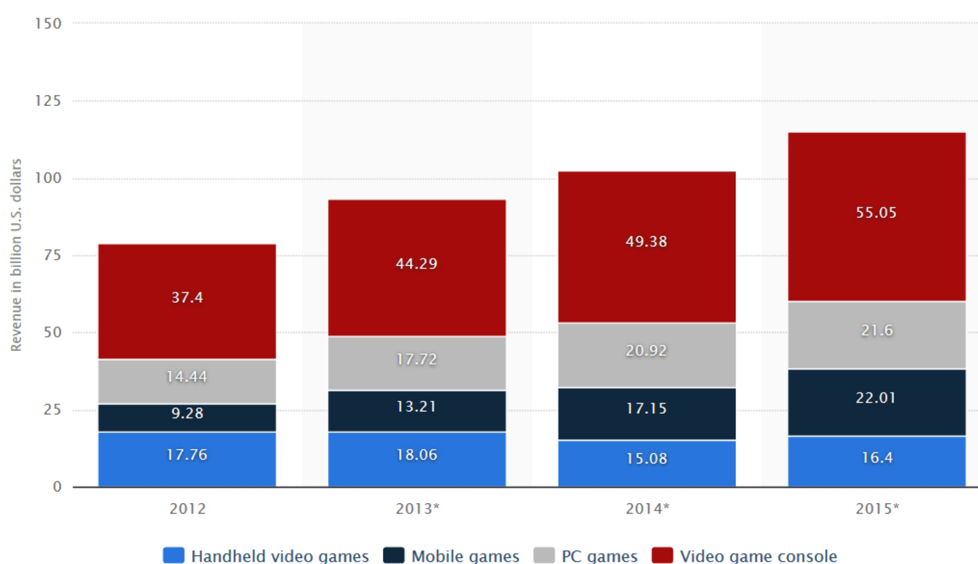

Illustrazione 1.1: Crescita del mercato dei videogiochi

1.3.5 L'evoluzione delle piattaforme

Con l'avanzare delle tecnologie e l'abbassamento dei costi della realizzazione di dispositivi elettronici, il numero (e le peculiarità) delle piattaforme per cui sviluppare videogiochi è aumentato sensibilmente.

Console

Dagli albori le console risultano essere il dispositivo più accessibile all'utente medio, con un buon compromesso fra potenza e semplicità, esse sono dispositivi costruiti appositamente per il gioco e vantano al momento la fetta di mercato più grande.

Negli anni, innumerevoli aziende si sono occupate della creazione di console realizzando spesso prodotti molto innovativi e particolari, di recente però il trend ha visto un abbandono per quanto riguarda la sperimentazione a favore di un avvicinamento dell'architettura ai moderni personal computer.

Personal Computer

Da sempre la piattaforma più potente e completa ma anche la più complessa, il personal computer, come le console, è stato fin dagli albori una piattaforma aperta ai videogiochi, e negli anni questo non ha fatto che affermarsi, portando alla nascita di brand specifici per il gioco, e, ancora più di recente, di computer box, computer pensati appositamente per il gioco, con un approccio più semplice ispirato alle console.

Handheld Video Games

Queste piattaforme possono essere definite come delle console portatili in cui si è avuto un trade off fra prestazioni e mobilità. Divenute estremamente diffuse negli anni 90 si trovano in questo momento a fronteggiare la crescita dei dispositivi mobile.

Dispositivi Mobile

Il mercato dei videogiochi per dispositivi mobile risulta essere il più giovane, questo perché, fino a pochi anni fa, la tecnologia per quanto riguarda cellulari e tablet non era in grado di offrire prestazioni sufficienti a giustificare lo sviluppo di molti videogiochi per essi. Questo però negli ultimi anni è drasticamente cambiato, al punto che, attualmente il mercato mobile risulta chiaramente quello in maggior espansione nonché il più aperto a prodotti indie.

1.4 Cosa compone e definisce un videogioco

Dal lato tecnico un videogioco non è altro che un'applicazione software che fa uso di diverse risorse al fine di interagire con l'utente.

Storia

Con il passare del tempo il videogioco si è avvicinato sempre maggiormente ad altri medium quali il cinema o i libri, includendo in maniera sempre più profonda una storia (realizzata dalla figura del *Game Writer)*, questo ha portato a vari "scambi" fra queste industrie, come ad esempio la creazione del videogioco Metro 2033, basato sul romanzo omonimo di Dmitry Glukhovsky, o la realizzazione della serie di film Resident Evil, basati sull'omonima serie di videogiochi.

La storia in un videogioco determinerà l'evolversi degli eventi e di conseguenza il codice del programma.

Gameplay

Il gameplay definisce in quale modo un giocatore può interagire con il videogioco, quali operazioni sono permesse (la possibilità di aprire una porta o di raccogliere un oggetto) e quali sono punite (uscire fuori dal tracciato in un gioco di corse risulta in una penalità).

Essendo il gameplay una parte fondamentale di un videogioco (definendo assieme alla storia il codice) con l'evolversi dell'industria si è vista la nascita di una figura professionale specializzata nella la sua definizione, il *Gameplay Designer.*

Codice

Un videogioco è un'applicazione software il cui compito è quello di interagire con il giocatore mediante una serie di risorse, il modo in cui queste interazioni avvengono è definito dal codice di gioco.

Modelli

Nel caso di un videogioco 3D, i principali elementi con cui il giocatore interagirà e che saranno mostrati a schermo sono modelli tridimensionali, ossia una rappresentazione software di un oggetto formata da un insieme di punti nel piano tridimensionale, connessi attraverso varie forme geometriche quali triangoli, linee etc, possono essere formati da un solo modello o da più sotto-modelli.

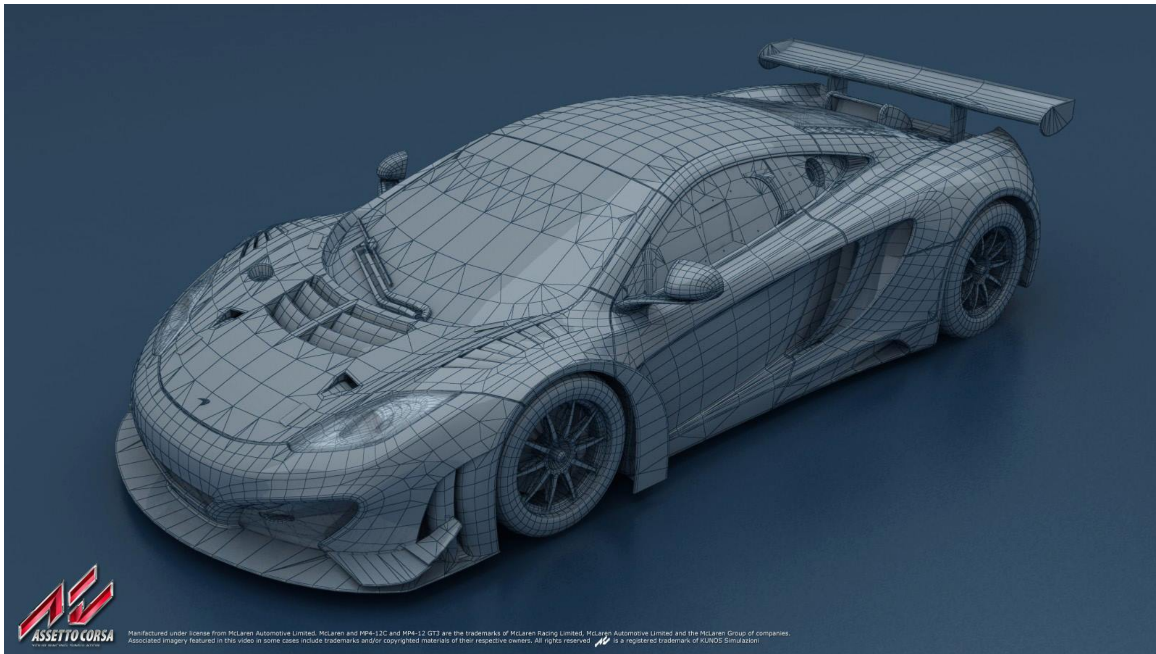

Illustrazione 1.2: Esempio di un modello 3D

Audio

Un'ulteriore via di interazione con il giocatore è data dalla presenza di suoni e musiche, necessari anche per facilitare la sua immersione nella storia, spesso gli studi di medie e grosse dimensioni tendono ad assumere compositori al fine di ottenere una colonna sonora adatta al prodotto in sviluppo ed unica.

Textures

Con il termine texture si definisce un'immagine software che, attraverso un procedimento chiamato texture mapping verrà applicata ad un modello 3D al fine di aggiungervi ulteriori dettagli[5]. Esistono diverse tipologie di texture, specializzate nell'aggiungere diversi tipi di dettagli, le più comuni sono:

1. **Diffuse Map:** Immagine digitale che definisce i colori del modello, una volta applicata definirà quali parti del modello saranno colorate in che modo, ma non vi applicherà alcun effetto speciale.
2. **Normal Map:** texture di colore blu, rosa e verde, utilizzata per simulare dettagli 3D senza il bisogno di modellarli, funziona unicamente in presenza di illuminazione dinamica e permette di limitare il numero di facce di un modello.
3. **Specular Map:** solitamente in bianco e nero, questa texture è utilizzata per definire in che modo, le parti del modello, reagiranno quando colpite dalla luce.

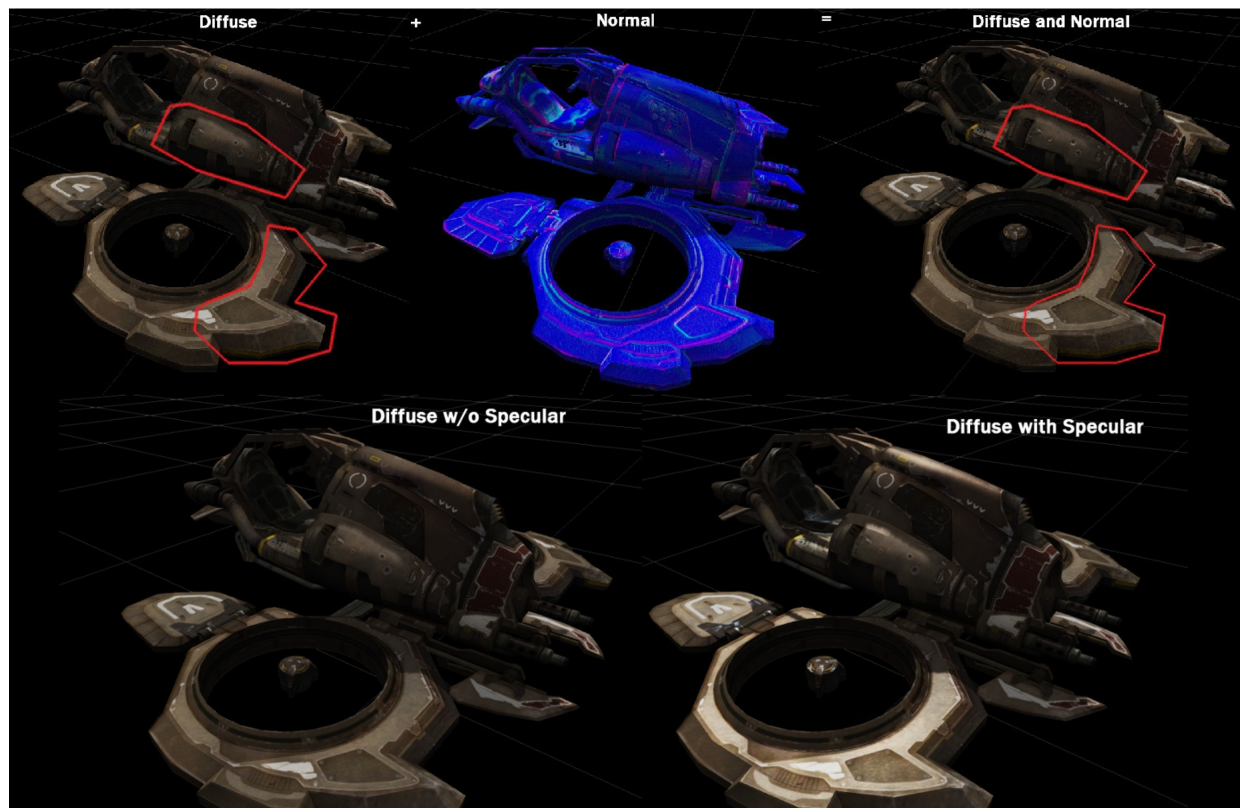

Illustrazione 1.3: Il risultato dell'applicazione delle varie tipologie di texture su un modello

1.5 Non solo risorse

Mentre per la maggior parte dei software l'obiettivo principale è quello di eseguire un determinato compito nella maniera più efficiente possibile, utilizzando al meglio le risorse disponibili, nel caso di un videogioco esso risulta più astratto, essendo quello di "intrattenere" l'utente, questo aggiunge un ulteriore livello di complessità durante la sua realizzazione, richiedendo al progettista di trovare il miglior equilibrio fra "intrattenimento offerto al giocatore" e "fattibilità del progetto".

Questo equilibrio si ripercuote su ogni componente precedentemente elencato già durante lo sviluppo di un'applicazione con una singola piattaforma come target, ma risulta ancora più critico nel caso in cui si sviluppi per più di una piattaforma, dato che le diverse specifiche tecniche porteranno al variare del rapporto fra "intrattenimento" e "fattibilità".

1.6 Le principali fasi nello sviluppo di un videogioco

Nella sua totalità, lo sviluppo di un videogioco non si adatta al meglio al tipico ciclo di vita di un software (ad esempio il modello a cascata), preferendovi approcci più dinamici quali ad esempio l'agile development.

Definizione 1.1: **Agile Development**. Nell'ingegneria del software per metodo di sviluppo “agile” si definisce un modello basato sulla prototipazione iterativa, in cui il cliente (in questo caso colui che deve definire se un gioco è “divertente” o meno) viene coinvolto direttamente nel processo di sviluppo.
Durante l'intero sviluppo di un videogioco vi è quindi una continua valutazione della qualità di intrattenimento del prodotto fino a quel punto realizzato, inizialmente il progetto viene provato da un gruppo di tester interni, poi, con il procedere dello sviluppo, si inizia ad invitare piccoli gruppi di utenti a testare il prodotto e valutarlo, prima con delle Closed Beta e poi con le Open Beta. In questo modo, se la software house risulterà in grado di accontentare le richieste degli utenti, si ritroverà una comunità attiva ed un numero sicuro di vendite al giorno del lancio del prodotto sul mercato, che si tradurrà in una maggiore possibilità di avere successo.

Anche se, ogni software house ha il proprio metodo di sviluppo, in generale, i passaggi più comuni nello realizzazione di un videogioco sono i seguenti[6].

Concept Development

La prima fase dello sviluppo di ogni videogioco, il cui obiettivo è quello di definire in che cosa consisterà il gioco, descrivendone la maggioranza del gameplay, preparando concept art e realizzando la prima bozza della storia (se presente).

Al termine di questa fase vengono prodotti tre documenti indispensabili per la presentazione del progetto ad un publisher:

1. **High Concept:** Una brevissima descrizione utilizzata per riassumere l'intero gioco.
2. **Pitch Doc:** Chiamato anche Game Proposal, un documento di poche pagine utilizzato per descrivere brevemente il gioco e perché potrà avere successo.
3. **Concept Doc:** Un'analisi di circa 10-20 pagine in cui viene descritto più a fondo il progetto suddividendolo in sezioni quali: High Concept, genere, gameplay, caratteristiche, ambientazione, storia, mercato target, piattaforme target, tempistiche, budget necessario e P&L, competitive analysis, membri del team e risk analysis.

Pre-Production

L'obiettivo di questa fase è quello di completare il game design, l'art bible (una descrizione dello

stile grafico del gioco), il piano di lavoro e realizzare un primo prototipo.

Game Design

In questo documento vengono descritte nel minimo dettaglio tutte le caratteristiche del gioco. Su di esso si baseranno poi i programmatori; è quindi necessario che ogni aspetto del gameplay sia descritto nei minimi dettagli.

Va ricordato che i designer, ascoltando i feedback ricevuti dai tester, continueranno a perfezionare le idee per la durata dell'intero progetto, quindi il documento finale sarà sempre aggiornato e corretto, spesso quotidianamente.

Sviluppo

E' la fase dove si crea il gioco vero e proprio, i programmatori realizzano il codice sorgente definitivo, i grafici i modelli a pieni dettagli e gli ingegneri del suono le musiche e gli effetti, vengono scritti i dialoghi, e realizzati i livelli, anche in questa fase i designer continuano a monitorare lo sviluppo.

Questa fase, nel caso di giochi di dimensioni standard può durare dai 6 mesi ai 2 anni, solitamente un sviluppo più rapido può solo significare una mancanza di testing ed un prodotto incompleto, al contrario, uno sviluppo che si protragga per più di due anni rischia di incappare in innumerevoli problemi quali ad esempio arretratezza delle tecnologie utilizzate o perdita di originalità del progetto (altri titoli potrebbero essere rilasciati nel mentre ed avere caratteristiche simili).

Demo

Solitamente inserita a cavallo fra la fase di sviluppo e l'arrivo in alpha del progetto, consiste nella realizzazione e release di una piccola porzione del gioco al pubblico, al fine di aumentare la visibilità del progetto sul mercato e raccogliere i primi feedback su ciò che si sta creando.

Alpha

La prima versione del gioco completa, ciò significa che generalmente il gioco è affrontabile dall'inizio alla fine nonostante l'assenza di piccole parti di codice o assets, l'arrivo a questa fase segna il mutamento dell'obiettivo del progetto, dalla costruzione di esso, al suo completamento.

Questa è anche la prima fase in cui tester esterni all'azienda sono chiamati a valutare il prodotto.

Beta

In questa fase il gioco è completo in tutte le sue parti, il focus cambia ancora passando al bug

fixing, cioè alla ricerca di errori nel progetto.

In questa fase il numero di tester esterni sale ancora, prima con la Closed Beta (un test aperto ad un numero definito ma piuttosto ampio di utenti), e poi con le Open Beta (test aperti a tutte le persone interessate).

Code Freeze

A questo punto il codice del gioco è “congelato”, cioè nulla viene più modificato se non a causa di bug critici non emersi in precedenza.

Release

Il gioco viene rilasciato al pubblico.

Patching

Per quanto la fase di testing sia stata esaustiva, alcuni bug possono essere sfuggiti, essi vengono così corretti post release attraverso la produzione di patches.

Testing

Il testing, come anche la prototipazione, non trova un posizionamento specifico nella scaletta di sviluppo perché eseguito parallelamente ad esso, fornendo feedback sullo stato del prodotto ed andando direttamente ad influenzarne il design.

2 STUDIO DELLE TECNOLOGIE

In questo capitolo si confrontano le piattaforme scelte per il videogioco in termini di specifiche tecniche e di mercato, si procede poi analizzando le risorse a disposizione e descrivendo il genere di gioco che si è deciso di sviluppare.

Il capitolo termina con una analisi delle principali tecnologie necessarie per lo sviluppo, e con un confronto, per ogni tipologia, fra le alternative considerate.

2.1 La scelta del genere

Per poter decidere quale genere di videogioco realizzare è stato prima necessario analizzare le piattaforme target e le risorse a disposizione del team.

2.1.1 Confronto fra le piattaforme

Per il confronto fra le due piattaforme prese in considerazione si è cercato di scegliere modelli usciti negli ultimi anni e con prestazioni nella media.

Pc	Mobile
Cpu: 4-8core, 2-3Ghz Gpu:HD 7850 Ram: 4-8 Gb	Cpu: 2-4 core 1-2Ghz Gpu: <<HD5450 Ram: 1-3Gb

Data la profonda differenza di architettura fra le due piattaforme, non è stato possibile fare un confronto diretto molto accurato, ciò che si è potuto dedurre però, è che la tecnologia utilizzata lato mobile risulta sensibilmente più limitata, il motivo di questo è da cercarsi nella necessità di sviluppare dispositivi dalle dimensioni e peso estremamente contenuti, mantenendo però un prezzo accessibile al pubblico.

Un'altra profonda differenza è la tipologia di input, nel primo caso abbiamo l'uso di mouse e tastiera o gamepad, periferiche estremamente accurate e che permettono un alto numero di input distinti (tutte le lettere presenti su una tastiera o tutti i tasti presenti su di un gamepad), i dispositivi mobile moderni invece prediligono l'uso di un touch screen, una tipologia di input incapace di trasmettere la stessa precisione delle precedenti periferiche, e che inoltre, condividendo lo schermo con il gioco, rischia di offuscarne il feedback.

Tutto questo rende lo sviluppo di certi generi di videogiochi problematico in ottica

multipiattaforma, dato che, non risulta possibile realizzare un prodotto qualitativamente accettabile per entrambe senza dover rimaneggiare pesantemente ogni aspetto di esso.

2.1.2 Analisi delle risorse

Appreso il livello prestazionale delle piattaforme target, è stato il momento di analizzare le conoscenze mie e dei miei collaboratori al fine di comprendere al meglio le capacità del team. Ogni componente del gruppo è stato confrontato con tutte le tipologie di risorse necessarie per lo sviluppo del gioco, attribuendogli un punteggio da 0 (nessuna conoscenza in materia) a 10(padronanza assoluta) e tenendo conto del numero di ore settimanali dedicabili al progetto.

	Alex Grassi	Filippo Maroni	Federico Fucci
Tempo (ore\settimana)	62	10	6
Scripting	9	0	7
Modellazione strutture	6	4	6
Modellazione anatomica	2	1	5
Grafica 2D	1	9	3
Media editing	7	0	3
Story Writing	9	7	5
Gameplay design	9	4	5
Level design	6	9	5

Alex Grassi

Laureato alla triennale di scienze e tecnologie informatiche ed attualmente laureando alla magistrale di ingegneria e scienze informatiche, in passato ha realizzato diversi prototipi di videogiochi 2D e 3D.

Crash

Progetto realizzato per la laurea triennale, Crash è un gioco di corse 3D multiplayer online in cui più macchine fluttuanti si sfidano su improbabili circuiti, l'ispirazione principale per questo gioco è stata F-Zero (uno dei titoli di maggior successo per console Super Nintendo).

Per il suo sviluppo si è utilizzato: Ogre3D per il rendering, Newton per la gestione della fisica e delle collisioni, mentre tutto il resto (caricamento dei modelli, gestione degli input, sincronizzazione motore fisico e grafico etc…) è stato realizzato in C++.

Il gioco è stato sviluppato unicamente per Windows e non è mai proceduto oltre il prototipo iniziale.

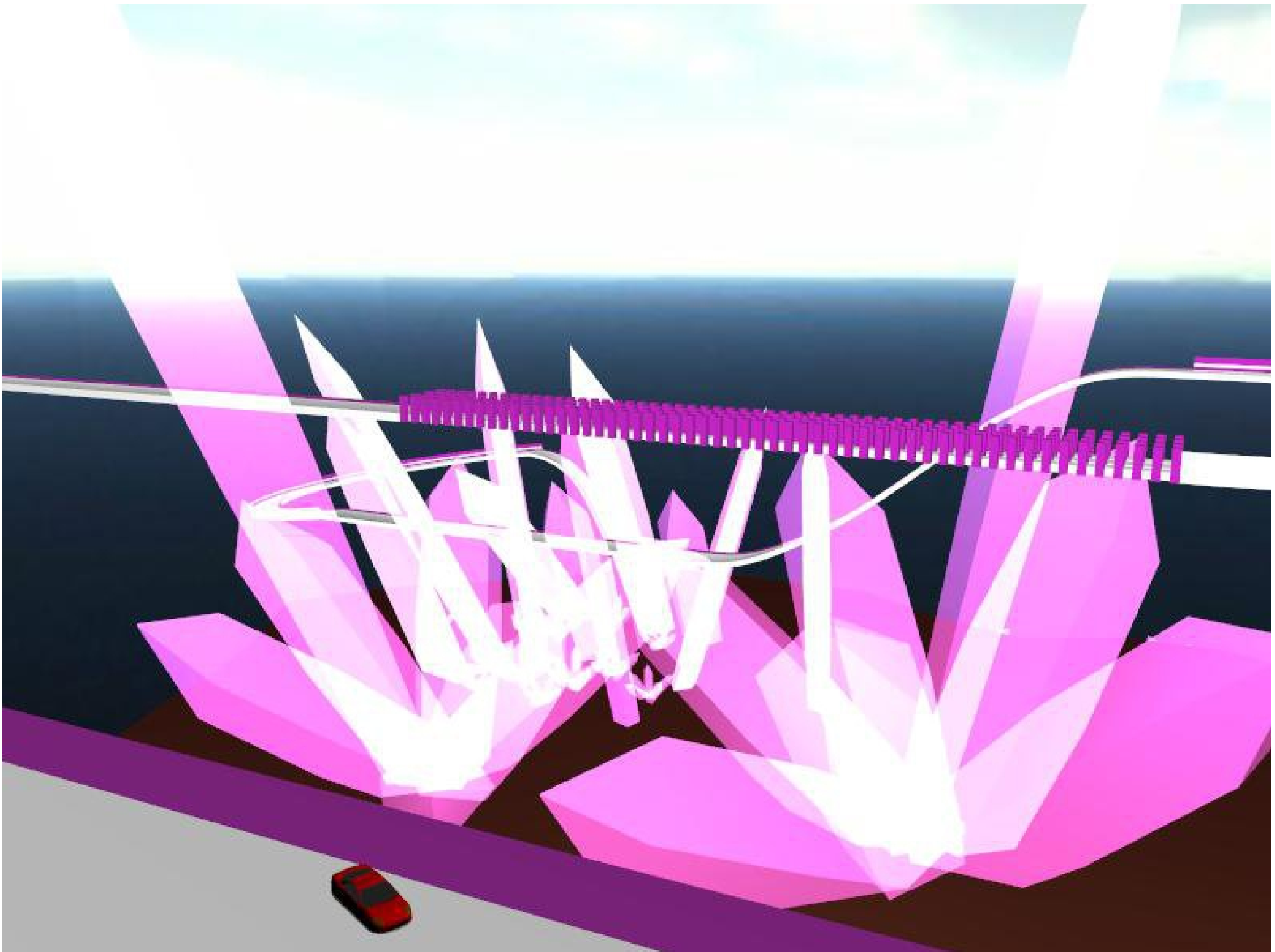

Illustrazione 2.1: Uno screenshot di Crash

A Strange Night

Progetto realizzato per l'esame di Mobile web design, A Strange Night è un gioco 2D in cui il giocatore deve difendersi da simpatici birilli killer, il gameplay prende ispirazione da titoli quali Space Invaders, in cui bisogna impedire agli antagonisti di raggiungere una determinata porzione dello schermo.

Per il suo sviluppo si era deciso di realizzare tutte le tecnologie necessarie internamente, scrivendo un motore grafico e fisico (in Java) chiamato A.M.A. (Ask Me Anything), ed estendendone le funzionalità parallelamente allo sviluppo del gioco.

Il tutto è stato sviluppato unicamente per Android ed ha raggiunto la fase di Alpha prima di essere abbandonato a causa della mancanza di risorse (essendo il gioco bidimensionale, ogni singolo elemento presente doveva essere disegnato ed animato separatamente, richiedendo tempi di sviluppo troppo lunghi).

Illustrazione 2.2: Uno screenshot di A Strange Night

Federico Fucci

Laureato al corso magistrale di ingegneria e scienze informatiche e attualmente assegnista di ricerca presso l'Università di Cesena, nel passato si è spesso occupato di modellazione e animazione 3D.

Illustrazione 2.3: Uno dei modelli realizzati da Federico

Filippo Maroni

Diplomato presso liceo artistico e laureato in architettura, attualmente architetto libero professionista, nel passato ha seguito corsi di disegno e realizzato alcuni fumetti.

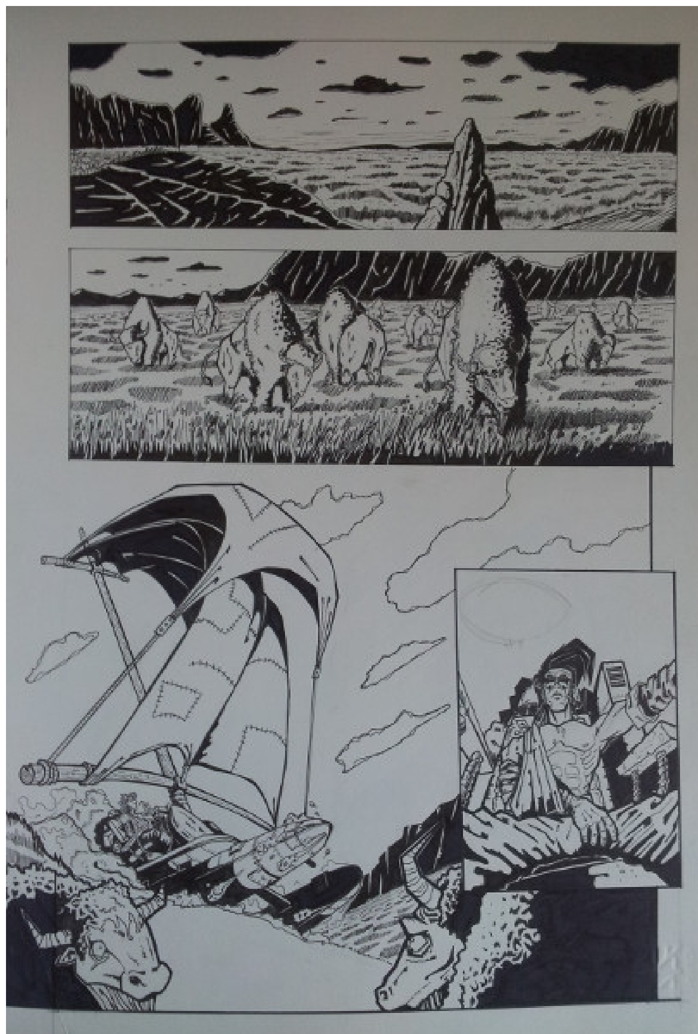

Illustrazione 2.4: Una tavola di Bounty Hunters, fumetto in sviluppo

2.1.3 La scelta

Tenendo conto delle caratteristiche tecniche delle piattaforme, delle conoscenze dei membri del team, e delle esperienze con i vari generi di videogiochi considerati (è sconsigliabile realizzare qualcosa di cui non si ha una buona conoscenza), si è deciso di sviluppare un videogioco a tre dimensioni basato sul genere Adventure, caratterizzato da:

- Visuale in prima persona, in modo da evitare la necessità di modellare ed animare il protagonista.

- Una storia che giustifichi perché il protagonista, durante l'intera avventura non si trovi mai ad interagire con altri NPC (non-player character), eliminando così la necessità di animare altri personaggi.
- Un'ambientazione che permetta la realizzazione della maggior parte della mappa di gioco attraverso la combinazione di moduli, precedentemente realizzati, riducendo così sensibilmente il numero totale di modelli da realizzare.
- Alcune brevi sezioni simili a titoli action (in modo da variare il gameplay e distinguere il prodotto dai concorrenti), ma progettate in modo da non richiedere alta precisione nella tipologia di input utilizzato, ma solo buoni riflessi, in modo da far si che gli utenti mobile non siano penalizzati e al contempo mantenere l'esperienza sulle due piattaforme il più simile possibile.
- La possibilità di salvare in ogni momento, in modo da venire incontro agli utenti mobile, che spesso tendono a giocare brevi partite negli intermezzi di tempo libero durante una normale giornata.
- Una durata complessiva di almeno 6 ore, considerata sufficiente in questo genere di giochi.

2.2 La scelta delle tecnologie

Con il genere scelto e le sue principali caratteristiche definite, è stato il momento di scegliere il software con cui sviluppare il videogioco.

2.2.1 Engines

Gli approcci possibili alla creazione di un videogioco possono essere principalmente suddivisi in tre tipologie:

1. **Semplice codice:** in questo caso il videogioco viene realizzato in tutte le sue parti internamente alla software house, questo richiede (nel caso di progetti anche solo minimamente complessi), tempi di sviluppo estremamente più lunghi o un secondo team dedicato esclusivamente allo sviluppo delle tecnologie necessarie.
2. **Utilizzo di uno o più middleware: i**n questo caso si decide di appoggiarsi almeno parzialmente a middleware realizzati da terzi, quali ad esempio un renderer grafico o un motore fisico, ma il compito di integrarli fra loro (ed implementare le risorse mancanti)

rimane della software house.

3. **Utilizzo di un Game Engine:** un framework realizzato da terzi appositamente ideato per la creazione e sviluppo di videogiochi e che include generalmente tutte le funzionalità richieste da essi.

Solitamente il primo approccio descritto viene scelto unicamente dalle software house più grandi, che possono permettersi il lavoro extra richiesto dallo sviluppare internamente le tecnologie necessarie, in favore di una maggior flessibilità una volta che lo sviluppo del videogioco ha inizio.

L'utilizzo di singoli middleware, considerato una buona alternativa per lo sviluppo su singole piattaforme, risulta invece poco consigliato in ottica multipiattaforma dato che questo approccio richiede un estensivo lavoro di riadattamento per ogni release.

L'approccio sempre più scelto nello sviluppo di videogiochi risulta essere quello di appoggiarsi ad un Game Engine, solitamente in sviluppo da diversi anni ed in grado di offrire prestazioni equiparabili a quelle di software realizzati ad hoc, in cambio di un piccolo trade off in flessibilità. Data l'ottica indie del progetto, si è optato per questa terza soluzione, e dalla ricerca di un game engine adatto sono emersi cinque candidati.

Unreal Engine

Attualmente alla versione 4.7.0, realizzato da Epic Games e considerato uno dei motori più versatili e potenti al mondo (nonché fra i più utilizzati).

- Debutta nel 1998
- C++ come linguaggio di scripting
- Shared-Source
- Editor per Windows e Mac (ed in sviluppo per Linux)
- Innumerevoli giochi di successo realizzati con esso, come ad esempio Gears Of War
- Performance e capacità di rendering fra le migliori al mondo
- Attualmente disponibile per 19 $ al mese +5% dei ricavi lordi
- Permette l'export per piattaforme Windows, Mac OS, iOS, Android, Xbox One e PlayStation 4, con Linux ed HTML 5 in sviluppo.

- Grossa community e documentazione

Illustrazione 2.5: Dreadnought, videogioco attualmente in sviluppo usando l'Unreal Engine 4

Ciò che rende l'Unreal Engine uno dei migliori motori con cui lavorare è l'approccio con cui è sviluppato, essendo utilizzato internamente da Epic Games per realizzare giochi, questo negli anni ha permesso alla casa di sperimentare (e quindi correggere) personalmente le problematiche che potevano sorgere nello sviluppo a causa di scelte di design del motore non corrette.

Come si vedrà in seguito, l'Unreal Engine non è stato scelto come motore per lo sviluppo del videogioco di questa tesi, non perché non adatto, ma perché purtroppo nel momento in cui si è iniziato lo sviluppo, l'offerta tramite abbonamento precedentemente descritta ancora non era presente, inoltre la learning curve piuttosto ripida avrebbe rischiato di rallentare eccessivamente lo sviluppo.

Unity

Arrivato alla versione 4.6, realizzato da Unity Technologies e considerato fino di recente un motore adatto allo sviluppo di progetti di piccole e medie dimensioni, la sua comprovata stabilità sta però convincendo sempre più aziende ad usufruirne anche per progetti di dimensioni considerevoli.

- Debutta nel 2005
- C#, UnityScript (ispirato a Javascript) e Boo (ispirato a Python) come linguaggi di scripting

- Closed-Source (con la possibilità di ricevere il sorgente previo acquisto separato)
- Editor per Windows e Mac
- Utilizzato in molti giochi negli ultimi anni (come ad esempio Interstellar Marines)
- Performance e qualità di rendering vicine ma inferiori a quelle dei migliori engine quali Unreal Engine e Crytek.
- Disponibile gratuitamente a patto di non superare i 100k dollari di entrate annuali oppure per 75$ al mese
- Permette l'export per piattaforme: Windows, Mac OS, Linux, Android, iOS, BlackBerry 10, Windows Phone 8, PlayStation 3, PlayStation 4, PlayStation Vita, Xbox 360, Xbox One, Wii U, e Wii.
- Documentazione e community nella media

Illustrazione 2.6: Interstellar Marines, videogioco in sviluppo utilizzando Unity

CryEngine

Arrivato alla versione 3.6.12, realizzato da Crytek e fra i più potenti motori grafici al mondo, utilizzato in passato per alcuni dei giochi graficamente più all'avanguardia della loro generazione.

- Debutta nel 2004
- Linguaggio di scripting basato su Lua
- Closed-Source (con la possibilità di ricevere il sorgente previo acquisto separato)
- Editor per Windows
- Gran numero di giochi di successo realizzati con esso, come ad esempio Crysis
- Performance e capacità di rendering fra le migliori al mondo
- Attualmente disponibile per 10€ al mese
- Permette l'export per piattaforme Windows, iOS, Android, Xbox 360, Xbox One, PlayStation 3, PlayStation 4 e Wii U.
- Grossa community e documentazione

Illustrazione 2.7: Ryse Son Of Rome, sviluppato utilizzando il CryEngine

Come nel caso dell'Unreal Engine, anche questo motore viene utilizzato internamente dalla casa per sviluppare videogiochi.

ShiVa Engine

Al momento arrivato alla versione 1.9.2 (con la possibilità per i possessori di una licenza di utilizzare la 2.0 in beta), realizzato da Stonetrip e utilizzato principalmente in ambito mobile.

- Debutta nel 2007
- ShivaScript (basato su Lua) come linguaggio di scripting
- Closed-Source
- Editor per Windows (dalla versione 2.0 anche per Mac e Linux)
- Utilizzato per diversi giochi mobile, quali ad esempio Prince Of Persia 2
- Performance e qualità di rendering di poco inferiori a motori quali Unity
- Per prodotti commerciali richiede l'acquisto di una licenza, base (200$) o avanzata (400$)
- Permette l'export per piattaforme: Windows, Mac OS, Linux, Android, iOS, BlackBerry, Windows Phone, PlayStation 3,Xbox 360, Wii con PlayStation Vita in sviluppo.
- Documentazione sufficiente (ma con alcune lacune) e community piccola ma estremamente attiva.

Illustrazione 2.8: Babel Rising – Titolo sviluppato da Ubisoft usando ShiVa

Generalmente considerato la versione più indie di Unity, ShiVa offre un ambiente di sviluppo immediato ed è in grado di produrre prototipi più rapidamente dei suoi rivali.

“

Working on these projects helped us realize that UDK just was not for us, it was simply too big, too hard to learn and not flexible enough. This is where ShiVa really stands out. We installed it and were able to make a simple scene with some basic gameplay almost instantly

Rafał Kula – Rejected Games Dev.

”

Godot

Arrivato recentemente alla versione 1.0, sviluppato da OKAM Studio ed utilizzato principalmente in ambito mobile.

- Sviluppato ed utilizzato internamente dallo studio OKAM sin dal 2001, nel febbraio 2014 è stato rilasciato sotto licenza MIT
- GDScript (basato su Python) come linguaggio di scripting
- Open-Source
- Editor per Windows, Mac e Linux
- Utilizzato per diversi giochi mobile, quali ad esempio Black Velvet Bandit o Anthill
- Performance e qualità di rendering inferiori rispetto a Unity
- Completamente free
- Permette l'export per piattaforme: Windows, Mac OS, Linux, Android, iOS, PlayStation 3 e PlayStation Vita.
- Documentazione molto completa ma community estremamente piccola.

Anche se arrivato solo di recente alla versione 1.0, Godot è un motore completo ed in sviluppo da molti anni, il cui problema principale al momento risulta essere la community molto piccola e la quasi totale mancanza di progetti realizzati da case diverse dalla OKAM, questo porta ad una scarsità di esempi che rende l'imparare ad utilizzare questo motore piuttosto ostico.

Illustrazione 2.9: Black Velvet Bandit

La scelta

Alla fine la scelta del motore da utilizzare è ricaduta su ShiVa, le motivazioni di tale scelta sono le seguenti:

- Buon prezzo

  L'attuale offerta di 200$ per una licenza a vita della versione 2.0 significa che, con una piccola spesa iniziale, il team potrà usufruire di un engine sufficientemente avanzato (in ottica mobile almeno) da essere utilizzabile per diversi anni senza spese aggiuntive.

- Semplicità

  ShiVa è progettato specificatamente per progetti di piccole e medie dimensioni, a differenza dei suoi rivali, questo significa un approccio allo sviluppo più semplice ed immediato (a patto di non eccedere nella complessità del progetto), tutto questo si sposa perfettamente con la tipologia di gioco da noi scelta (i titoli adventure raramente contengono elementi molto complessi, preferendovi un gameplay più lineare ed immediato)

- Linguaggio di scripting

  Il linguaggio di scripting di ShiVa, ShiVa Script, è basato su Lua e ne eredita la semplicità ed immediatezza, estendendone le funzionalità con elementi quali gli stati o gli handlers.

- Community

  La community intorno a ShiVa è molto più piccola di quelle dei suoi concorrenti più blasonati, ma estremamente attiva e pronta ad aiutare i nuovi utenti, inoltre, sul forum ufficiale è possibile mettersi in contatto diretto con i suoi sviluppatori sempre disponibili ad aiutare con lo sviluppo di un'applicazione o ad accogliere richieste di nuove funzionalità per la successiva versione beta, in generale ShiVa offre quel tipo di supporto che solo un azienda con un numero ancora limitato di clienti è in grado di offrire, ed essendo questo il primo progetto di questo team, è esattamente ciò di cui ha bisogno.

2.2.2 ShiVa Engine

L'editor di ShiVa è un software di tipo WYSIWYG, composto da 21 moduli e studiato per lo sviluppo di qualunque genere di giochi ed applicazioni.

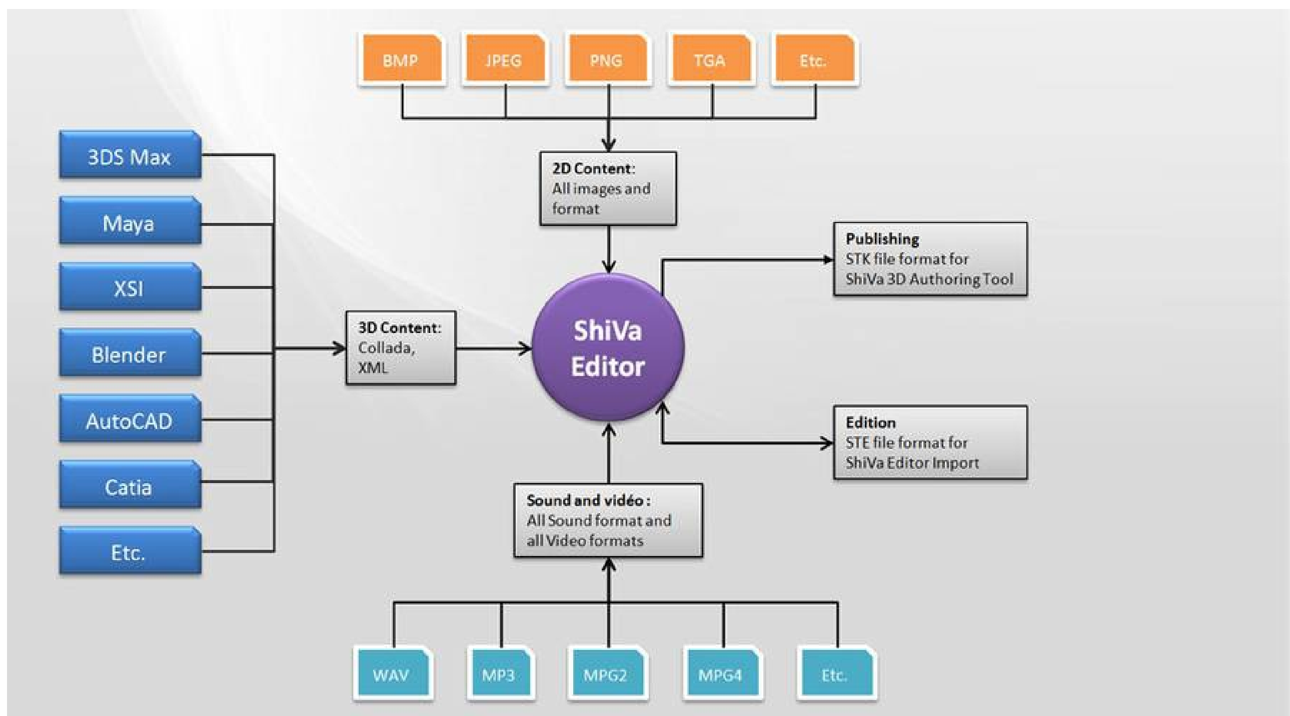

Illustrazione 2.10: I formati supportati da Shiva

Alcune delle sue caratteristiche principali sono:

Geometria

In grado di renderizzare fino a 15 milioni di triangoli per frame, LOD integrato e supporto per diversi formati quali Collada 3-4 ed Autodesk DWF.

Possibilità di creare terreni attraverso l'uso di informazioni geo-spaziali (GeoTIFF, NASA and IGN)

Gestione delle luci e delle ombre

Supporto per ombre dinamiche direzionali e omnidirezionali, fino a 32 luci per superficie.

Generatore di lightmap integrato con la possibilità di export\import.

Effetti particellari

Editor integrato per la generazione di effetti particellari, possibilità di realizzare effetti quali esplosioni, fuoco, pioggia, neve etc.

Shaders

Phong

Shader che utilizza l'interpolazione delle normali nella rasterizzazione dei poligoni al fine di ottenere migliori risultati nelle riflessioni speculari, il calcolo viene eseguito pixel per pixel, risultando estremamente accurato, ma computazionalmente molto più pesante del classico flat shading.

Gouraud

Shader utilizzato per ottenere un graduato cambio di colore simile a quello ottenuto tramite phong shading, a differenza di quest'ultimo però, i calcoli non avvengono per i singoli pixel della superficie ma per i suoi triangoli, così facendo il Gouraud risulta estremamente più leggero ma anche molto meno preciso.

Toon

Chiamato anche cel shading, è un sistema di rendering non foto realistico, finalizzato a far apparire le immagini renderizzate come se fossero disegnate a mano, con spessi bordi e evidenti tinte unite, avvicinandole così allo stile dei fumetti o dei cartoni animati.

Questo viene fatto renderizzando il modello con un numero limitato di tonalità per ogni colore, ottenendo così una maggiore piattezza dell'immagine.

Animazioni

Animazioni di tipo gerarchico per personaggi e creature, con numero di giunture utilizzabili illimitato e vertex influenzabili da quattro bones, possibilità di blending di 8 animazioni per volta.

HUD e 2D

UI builder integrato contenente i componenti più comuni quali: label, bottoni, editbox, listbox etc.

Effetti di transizione ed animazione per i componenti anche configurabili tramite HUD editor apposito.

Network

Codice server incluso per la gestione di piccole partite multiplayer (fino a 16 giocatori) con possibilità di acquisto di una licenza separata per avere accesso a server in grado di supportare migliaia di giocatori per volta.

Suono

Supporto sorround 5.1, possibilità di utilizzare 3D sound positioning e spazialization.

Physics

Motore fisico basato su ODE (Open Dynamics Engine) in grado di supportare collisioni fra mesh, corpi statici o in movimento, senza limiti al numero o alla dimensione delle facce e capace di gestire parametri quali massa, frizione, rimbalzo etc.

In grado inoltre, di simulare giunture quali l'asse di un'auto, gli ingranaggi di un motore, molle etc.

ShiVa Script

Il linguaggio di scripting adottato è basato su lua[7], di cui ne estende varie caratteristiche.

> *LUA*
>
> Linguaggio di programmazione multi-piattaforma sviluppato per lo scripting, dinamico, riflessivo ed imperativo che combina una sintassi procedurale di semplice descrizione dei dati con potenti costrutti detti table, basati su array associativi a semantica estensibile.
>
> LUA è dinamicamente tipizzato, esegue il codice interpretando bytecode da un registro basato su macchina virtuale e dispone di gestione automatica della memoria con garbage collection incrementale, che lo rende ideale per la configurazione, lo scripting e la prototipazione rapida del software.

ShiVa Script segue la sintassi di Lua e ne incorpora buona parte delle API per poi estenderne le funzionalità aggiungendovi funzioni specifiche al motore grafico, costanti, ed i concetti di handlers, states ed AIModels.

Definizione2.1:**AIModels.** Un AIModel può essere visto come una classe, esso funge da container per tutte la variabili, funzioni, stati ed handlers che lo caratterizzano, tramite esso possiamo organizzare il codice, gestire gli inputs e variarne il comportamento attraverso l'uso degli stati. Ve ne possono essere diverse istanze e può essere collegato a qualunque utente o oggetto in gioco, nel primo caso esso diventa una delle AI principali, venendo ciclato ad ogni iterazione, nel secondo invece la sua attività è legata a quella dell'oggetto.

Definizione 2.2: **Handlers.** Un handler è un ricevitore di eventi, un evento è un messaggio ad un handler di una specifica AI. Il messaggio può essere scambiato fra le AI di due oggetti distinti, fra un utente ed un oggetto (e viceversa), fra due utenti (nel caso di un gioco multiplayer), o ancora richiamato dalle stessa AI o da una diversa AI dello stesso oggetto.

Una volta ricevuto un messaggio, un handler si comporterà esattamente come una funzione, eseguendo il codice contenuto al suo interno.

Il comando con cui si lancia un evento determina il momento in cui esso verrà ricevuto, gli eventi possono essere istantanei (sendEventImmediate), messi in coda per essere eseguiti alla fine del frame corrente (sendEvent) ,o ancora, eseguiti dopo una determinata quantità di tempo (postEvent (nDelay, Event)).

Definizione 2.3: **States.** Un AIModel può essere gestito in maniera simile ad una macchina a stati finiti tramite l'uso degli stati, quando si definisce una AI è possibile definire un numero arbitrario di stati ed impostare come transiterà fra di essi, ogni stato è composto da tre funzioni, onEnter ed onLeave verranno eseguite al momento in cui una AI transiterà da uno stato ad un altro, mentre onLoop sarà la funzione richiamata ad ogni iterazione che definirà il comportamento della AI in quello stato.

Un esempio di uso ed interazione fra AIModels

Uno esempio molto semplice di come gestire il comportamento di un oggetto attraverso l'uso di AIModel può essere quello di modellare il comportamento di una Locked Box, scatola protetta da codice contenente oggetti necessari per procedere nell'avventura, il cui comportamento è definito da tre AI comunicanti fra loro.

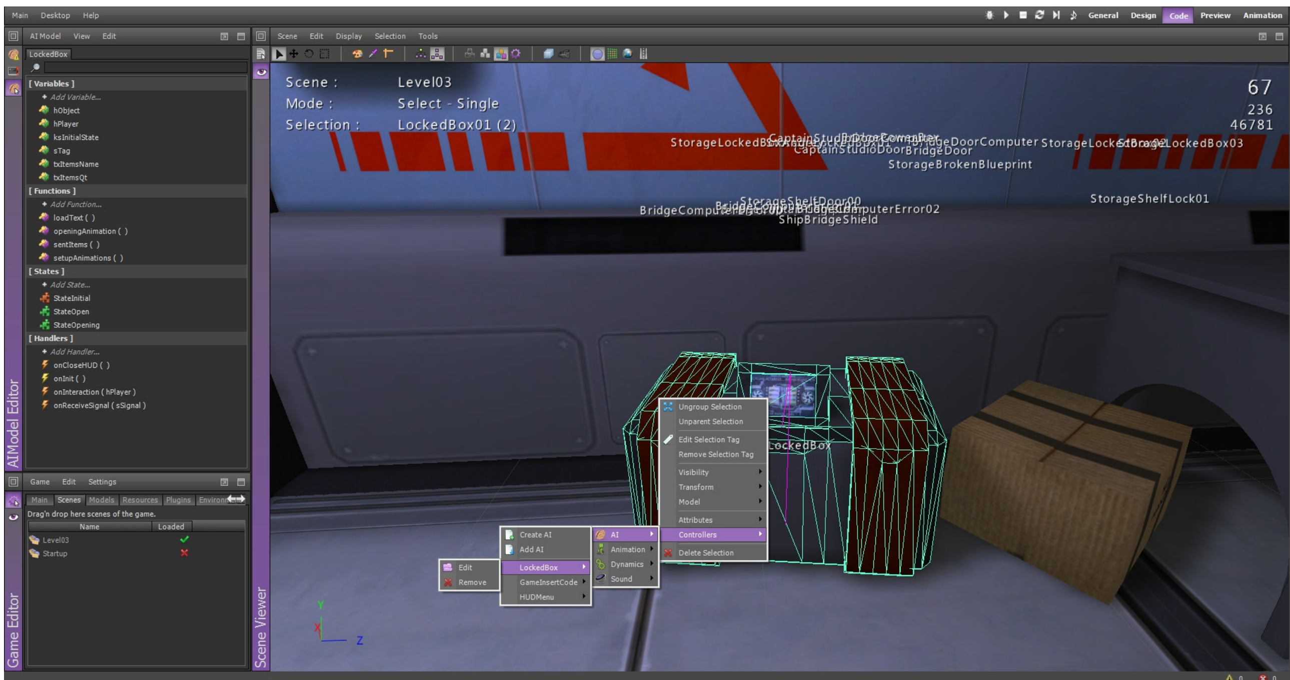

Illustrazione 2.11: Il modello su cui si interviene ed i suoi tre AIModel

La prima AI, LockedBox, gestisce l'interazione del giocatore con il modello e definisce quale oggetto riceverà il giocatore una volta aperta la scatola, i suoi stati principali sono: Initial, Opening e Open; se interagita durante lo stato Initial, invia un evento all'AI HUDMenu, avviandola.

La seconda AI, HUDMenu, è una AI standard, realizzata per gestire la creazione e navigazione dei menu 2D creati in gioco dai vari oggetti, essa si occupa di visualizzare a schermo le varie informazioni relative all'oggetto (trovate all'interno di un file xml utilizzando la tag dell'oggetto) e, se presente, avviare un sottogioco (in questo caso GameInsertCode).

L'ultima AI, GameInsertCode, permette al giocatore di inserire un codice in un numpad e, se il codice risulta corretto, informa, tramite un evento, la prima AI del fatto che il codice inserito è stato accettato, facendola così transitare nello stato Opening.

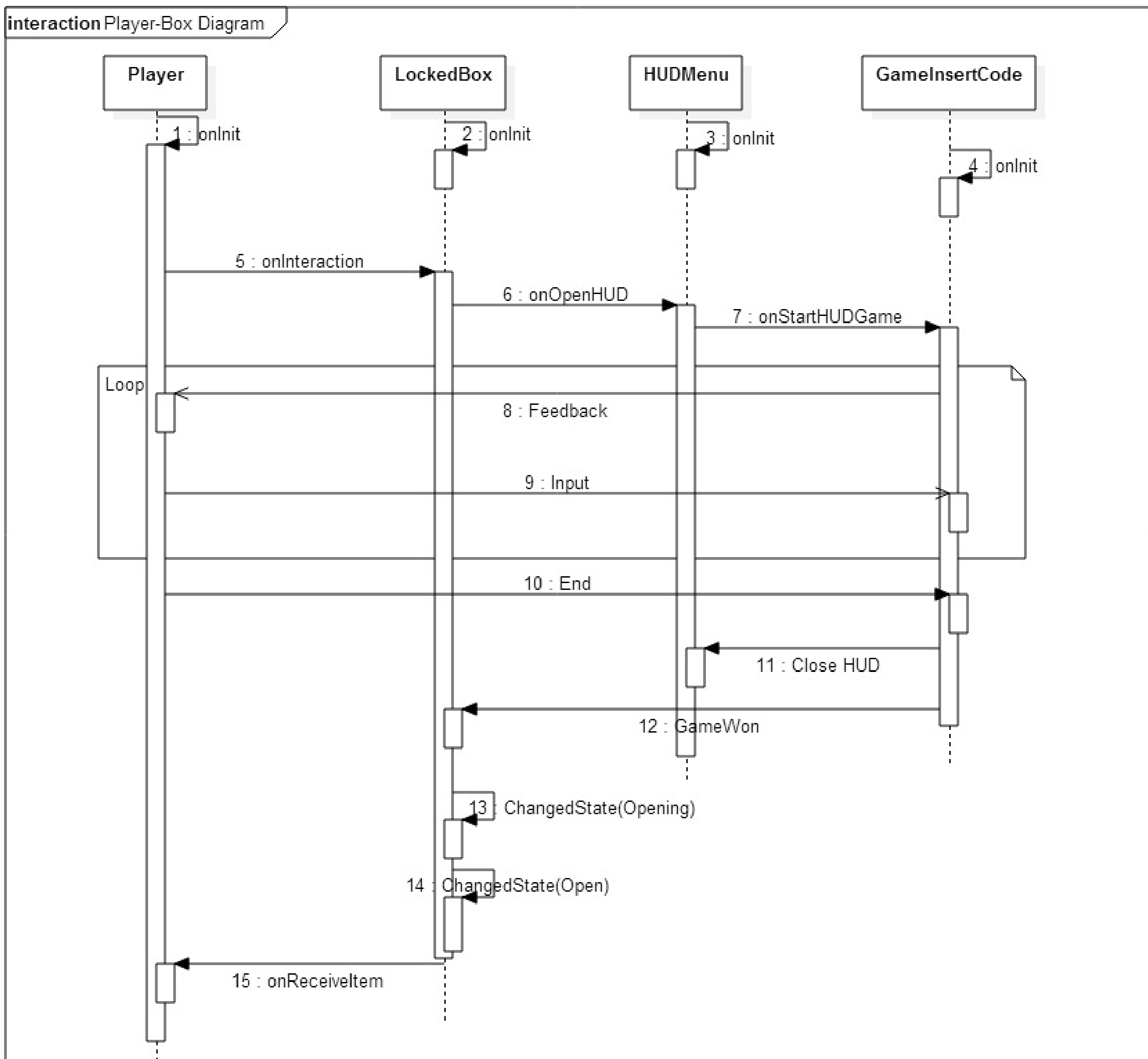

Illustrazione 2.12: Diagramma delle interazioni fra i tre AIModels della scatola ed il Player

2.2.3 Modellazione

Una volta deciso il game engine, la scelta del programma con cui realizzare i modelli 3D necessari in gioco è risultata semplice, questo perché la community di ShiVa supporta principalmente tre software di 3D modelling: Maya, 3D Studio Max e Blender; i primi due elencati però richiedono per uso commerciale l'acquisto di una licenza del costo di circa 4000 euro, mentre il terzo risulta gratuito, anche in questo caso, ci si è resi conto che le risorse del team non avrebbero permesso di sfruttare appieno i programmi più blasonati ed a pagamento, inoltre tutti i membri possedevano già una conoscenza a diversi livelli di Blender, mentre nel caso degli altri software la situazione risultava più frazionata.

Blender

Blender è un software gratuito di modellazione, rigging, animazione, compositing e rendering di immagini tridimensionali, sviluppato dalla Blender Foundation sin dal 2002.

Anche se non sofisticato e all'avanguardia quanto alcuni diretti rivali quali 3D Studio Max, Blender permette comunque lo sviluppo di modelli estremamente dettagliati e sempre più spesso viene considerato per lo sviluppo di film e videogiochi commerciali.

Illustrazione 2.13: Un fotogramma da Tears Of Steel, cortometraggio realizzato in Blender nel 2012

2.2.4 Character Creation

Per realizzare i modelli dei membri dell'equipaggio in tempi brevi si è deciso di affidarsi ad un software di character creation chiamato Fuse.

Questa tipologia di programmi permette di realizzare modelli estremamente dettagliati partendo da una serie di template predefiniti, comprendenti la maggioranza degli aspetti fisici più comuni, e modificandone poi i parametri al fine di adattarli al meglio al titolo che si sta sviluppando.

Fuse

Fuse è un software sviluppato da Mixamo di data-driven 3D charater creation, caratterizzato da un ampia scelta di template per ogni porzione del corpo, oltre che svariati capi di abbigliamento e texture dinamiche per personalizzare appieno il modello, permette inoltre il rigging automatico dei modelli e l'applicazione di svariate animazioni già integrate nel software o acquistabili attraverso lo store online.

I modelli così realizzati risultano estremamente dettagliati e possono essere esportati in svariati formati, permettendone un'ulteriore raffinamento all'interno di software di modellazione avanzata come Blender.

Il software è stato acquistato attraverso la piattaforma di digital delivering Steam ad un prezzo estremamente contenuto (100 euro).

Illustrazione 2.14: L'interfaccia di Fuse durante l'editing di un personaggio

2.2.5 *Altri software utilizzati*

Per le rimanenti risorse necessarie le scelte sono state le seguenti

Textures

Le maggioranza delle textures è stata acquisita sottoscrivendo un abbonamento ad un sito (Game Textures) specializzato in questo campo, adattandole poi alle nostre necessità e ripiegando sul crearle personalmente solo quando strettamente necessario, in entrambi i casi usando Gimp, un programma gratuito per la creazione e modifica di immagini digitali.

Audio

Per quanto riguarda l'audio in gioco, ci si è appoggiati a diversi siti quali ad esempio:

- Sound Bible

  Contenente una raccolta di suoni disponibili sotto diverse licenze ma principalmente gratuiti.

- Sound Image

  Contenente una raccolta di colonne sonore disponibili sotto licenza Attribution 4.0 International (CC BY 4.0), quindi completamente gratuito a patto di citarne l'autore nel prodotto che le utilizza.

Altri suoni sono stati registrati personalmente o creati tramite software free quali JFXR

3 CONCEPT DOC

In questo capitolo si è sviluppato il concept doc del videogioco, questa sezione serve a documentare tutte le caratteristiche del prodotto e descriverne le tempistiche di realizzazione previste.

3.1 High Concept

Adrift – The Cassini Accident

Un'avventura in cui il giocatore si sveglia a bordo di una nave spaziale alla deriva, completamente solo e senza memoria, armato unicamente del suo intelletto dovrà esplorare la nave e scoprirne i misteri, al fine di ritrovare i suoi ricordi, ed una via verso casa.

3.2 Genere

Il gioco è un ibrido fra il genere action, cioè con sezioni in cui vengono testati i riflessi del giocatore, ed adventure, in cui invece ragionamento logico e spirito di osservazione sono richiesti per progredire nella storia.

3.3 Gameplay

Il gameplay è composto da una serie di sottogiochi action ed adventure (sia bidimensionali che tridimensionali) e da parti esplorative in cui il giocatore deve percorrere la nave alla ricerca di indizi per far progredire la storia.

Un esempio di sottogioco action è il *Packet Routing*, in cui il giocatore si trova a “guidare” tramite un'interfaccia 2D un pacchetto su una connessione di rete danneggiata, nel tentativo di fargli raggiungere la sua destinazione.

Un esempio invece di un sottogioco più indirizzato al genere adventure è il *Notes Restoration,* in cui il giocatore si trova a dover ricostruire degli appunti strappati trascinandone i pezzi nella giusta posizione.

Nelle fasi esplorative infine il giocatore deve trovare determinati oggetti nascosti nell'ambiente 3D, la cui posizione viene suggerita da note sparse per il livello, esse inoltre raccontano avvenimenti accaduti sulla nave, andando lentamente a comporne la storia.

3.4 Features

- Stile grafico semplice ma accattivante, ispirato al genere Cyber Punk
- Gameplay a due e tre dimensioni
- Una storia ricca di particolari ed affascinante
- Decine di sottogiochi diversi con cui confrontarsi

3.5 Settings

Il gioco si svolge nel 24° secolo, interamente a bordo della nave spaziale da ricerca Cassini, il cui compito è quello di scoprire nuovi sistemi planetari e studiarne i pianeti, valutandone la possibilità di terraforming.
Il futuro raccontato in questo gioco è di stile cyberpunk, in cui la tecnologia è progredita sensibilmente, ma non il suo "design" rimasto in qualche modo legato al 21° secolo, gli ambienti di gioco saranno quindi metallici, di ispirazione industriale e la maggior parte dei componenti con cui interagirà l'utente richiederanno una soluzione "meccanica".

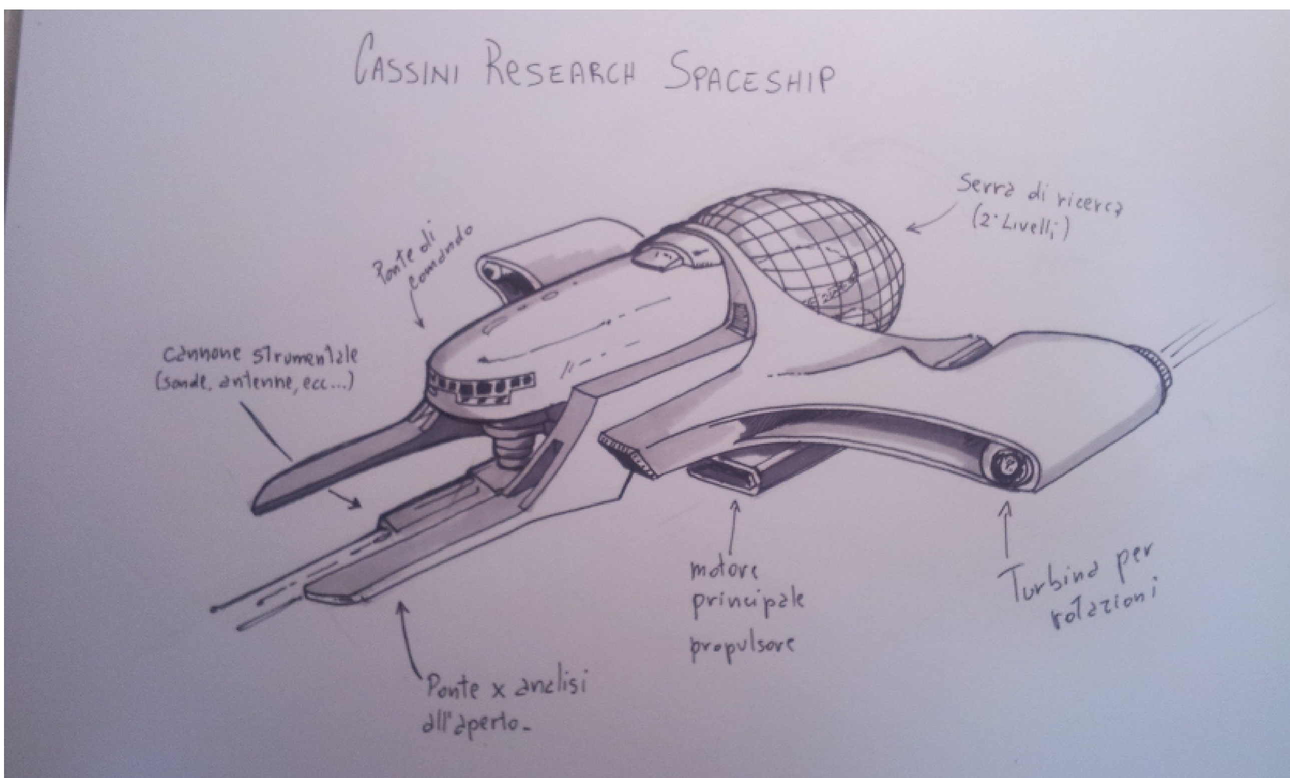

Illustrazione 3.1: Concept della nave

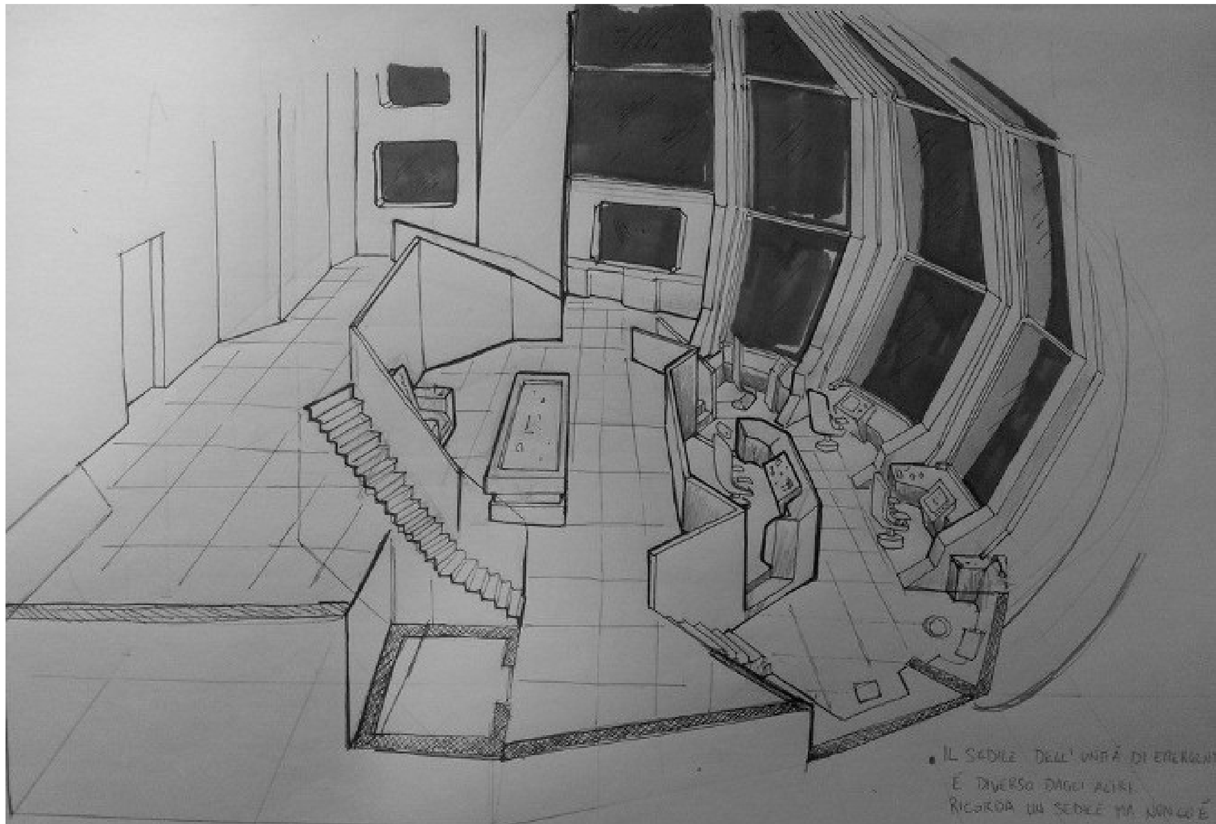

Illustrazione 3.2: Il ponte della nave

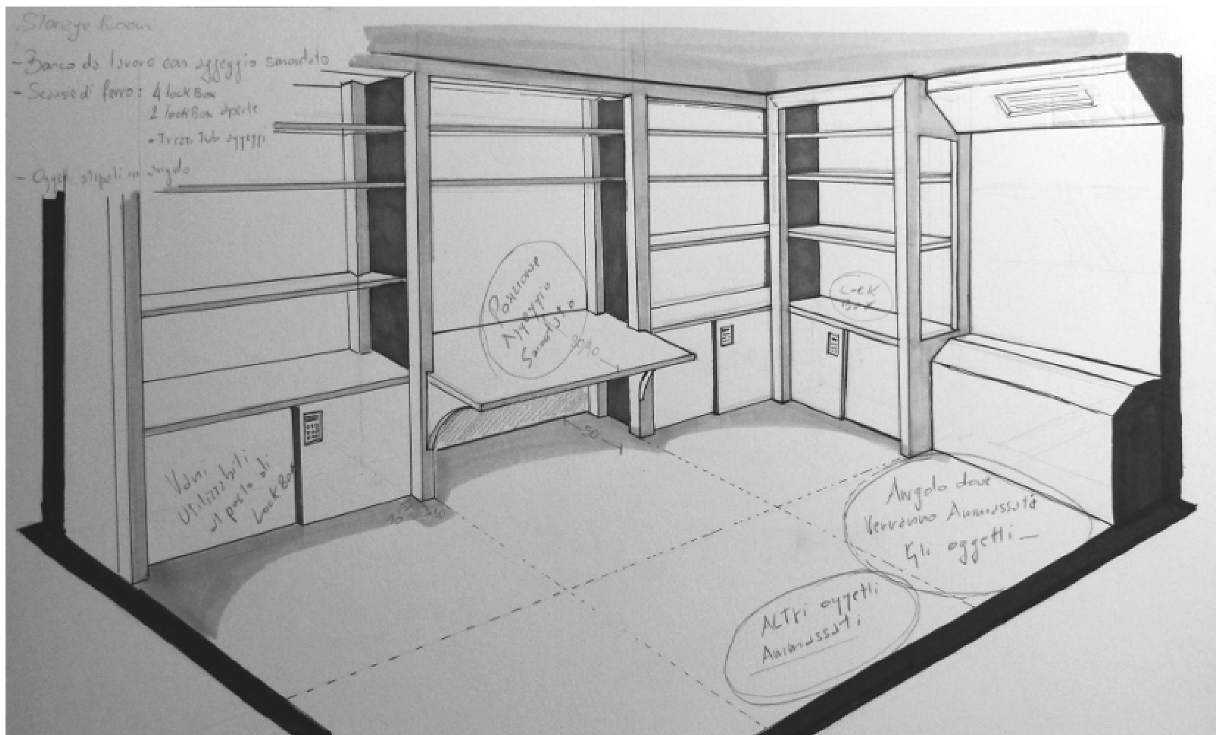

Illustrazione 3.3: Uno degli ambienti di gioco

3.6 Story

Durante l'esplorazione di un sistema planetario recentemente scoperto l'equipaggio della Cassini rileva una nuova tipologia di radiazioni emanate dalla stella del sistema e decide di avvicinarsi per studiarne meglio la natura, durante questi studi scopre però che le radiazioni sono nocive per l'essere umano e, per evitare di soccomberne, sceglie di entrare in ibernazione e lasciare alla nave il compito di invertire la rotta e tornare ad una distanza di sicurezza.
Ciò che però l'equipaggio non sa è che le radiazioni interferiscono anche con i sistemi di bordo, la nave quindi non inverte la rotta, e, ormai catturata dalla sua gravità, prosegue verso la stella.
Il protagonista si sveglia qualche settimana dopo questi eventi, senza alcuna memoria, la nave è alla deriva, sempre più vicina alla stella ed ha subito diversi danni, il resto dell'equipaggio è ancora in ibernazione e non è possibile risvegliarlo a causa del protocollo di sicurezza medico in funzione. Lentamente il protagonista inizierà ad esplorare la nave e grazie a notepad sparsi per gli ambienti di gioco comincerà a ricostruire la storia dell'equipaggio della Cassini, scoprirà così i nomi ed i ruoli dei suoi componenti:

- **Alastair:** Capitano della nave, amante dell'esplorazione e della scoperta, nella sua cabina si trovano mappe navali, terrestri e spaziali, libri e modellini di vascelli di ogni era.
  Ama gestire la sua nave con libertà, le formalità sono evitate ma allo stesso tempo si aspetta sempre il massimo dell'impegno da ogni membro dell'equipaggio. Durante le fasi concitate a seguito della scoperta della nocività delle radiazioni è stato l'ultimo ad entrare in ibernazione, volendosi assicurare prima che tutti i sistemi fossero a norma, purtroppo questo può avergli causato gravi danni alla salute, che solo un esame più approfondito, una volta tornati al centro di comando, potrà identificare appieno.
- **Andrey:** Ingegnere di bordo, si occupa della manutenzione della nave, nella sua cabina troviamo schematiche di navi e componenti, disegni e dischi (è un'amante della musica rock), passa buona parte del tempo libero ad argomentare con Dors, come emergerà dalle varie note sparse per la nave, sente la mancanza di casa e sta considerando di abbandonare l'esplorazione spaziale per passare più tempo con la sua famiglia.
- **Zoe:** Medico e biologo di bordo, è amante del restauro e delle piante, nella sua cabina troviamo piante esotiche e vecchi oggetti (violini, quadri...teschi), il suo compito è quello di studiare le reazioni di uomini e piante presenti nella serra della Cassini all'esposizione della luce di una nuova stella, al fine di identificare eventuali pericoli alla colonizzazione di un sistema.

- **Dors:** Ricercatrice di bordo, il suo compito è quello di analizzare eventuali anomalie incontrate durante l'esplorazione e mantenere operative le sonde utilizzate per l'esplorazione dei nuovi pianeti, nella sua cabina troviamo: pezzi di androidi, libri di scienza, tablet e computer, appunti sparsi un po' ovunque.

Con il procedere dell'avventura il giocatore verrà a conoscenza della nocività delle radiazioni e dedurrà da questo che oramai per lui è troppo tardi per salvarsi, ma capirà anche come solo le sue azioni possono salvare il resto dell'equipaggio.
Nelle fasi finali del gioco la vista e la velocità del giocatore diminuirà, portando l'utente a collegare questo ai sintomi delle radiazioni.
Una volta effettuata l'ultima riparazione necessaria ed invertita la rotta si avvierà una cut-scene finale che vedrà il giocatore muoversi lentamente verso le cabine criogeniche e finalmente aprirle, cadendo poi al suolo, l'equipaggio, una volta risvegliato non lo degnerà di alcuna attenzione, iniziando invece a discutere della situazione e di cosa farà una volta tornato al centro di comando, l'ultima frase che comparirà a schermo prima dei titoli di coda sarà:
"Ehi Dors, mi sa che il tuo androide è andato..."

3.7 Target Audience

La target audience per questo gioco si differenzia in base alla piattaforma

Mobile

Nel caso dei dispositivi mobile, l'obiettivo è quello di catturare l'utente medio interessato a giochi graficamente attraenti, e che potrebbe vedere i vari sottogiochi di Adrift come una serie di puzzles da affrontare in piccoli ritagli di tempo durante la giornata, il modello economico scelto è quello di offrire il gioco gratuitamente ma con pubblicità, permettendo però l'acquisto, in uno store interno all'applicazione, di una chiave per rimuovere la pubblicità ed ottenere svariati miglioramenti, come ad esempio maggior flessibilità negli enigmi.

PC

Nel caso dei computer invece l'obiettivo è quello di interessare quella fetta di utenti appassionati di giochi di esplorazione\avventura, in questo caso, sarà la storia intricata e l'ambientazione piuttosto particolare su cui si punterà maggiormente durante la fase di marketing.

Il gioco verrà rilasciato attraverso i principali servizi di digital delivering quali Steam o GOG ed a differenza della sua controparte mobile, sarà a pagamento e completo di tutto.

3.8 Hardware Platform

La leading platform per cui si svilupperà sarà Android, per quanto riguarda le prestazioni si renderà il gioco fruibile su dispositivi di livello pari o superiore a quello di un Galaxy tablet 8.9 e con uno schermo di almeno 5 pollici.

Requisiti minimi:

- OS: Android 3.0
- CPU: Dual-core 1 Ghz
- GPU: ULP GeForce o equivalente
- RAM: 1Gb

Requisiti raccomandati:

- CPU: Quad-core 2.5 Ghz
- GPU: Adreno 330 o superiore
- RAM: 2Gb

Successivamente si procederà alla release per PC, dando priorità a Windows ed in seguito, se considerato finanziariamente vantaggioso a Mac Os e Linux, prima del rilascio il gioco sarà rivisto e migliorato tenendo conto delle maggiori prestazioni di questa piattaforma rispetto a quella mobile, in questo caso i requisiti saranno:

Requisiti Minimi:

- OS: Windows XP o superiore
- CPU: Dual-core 2.5 Ghz
- GPU: Radeon 3850 o equivalente
- RAM: 2Gb

Requisiti raccomandati:

- CPU: Quad-core 3 Ghz
- GPU: Radeon 6850 o superiore

- RAM: 4Gb

In seguito si considererà se allargare la release ad altre piattaforme quali iOS, Windows Phone e la settima generazione di console.

3.9 Estimated Schedule

Il tempo totale necessario allo sviluppo e release del prodotto per la prima piattaforma è stimato essere di circa 11 mesi, le principali fasi saranno.

- **Studio dell'ambiente di sviluppo - 1 mese**

  Studio di ShiVa e del suo workflow nonché realizzazione del primo prototipo, in questa fase si confermerà se la scelta del game engine è stata corretta e si testeranno le prime idee di gameplay.

- **Sviluppo dell'architettura - 5 mesi**

  Sviluppo dell'architettura necessaria a sostenere il progetto, in questo periodo verranno implementate tutte le funzioni che si prevede saranno necessarie al gioco, durante l'intera fase si procederà alla creazione di prototipi ad intervalli regolari (2 settimane) che verranno provati da un ristretto gruppo di tester.

- **Demo - 1 mese**

  Realizzazione della prima demo giocabile, il prodotto di questa fase sarà la prima versione del gioco considerabile aperta al pubblico, nella demo non dovranno esserci placeholder di alcun genere, l'esperienza dovrà durare almeno 30 minuti e dare all'utente un assaggio di tutto ciò che troverà nel prodotto completo.

- **Alpha - 3 mesi**

  In questi tre mesi si lavorerà al completamento del gioco in ogni suo aspetto, conclusa questa fase avrà inizio la fase di testing più ampia.

- **Beta - 2 settimane**

  Il gioco completo sarà testato in ogni suo aspetto da un numero di tester maggiore che in precedenza, la conclusione positiva di questa fase segnerà l'ingresso nel progetto in code freeze e darà il via alla release.

- **Release - 2 settimane**

  Nelle due settimane finali ci si occuperà di preparare il gioco alla release.

Durante tutte le fasi il progetto verrà mantenuto funzionante per entrambe le piattaforme inizialmente scelte (Android e Windows), dando però priorità a quella mobile.

3.10 Competitive Analysis

Non esistono molti prodotti che hanno scelto lo stesso bilanciamento di Adrift fra azione e avventura e fra gioco 3D e 2D, questo gli permetterà di distinguersi maggiormente e di attirare quei giocatori interessati ad un approccio diverso dai soliti giochi di avventura, Adrift potrà ad esempio essere la scelta giusta per quegli utenti interessati ad un gioco che offra maggiore libertà di movimento ed esplorazione rispetto a titoli più statici quali “J.U.L.I.A.: Among The Stars”, oppure, risultare la scelta migliore per utenti che invece desiderano parti action nel loro titolo ma non ampie come ad esempio accade in “Consortium”.

Illustrazione 3.4: J.U.L.I.A.: Among The Stars

J.U.L.I.A.: Among The Stars

Questo titolo fa parte della tipologia di giochi di avventura soprannominata “Point & Click”, in cui il movimento dell'utente è definito da una serie di telecamere statiche ed il gioco procede di sottogioco in sottogioco senza offrire la possibilità di esplorare l'ambiente liberamente.

Consortium

Questo titolo invece è il risultato di un mix dei generi adventure ed action, ma in questo caso il secondo risulta predominante, il giocatore si troverà ad affrontare problematiche e sfide logiche, ma al contempo sarà costretto a confrontarsi con parti di gioco di ispirazione FPS (first person shooter).

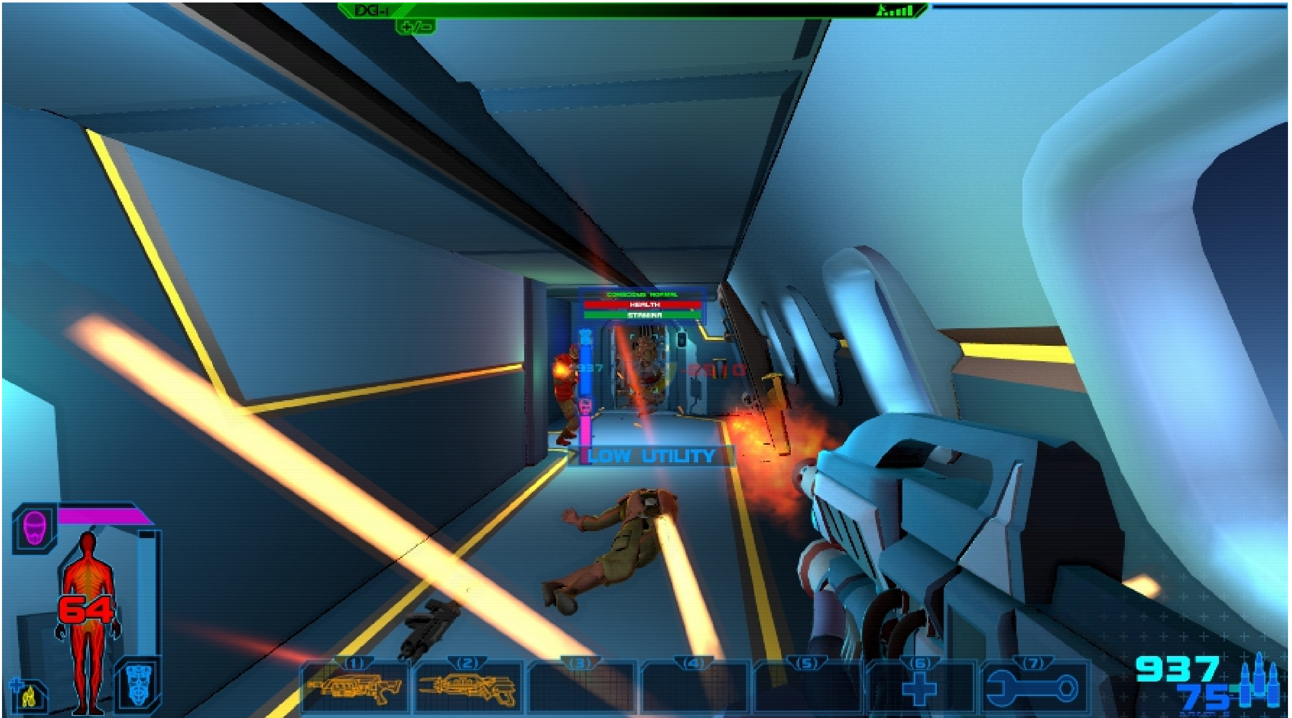

Illustrazione 3.5: Consortium

4 PROGETTAZIONE E SVILUPPO

In questo capitolo si descrive come è proceduta la realizzazione del titolo, partendo dalla fase di studio dell'ambiente di sviluppo fino ad arrivare alla creazione della demo.

Si descrive poi come l'approccio agile e la prototipazione iterativa ne abbiano influenzato la realizzazione, per poi concludere con alcuni screenshot del prodotto.

4.1 Studio dell'ambiente di sviluppo

Per prima cosa è stato necessario studiare l'ambiente di sviluppo e sperimentare le varie tecnologie messe a disposizione da ShiVa, realizzando una serie di prototipi.

ShiVa si è dimostrato all'altezza delle aspettative, sia per quanto riguarda semplicità di utilizzo che potenza, fornendo inoltre un serie di tutorial e codici di esempio decisamente chiari, che hanno permesso in poco tempo di acquisire le conoscenze fondamentali per poter iniziare a sviluppare Adrift.

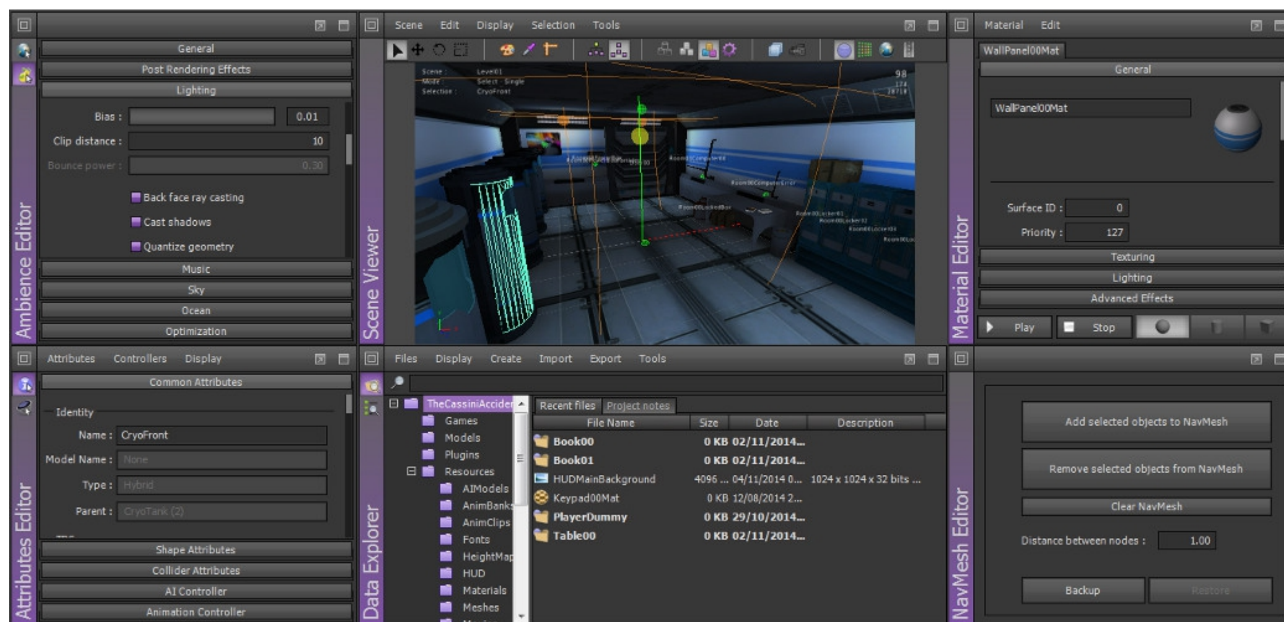

Illustrazione 4.1: L'editor di ShiVa durante lo sviluppo del primo prototipo di Adrift

La realizzazione di vari prototipi si è rivelata importante non solo per verificare la fattibilità di determinate idee, ma anche, per testare la capacità delle piattaforme target di supportarle, permettendo di identificare quelle tipologie di gameplay non applicabili a causa di problemi di prestazioni.

L'esempio più eclatante è sicuramente quello che riguarda il metodo di illuminazione utilizzato in gioco. Inizialmente era previsto un sistema di luci dinamiche, luci che, in alcuni casi, si sarebbero accese e spente per simulare problemi alla nave, o rendere la ricerca di determinati oggetti più complicata. Per verificarne la fattibilità si è creata una stanza e la si è popolata con alcuni oggetti ed una luce dinamica, il risultato di questo primo test ha immediatamente evidenziato come tutte le piattaforme mobile, salvo quelle di ultima generazione, non risultavano in grado di sostenere un frame rate adeguato appena si impiegava anche solo una fonte di illuminazione dinamica.

Questo ha portato all'implementazione di un sistema di luci statiche (utilizzando delle lightmap per simulare le ombre), ed alla modifica di alcuni aspetti del gioco, ad esempio, per rappresentare problemi alla nave si è sostituito al lampeggiamento delle luci, la presenza di fumo ed il rumore di contatti elettrici sfrigolanti.

Qui di seguito troviamo un confronto fra una luce dinamica ed una statica, la prima proietta ombre meno definite ma in grado di adattarsi allo spostamento degli oggetti, essendo calcolate ad ogni iterazione del motore grafico, la seconda invece viene calcolata durante lo sviluppo del gioco e produce una texture chiamata light map, che viene applicata sopra gli oggetti simulandone gli effetti di luce\ombra (con un impatto prestazionale praticamente nullo), le ombre così ottenute risultano molto più definite e realistiche, ma sono anche statiche, cioè, allo spostarsi di un oggetto, l'ombra non si aggiorna, rendendo questo approccio inadatto agli oggetti dinamici.

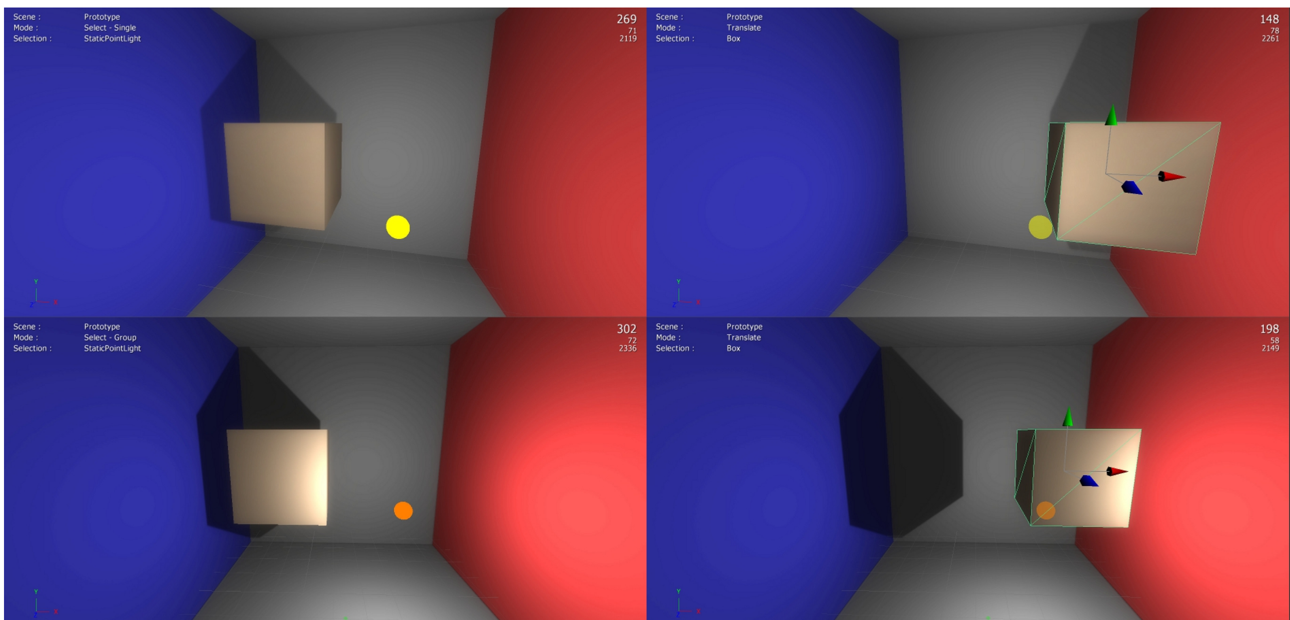

Illustrazione 4.2: Una luce dinamica ed una statica (con relative ombre) a confronto

Un altra tipologia di gameplay prevista era costituita da sezioni in cui il giocatore, armato di una torcia sarebbe stato costretto ad esplorare ambienti bui alla ricerca di oggetti o interruttori con cui interagire. L'implementazione più logica per una sezione del genere avrebbe richiesto l'uso di una luce dinamica, il cui utilizzo era però stato escluso dal prototipo precedente. Per ovviare a questo si è deciso di provare a simulare il funzionamento di una torcia andando ad alterare le proprietà dei materiali "illuminati", questo si è fatto implementando una classe "torcia", in cui, ad ogni iterazione un raycast viene lanciato nella direzione in cui il modello della torcia è indirizzato. Al momento della collisione con un oggetto, la classe ne acquisisce il materiale colpito e ne aumenta l'intensità del colore simulandone così l'illuminazione.

Se per un semplice prototipo i risultati sono stati incoraggianti, mostrando un impatto prestazionale praticamente nullo e simulando piuttosto bene il lavoro di una vera torcia, ci si è accorti che per ottenere gli stessi risultati su ambienti di gioco complessi sarebbe stato necessario introdurre un nuovo vincolo durante la modellazione degli oggetti in gioco, questo perché, con questa tecnica, si varia la colorazione dell'intero materiale colpito dal raycast, e non solo la parte interessata dalla finta torcia; dato che ShiVa permette l'applicazione di un solo materiale per modello, richiedendo la generazione di un sotto-modello per ogni materiale extra inserito, per avere una simulazione credibile ogni modello sarebbe dovuto essere composto da svariati sotto-modelli di piccole dimensioni, questo per evitare situazioni come ad esempio, puntare la torcia ad un angolo di una parete e vederla illuminare completamente perché composta da un solo modello.

Rispettando questa regola durante la modellazione dell'ambiente di gioco si sarebbe ottenuta una struttura composta da un numero di modelli molto più alto, danneggiando conseguentemente le prestazioni, dato che, uno dei limiti principali nelle piattaforme mobile è il numero di draw call sostenibili ad ogni iterazione, per questa ragione si è preferito abbandonare, almeno in ottica mobile, l'idea di utilizzare una torcia.

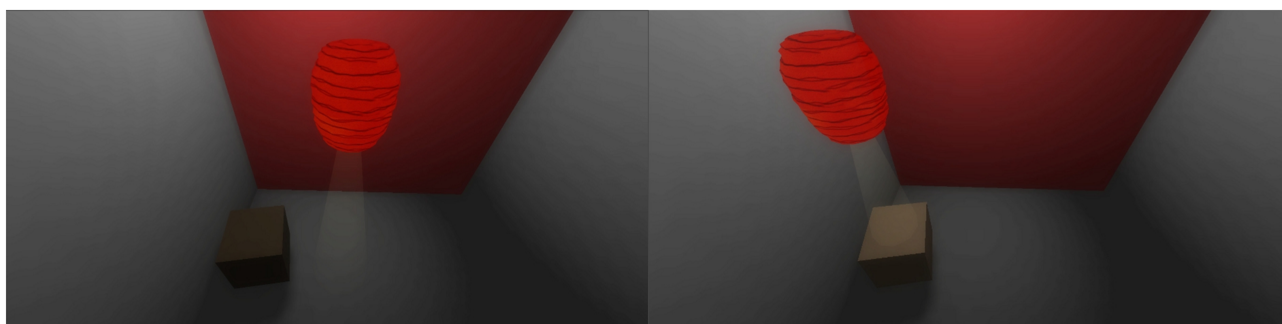

Illustrazione 4.3: La finta torcia in funzione

4.2 Analisi delle prestazioni per piattaforma

Lo sviluppo dei primi prototipi ha inoltre aiutato ad approfondire le differenze prestazionali fra le due piattaforme.

Esclusi i dispositivi mobile più moderni, la maggior parte dei tablet e cellulari prodotti risulta avere una GPU mediamente inferiore alla CPU, questo perché, probabilmente, al tempo della loro produzione la tecnologia non era ancora riuscita a raggiungere le richieste del mercato per quanto riguarda la qualità grafica richiesta.

I principali limiti identificati nei dispositivi mobile rispetto ai personal computer sono stati:

- **Numero inferiore di draw call totali:** Il numero di draw call indica quanti modelli separati devono essere stampati a schermo in un singolo frame, ciò significa che non è influenzato dalla complessità o dalle dimensioni dei modelli, ma semplicemente dal loro numero.
- **Effetti di post processing:** La maggior parte dei dispositivi mobile in grado di supportare unicamente OpenGL ES 1 e 2 mostrano un calo drastico di prestazioni nel momento in cui si fa uso di effetti quali bloom, blur o anche semplicemente il cambio della gamma.
- **Dimensioni delle texture:** La maggior parte dei dispositivi mobile riesce a supportare, senza drastici cali di prestazioni, textures fino a 1024x1024 pixel, a differenza dei PC in cui i giocatori sono oramai abituati a risoluzioni di 2048×2048 o addirittura 4096×4096.
- **Texture supportate:** Va anche considerato che l'impossibilità di utilizzare luci dinamiche determina anche l'inutilizzabilità delle texture di tipo normal map, che, come spiegato in precedenza, permette di aumentare il numero di dettagli di un modello senza la necessità di definirli tramite ulteriori facce.

4.3 Scelta della leading platform

La produzione dei primi prototipi ha evidenziato come la profonda differenza fra le piattaforme target impedisca lo sviluppo di un solo prodotto identico su entrambe, ma in grado di utilizzarne appieno le rispettive caratteristiche, facendo così sorgere la necessità di effettuare interventi specifici per le diverse piattaforme.

In parte questa necessità è stata minimizzata attraverso l'implementazione nel codice di controlli che modifichino alcune caratteristiche del gioco in base alla piattaforma su cui viene lanciato, ad

esempio, andando ad incrementare il numero di effetti particellari utilizzati se il sistema operativo rilevato sia Windows rispetto che Android.

Ciononostante si è rivelato necessario eleggere una piattaforma principale, su cui concentrarsi inizialmente al fine di sfruttarne appieno le caratteristiche e rilasciare il gioco per essa, per poi, in un secondo momento, effettuare le necessarie ottimizzazioni sulla seconda.

La piattaforma eletta come principale è stata quella mobile, il cui mercato, molto più giovane rispetto a quello della piattaforma PC risulta più ricettivo ai nuovi prodotti, inoltre, avendo questa piattaforma prestazioni molto più limitate, risulta più semplice realizzare un prodotto in linea con la qualità media sul mercato.

Una volta effettuata la release su questa piattaforma si potrà utilizzare l'esperienza maturata per migliorare il gioco, implementando tutte quelle caratteristiche grafiche e di gameplay che non era stato possibile inserire in precedenza, rendendo così il prodotto competitivo anche nel mercato PC.

Ciononostante risulta imperativo che l'intero progetto mantenga la sua natura multi-piattaforma in ogni fase, al fine di non effettuare scelte di gameplay o altro, che porterebbero ad alienare completamente la seconda piattaforma target.

4.4 Analisi dei requisiti e progettazione

Attraverso l'analisi del concept doc si è poi definita l'architettura del sistema, per la cui progettazione si è optato per un approccio misto fra Top-Down e Bottom-Up[8], preferendo il primo al momento di definire le caratteristiche principali del sistema, quali le entità coinvolte e le interazioni fra di esse, ed il secondo al momento di sviluppare le meccaniche singole delle interazioni.

Il sistema si è rivelato composto dal sottosistema del player (l'avatar del giocatore all'interno dell'applicazione) e dalle sue interazioni con i vari sottosistemi degli oggetti con cui può interagire.

E' poi emerso come vi siano due tipologie di comunicazioni principali:

- Fra il giocatore e gli oggetti in gioco
- Fra i vari oggetti in gioco

Viste le caratteristiche precedentemente descritte degli AIModels si è deciso di realizzare ogni oggetto interattivo in maniera simile ad una macchina a stati finiti, e di gestirne le interazioni attraverso uno scambio di messaggi mediante l'uso di handlers predefiniti.

Dopo aver realizzato un prototipo composto da una prima rappresentazione primitiva del giocatore

e diversi oggetti interattivi, si è constatato come il numero di stati in cui tali oggetti si possono trovare risulti piuttosto limitato, ragion per cui si è deciso di realizzare un singolo AIModel per la gestione della maggior parte degli oggetti in gioco, definendone il comportamento attraverso più file XML.

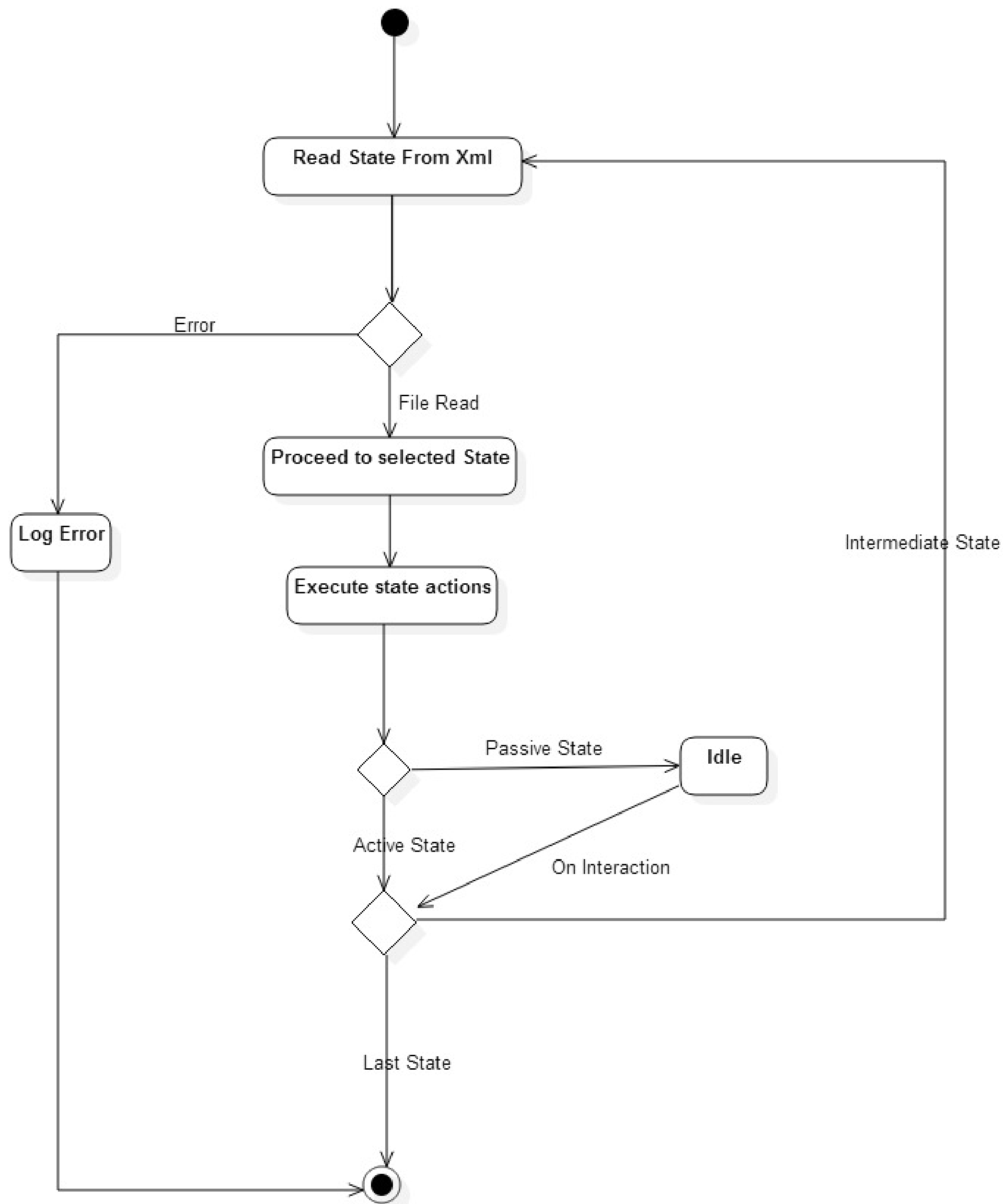

Illustrazione 4.4: State diagram alla base del comportamento degli oggetti in gioco

Durante questa fase è emerso anche come alcune caratteristiche dell'engine non risultassero sufficientemente sviluppate per soddisfare i requisiti, introducendo così la necessità di estenderle, questo ha portato alla realizzazione di sette main AIModels:

- **Comp2dAI:** Adibita all'estensione delle funzionalità 2D del motore
- **ControllsAI:** Per la gestione degli input dal giocatore nei vari stati in cui si può trovare il gioco
- **CutSceneAI:** Per la gestione delle cut scenes in gioco
- **JoypadEmulatorAI:** Attivata unicamente nel caso in cui sia rilevato uno schermo touch; quel che fa è tradurre i vari input ricevuti dallo schermo in un formato compatibile con ControllsAI
- **MainAI:** AI per la gestione degli stati del gioco e per la coordinazione delle altre AI principali
- **MenuAI:** Per la creazione e gestione delle pagine del menu (diverse in base alla piattaforma)
- **SoundAI:** Per la gestione dei suoni e delle musiche in gioco

Gestione delle interazioni fra i sottosistemi.

Si è deciso di standardizzare le comunicazioni fra i vari sottosistemi, definendo un numero limitato di handlers a cui ricondurre le interazioni più comuni in gioco.

- OnInteract: utilizzato dal giocatore per informare un oggetto che si sta avviando un interazione con esso.
- OnReceiveSignal: per gestire lo scambio di messaggi fra i vari sottosistemi, quando richiamato questo handler prende in ingresso uno o più parametri, i quali definiscono il tipo di messaggio ed eventualmente il dato trasferito con esso.
- OnStartHUD\StopHUD: per avviare o terminare le interfacce 2D

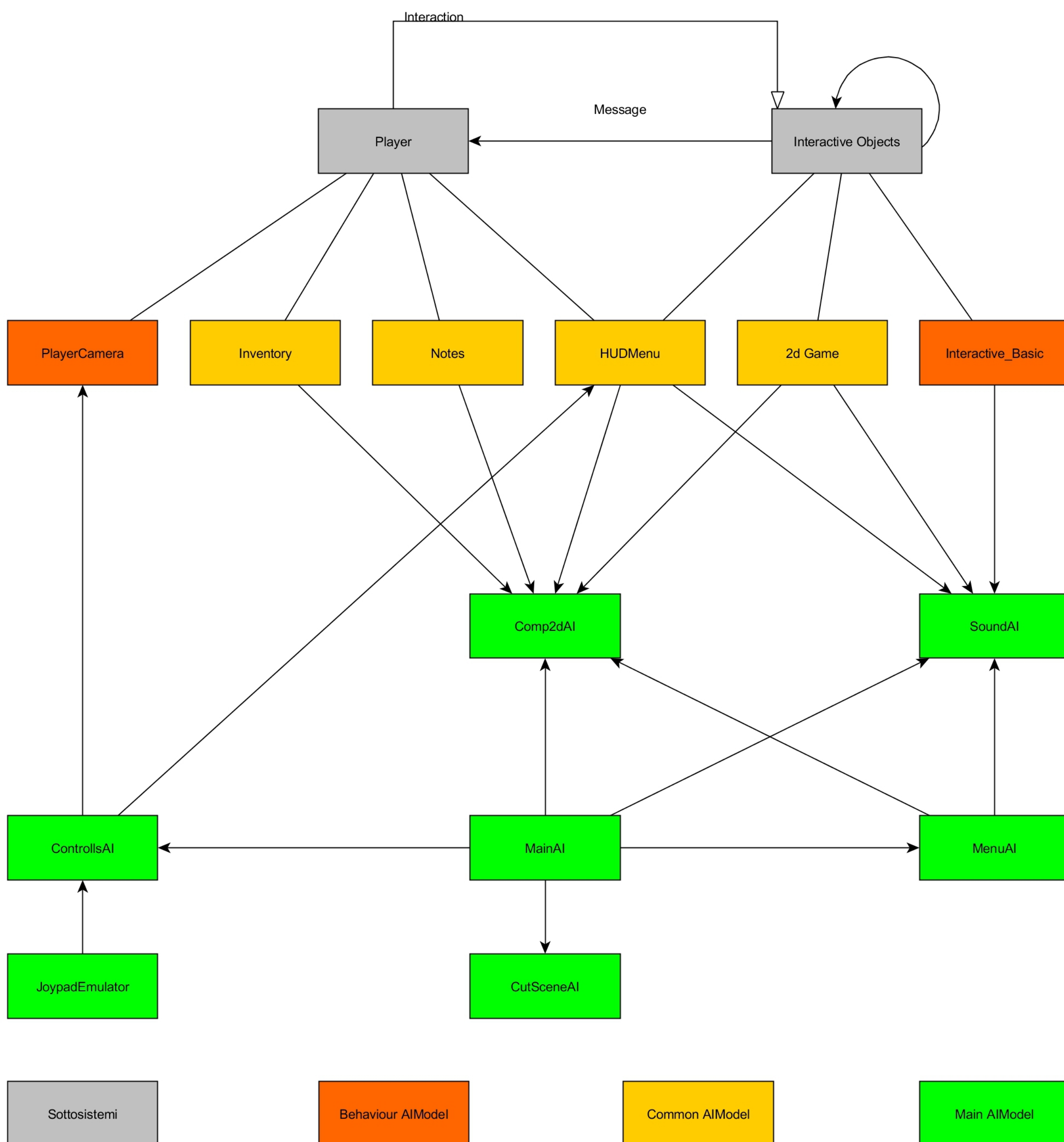

Illustrazione 4.5: Schema rappresentante le interazioni fra i due sottosistemi principali, gli AIModels da cui sono composti, e i sette main AIModels introdotti

4.5 Sviluppo

In questa fase ci si è concentrati sullo sviluppare il primo ambiente di gioco (la stanza iniziale) per poi adibirlo ad area di test e, mano a mano che la storia ed il gameplay del gioco venivano definiti, testare al suo interno le varie risorse create.

4.5.1 Definizione dei comandi

Come detto in precedenza, le due piattaforme target utilizzano tipologie di input differenti, perciò, si è rivelato necessario trovare un modo per gestire l'interazione del giocatore in maniera il più simile possibile, indipendentemente dalla piattaforma utilizzata.

Nelle parti di gioco 3D, per muovere il giocatore si è deciso di eleggere una periferica come preferita (in questo caso un gamepad), andando poi a simularne l'uso nel caso di periferiche differenti.

- **Touch**

  Per i dispositivi touch si è proceduto realizzando una classe extra, al fine di simulare un gamepad tramite il tocco di determinate zone dello schermo.

  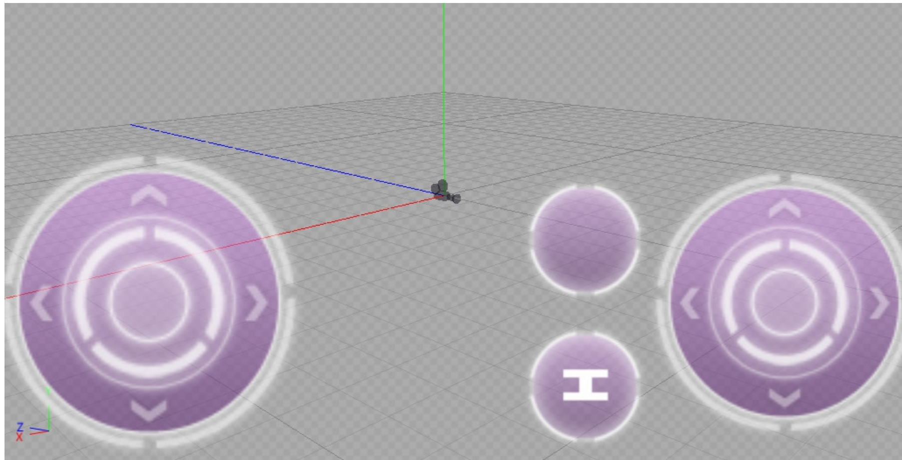

  Illustrazione 4.6: Lo schermo touch implementato

  Il toccare lo schermo all'interno dei due cerchi più grossi equivale al premere uno stick analogico di un gamepad nella direzione in cui sta avvenendo il tocco, allo stesso modo, il tocco dei due cerchi più piccoli simula invece la pressione di due tasti.

- **Keyboard & Mouse**

  Nel caso di mouse e tastiera invece si è proceduto adibendo la prima alla gestione dello stick del movimento e dei bottoni, e lasciando al mouse il compito di simulare lo stick destro.

Le parti di gioco in 2D invece si sono rivelate più semplici da accomunare, ciò che si è fatto è stato gestire il touch dello schermo come se fosse un mouse.

4.5.2 Creazione del corpo fisico del giocatore

Una delle prime cose fatte per poter iniziare a navigare l'ambiente di gioco e testare le funzioni scritte è stato realizzare il corpo fisico del giocatore.

Per corpo fisico si intende l'entità collegata ad un modello, (caratterizzata da attributi quali: peso, forma, frizione etc.) utilizzata per introdurre quest'ultimo all'interno della simulazione della fisica di gioco.
All'interno di ShiVa questa entità prende il nome di “Dynamic”.

Definizione 4.1: **Dynamics.** Attributo (o “Controllore”) che risulta possibile associare ai vari oggetti in gioco, il cui compito è attribuire al modello determinate caratteristiche fisiche quali: peso, frizione, tangibilità, se sottoposto o meno alla gravità etc.; questo controllore può assumere forme geometriche molto semplici quali sfere, cubi o capsule (un cilindro con due sfere come estremità), ma risulta possibile combinarne diverse al fine di ottenere una forma che meglio si avvicini a quella dell'oggetto interessato.

Per ridurne al minimo il peso computazionale si è cercato di realizzare un corpo il più semplice possibile, mantenendo comunque delle fattezze adatte al gioco. Dopo vari prototipi si è concluso che l'utilizzo di una sfera per simulare i piedi del giocatore e di una scatola rettangolare per simularne il resto del corpo sarebbe stato sufficientemente leggero, offrendo al contempo collisioni sufficientemente realistiche con l'ambiente di gioco.

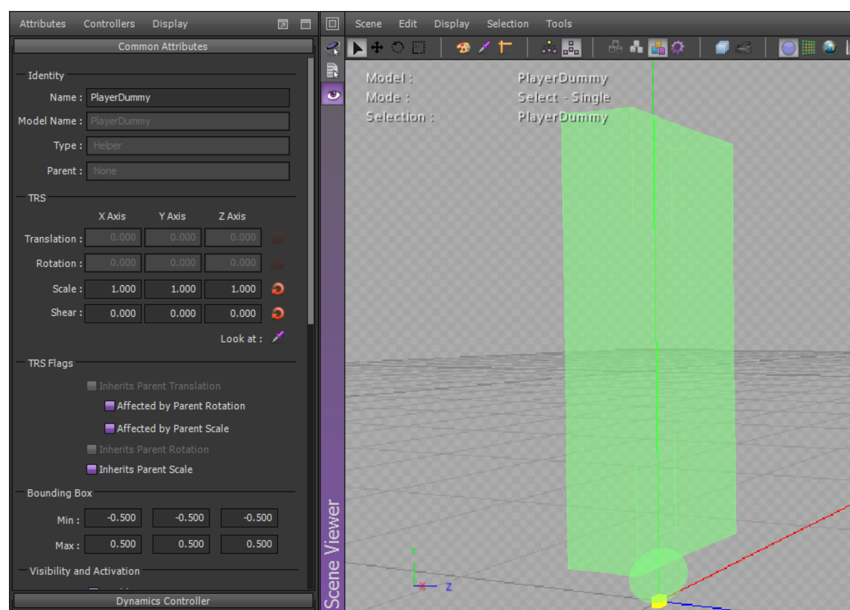

Illustrazione 4.7: Il corpo fisico del giocatore

4.5.3 *Gestione della telecamera di gioco*

Come detto in precedenza si è deciso di sviluppare il gioco in First Person View, cioè, permettendo al giocatore di "vedere" il mondo di gioco dalla prospettiva del suo personaggio. Questo si è fatto collegando al corpo fisico realizzato in precedenza una telecamera, essa serve a simulare gli occhi del giocatore, ed in base alla direzione in cui viene puntata, determina la direzione del corpo del personaggio.

La telecamera così realizzata seguirebbe il personaggio permettendo al giocatore di osservare l'ambiente di gioco liberamente, però, dato che il corpo fisico realizzato in precedenza non possiede gambe il suo movimento risulterebbe privo di quel naturale oscillamento presente nella camminata umana, trasmettendo così la sensazione che la telecamera di gioco si trovi su un "binario" e limitando di conseguenza l'immedesimazione.

Per rimediare a ciò si è scritta una funzione adibita ad applicare un piccolo offset verticale ed orizzontale alla telecamera durante la camminata.

```
local nDt = application.getLastFrameTime ( )
local nDegV, nDegH, nSwingX, nSwingY, nSwingZ = 0, 0, 0, 0, 0
-- If the player is applying "forward" trust, then calculate the camera swinging offset
if (this.nMoveY ( ) ~= 0) then
    this.nCameraSwingTime ( this.nCameraSwingTime ( ) +(nDt) )
    -- Vertical Swing
    nDegV = math.sin(this.nCameraSwingTime ( ) * 180 )
    -- Horizzontal Swing
    nDegH = math.cos(this.nCameraSwingTime ( ) * 90 )
    -- Apply the swinging in the right direction (depending on the camera direction)
    nSwingX, nSwingY, nSwingZ = math.vectorSetLength ( Zx, Zy, Zz, nDegH *
                            (this.nCameraSwingRatio ( )*this.nSpeedMultiplier ( )))
    nSwingY = nDegV * this.nCameraSwingRatio ( )*this.nSpeedMultiplier ( )
end
```

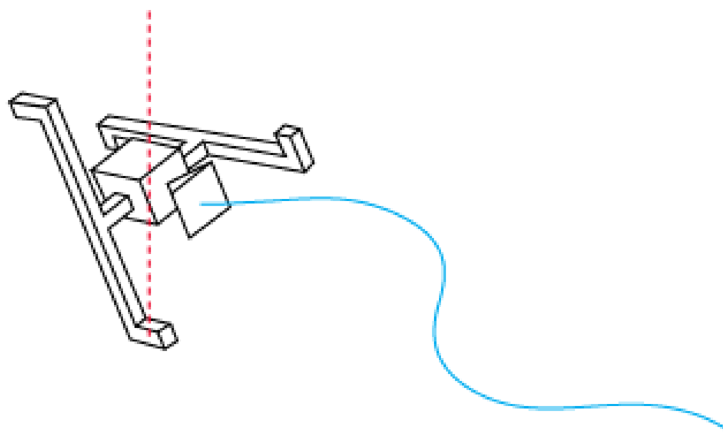

Illustrazione 4.8: Rappresentazione grafica del movimento della telecamera in gioco

4.5.4 Introduzione dell'interazione con gli oggetti

Si è scelto di inserire l'interazione con gli oggetti all'interno dell'AI del giocatore, facendo poi in modo che il mirino "reagisca" cambiando texture ogni volta che il giocatore inquadri un oggetto con cui è possibile interagire.

Per far questo si è deciso di racchiudere gli oggetti con un'AI pronta ad interagire con il giocatore all'interno di un sensore, rilevato tramite un raycast.

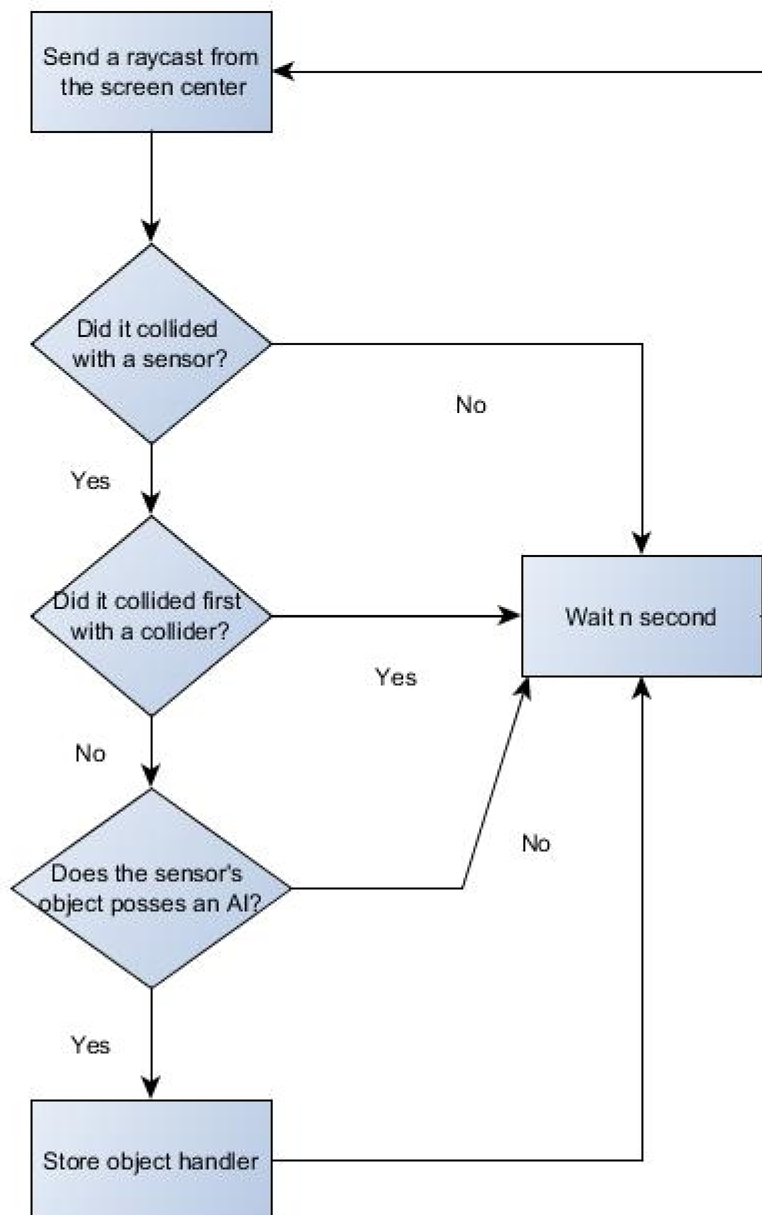

Illustrazione 4.9: La logica alla base del rilevamento degli oggetti con cui interagire

Rilevato il sensore di un oggetto con cui si può interagire ne viene salvato in memoria il puntatore, in modo che, se il giocatore deciderà di avviare l'interazione, sarà sufficiente inviare un evento tramite il puntatore salvato, informando l'oggetto del fatto che un'interazione sta iniziando.

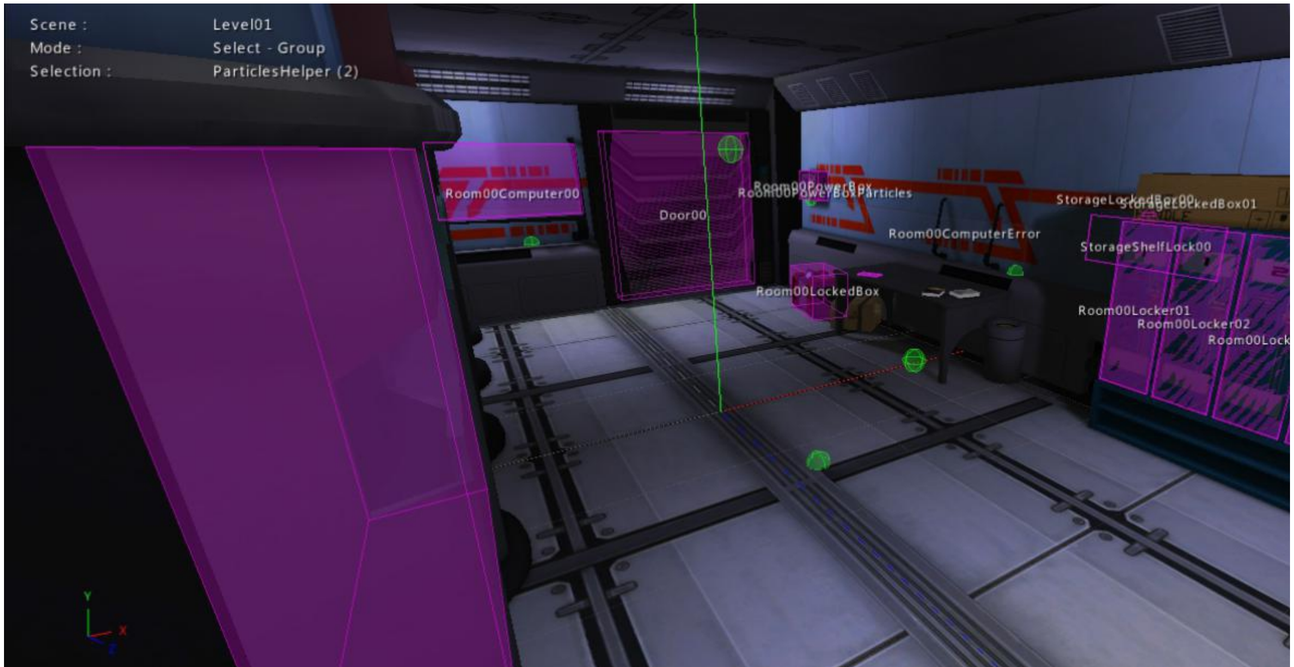

Illustrazione 4.10: Screenshot dell'ambiente di gioco con i sensori attorno agli oggetti con cui è possibile interagire

Illustrazione 4.11: I due stati del mirino

4.5.5 Gestione delle variabili dei componenti in gioco

Ogni elemento in gioco è caratterizzato da diverse variabili, che si è deciso di salvare all'interno di vari file xml in modo da permetterne la modifica rapidamente, senza la necessità di accedere al codice (i dati di ogni modello sono identificati dalla tag di quest'ultimo).

Oltre che per le variabili, questo è stato fatto anche per il testo delle interfacce, aggiungendo così la possibilità di tradurre il gioco in più lingue.

Attualmente sono stati realizzati quattro file xml:

1. **Cassini_Cfg:** In questo file vengono salvate tutte le impostazioni del gioco, come ad esempio la risoluzione di rendering, l'utilizzo del bloom ed il volume della musica.
2. **Cassini_Eng\Ita:** Questo file xml contiene tutto il testo richiesto in gioco.
3. **Cassini_Var:** Contiene tutte le variabili dei modelli in gioco, al suo interno sono salvate informazioni come ad esempio gli oggetti contenuti in una scatola o quale segnale richiederà una porta per tornare a funzionare (ad esempio il segnale che la scatola elettrica relativa è stata riparata).
4. **Cassini_Games:** Questo ultimo file xml contiene tutte le informazioni necessarie ai vari sottogiochi per funzionare, al suo interno troviamo informazioni come ad esempio la posizione dei nemici in un determinato sottogioco, il codice necessario per aprire una scatola a codice etc.

4.5.6 Definizione del comportamento degli oggetti

Come detto in precedenza, per definire il comportamento della maggior parte degli oggetti con cui il giocatore può interagire, si è deciso di realizzare una classe standard, la quale, all'avvio di una partita, legge all'interno di un file XML la lista degli stati che la caratterizza ed in che modo transitare fra di essi.

Questa classe, chiamata *Interactive_Basic*, è caratterizzata da un handler *onInteraction*, per ricevere l'interazione con il giocatore, ed un handler *onReceiveSignal* per ricevere tutte le tipologie di segnali che si sono configurati, quali ad esempio:

- **NextState:** Per informare la classe che è giunto il momento di passare allo stato successivo
- **Signal:** Per mandare un segnale ad un altro oggetto

- **ReleaseInterface:** Per chiudere l'interfaccia di interazione con il giocatore
- **SetChildVisible:** Per cambiare la visibilità di un suo sotto-modello (ad esempio per simulare la riparazione di un componente)

Alcuni degli stati introdotti per definire il comportamento degli oggetti sono stati:

- **StateBroken:** Per far transitare la classe allo stato successivo il giocatore deve interagire con essa consegnandole l'oggetto richiesto per ripararla
- **StateGame:** In questo stato, quando il giocatore interagisce con la classe viene aperta un'interfaccia contenente un sottogioco (definito all'interno del file XML), la classe transita allo stato successivo una volta terminato il sottogioco
- **StateLootable:** Nel momento in cui il giocatore interagisce con la classe essa inserisce uno o più oggetti nel suo inventario per poi transitare autonomamente allo stato successivo

L'interfaccia 2D dell'oggetto è collegata al suo stato, permettendo così di avere diverse interfacce per i suoi diversi stati.

Esempio

Durante una delle prime fasi di gioco, il giocatore si trova a dover riparare un generatore elettrico per poter procedere nell'avventura, il cui comportamento è definito nel seguente modo:

```
<GeneratorsElectricity>
	<States>
		<StateBroken>
			<ItemToFix>InvCablesPipe</ItemToFix>
			<OnReceiveSignal>
				<Signal>Repaired</Signal>
				<Actions>
					<Action01>SetChildVisible</Action01>
					<Action02>SetChildVisible</Action02>
					<Action03>NextState</Action03>
				</Actions>
				<ActionsData>
					<Action01>2-False</Action01>
					<Action02>1-True</Action02>
					<Action03>StateGame01</Action03>
				</ActionsData>
			</OnReceiveSignal>
		</StateBroken>
		<StateGame01>
			<HUD><Type>StateGameLogs</Type></HUD>
			<Games><Game01>GameRerouter</Game01></Games>
			<OnReceiveSignal>
				<Signal>GameWon</Signal>
				<Actions>
					<Action01>Signal</Action01>
					<Action02>ReleaseInterface</Action02>
```

```
                                        <Action03>NextState</Action03>
                                </Actions>
                                <ActionsData>
                        <Action01>BridgeCaptainComputer-Computer00-ElectricityRestored</Action01>
                                        <Action03>StateWorking</Action03>
                                </ActionsData>
                        </OnReceiveSignal>
                </StateGame01>
                <StateWorking>
                        <HUD><Type>StateGameLogs</Type></HUD>
                        <Games><Game01>GameRerouter</Game01></Games>
                </StateWorking>
        </States>
</GeneratorsElectricity>
```

Inizialmente l'oggetto si trova nello stato *StateBroken*, nel momento in cui il giocatore interagisce con esso, procede nel controllare se nel suo inventario è presente l'oggetto necessario alla sua riparazione (definito da *ItemToFix*), e nel caso, lo rimuove, a questo punto si auto invia un messaggio con il segnale *Repaired*, procedendo poi eseguendo le azioni legatevi, in questo caso cambiando la visibilità di due sotto modelli, per poi transitare allo stato successivo StateGame.

In questo nuovo stato, quando interagito dal giocatore, cede il controllo all'AIModel *HUDMenu*, che procede apprendono un interfaccia grafica 2D definita da <HUD><Type>, attraverso questa interfaccia il giocatore può poi avviare il sottogioco definito da <Games> <Game01>, sottogioco che, una volta ultimato, invia il segnale *GameWon* all'oggetto, avviando così l'esecuzione delle successive azioni, in questo caso inviando un messaggio all'oggetto avente tag BridgeCaptainComputer, rilasciando l'interfaccia 2D e procedendo allo stato successivo, che in questa occasione, è quello finale.

4.5.7 Estensione delle funzionalità 2D del motore

Anche se ShiVa supporta nativamente oggetti 2D, sotto forma di componenti di una form, il numero di funzionalità presenti risulta estremamente limitato, poiché pensato appositamente per la creazione di menu statici e non di componenti animati, perciò, per poter sviluppare il tipo di gameplay scelto per Adrift, si è rivelato necessario scrivere una classe per estenderne le funzionalità.

La classe in questione, chiamata Comp2dAI deve essere utilizzata per la creazione di qualunque componente 2D che si ha intenzione di utilizzare in gioco, una volta creato, esso viene inserito in una hashtable di hashtable contenenti tutte le informazioni necessarie alla gestione del componente e delle azioni associatevi, hashtable multidimensionale che viene poi ciclata ad ogni iterazione eseguendo ogni azione contenutavi.

Illustrazione 4.12: Activity diagram della gestione delle activity dei componenti 2D

Questa funzione è stata utilizzata, ad esempio, per animare gli elementi del sottogioco *Packet Routing*.

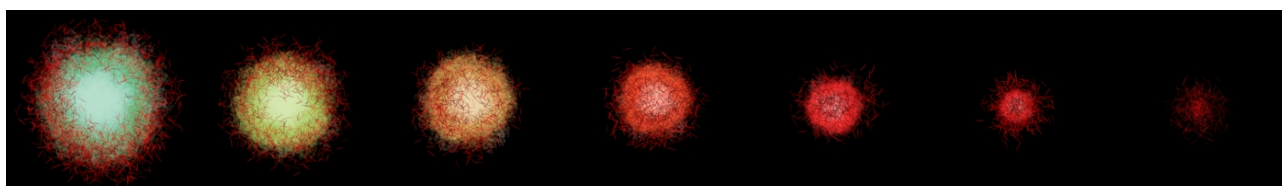

Illustrazione 4.13: L'animazione del pacchetto guidato dal giocatore

Alcune delle funzioni più interessanti implementate fino ad ora sono:

- La possibilità di fare comparire a schermo dei messaggi per informare il giocatore che, per esempio, sono stati raccolti o persi degli oggetti, questi messaggi scompariranno automaticamente dopo 3 secondi (grazie ad un sistema di timers che si è implementato).
- Impostare un effetto di fade in\out ad un componente.
- Inserire all'interno di una label uno sliding text laterale.
- Inserire un oggetto all'interno di un motore che controllerà le collisioni fra i componenti 2D dell'interfaccia. I componenti sono suddivisi in passivi e attivi, i primi rappresentano cose non in movimento, come pareti o ostacoli, e su cui non viene effettuato attivamente il controllo delle collisioni, i secondi invece sono quegli oggetti che si muoveranno all'interno del gioco e su cui sarà necessario effettuare il controllo delle collisioni, sia con gli oggetti passivi che con gli altri oggetti attivi.
- Impostare ad un componente una velocità di movimento ed una destinazione, ad ogni iterazione esso ricalcolerà la sua posizione rispetto alla destinazione ed applicherà la velocità nella direzione migliore per raggiungerla.

4.5.8 *Creazione delle interfacce 2D*

Per permettere una realizzazione rapida di interfacce grafiche che risultino inoltre semplici da modificare, è stato necessario introdurre un sistema automatico di generazione delle pagine, dato che, anche se ShiVa include al suo interno un editor grafico per la creazione di form, questo sistema risulta pensato per lo sviluppo di poche pagine sconnesse fra loro (ad esempio una schermata di caricamento, oppure la creazione dell'interfaccia a schermo del giocatore), e dunque, del tutto inadeguato allo sviluppo di menu complessi e facilmente editabili, come invece richiesto in Adrift.

Per risolvere questo problema, si è definita una serie di funzioni per la creazione di interfacce tramite template, richiamabili da qualunque entità in gioco attraverso handler in Comp2dAI.

Ogni modello in gioco a cui è collegata una classe per la creazione di un'interfaccia, trova all'interno del file xml Cassini_Var tutte le informazioni necessarie a richiedere e definire nel dettaglio la pagina a Comp2dAI.

Esempio

Un computer in gioco, definito dalla tag "CryoComputer", ha collegata un'istanza della classe HUDMenu, durante l'interazione con l'utente essa acquisisce le informazioni necessarie alla creazione della sua interfaccia da un file xml.

```
<CryoComputer>
...
        <HUD><Type>StateGameLogs</Type></HUD>
        <Games><Game01>GameMemoryRestore</Game01></Games>
        <OnReceiveSignal>
                <Signal>GameWon</Signal>
                <Actions>
                        <Action01>Signal</Action01>
                        <Action02>ReleaseInterface</Action02>
                        <Action03>NextState</Action03>
                </Actions>
                <ActionsData>
                        <Action01>CryoDoorComputer-DoorComputer-MemoryFix</Action01>
                        <Action03>StateWorking</Action03>
                </ActionsData>
        </OnReceiveSignal>
</CryoComputer>
```

In questo caso il template e di tipo *StateGameLogs*, questo significa che l'interfaccia 2D che descrive è composta dai seguenti link:

- **State**: un link che, quando cliccato, apre una nuova pagina contenente una descrizione dello stato attuale dell'oggetto.
- **Game**: il link che permette l'apertura di un sotto gioco (in questo caso di tipo *GameMemoryRestore*).

- **Logs**: un link contenente la cronologia degli eventi accaduti all'oggetto.

Illustrazione 4.14: L'interfaccia 2D in funzione

4.5.9 Gestione dell'audio

Per quanto riguarda l'audio in gioco si è deciso di affidarne la gestione ad una singola classe denominata SoundAI; questo per semplificare la progettazione delle singole classi collegate ai modelli.

Ciò che si è fatto è stato creare una risorsa contenente tutti i suoni necessari in gioco, gestendone l'utilizzo attraverso una serie di handler, qualunque elemento che necessiti di effettuare un qualche tipo di suono (ad esempio il rumore di una scatola che si apre), invia un evento alla SoundAI indicandole il suono da eseguire, sarà poi quest'ultima ad occuparsi di gestirne l'esecuzione in base alle impostazioni audio scelte dal giocatore.

4.6 Creazione di un sottogioco

Di seguito si descrive nel dettaglio come è stato sviluppato il sottogioco *Hidden Objects*, uno dei tanti in Adrift.

Stesura del documento iniziale

Per prima cosa si è redatto un documento in cui si è descritto il gameplay, le regole e lo stile grafico.

Hidden Objects

In questo sottogioco l'utente deve individuare, in un'immagine 2D contenente tantissimi oggetti, gli elementi elencati in una lista, questi elementi interagiscono o si possono combinare fra loro sbloccandone dei nuovi al fine di raggiungere l'obiettivo del sottogioco, rappresentato dall'ottenimento di un oggetto necessario in qualche altra parte dell'avventura.

Il modo in cui gli elementi si relazionano fra loro è descritto all'interno di un file xml, attraverso una serie di comandi che vengono poi interpretati durante la creazione della schermata di gioco.

La grafica è composta da un'immagine di sfondo 2D e da un insieme di componenti aventi come texture le immagini di vari oggetti.
Alcuni di questi componenti sono interattivi e, ricevuto l'input del giocatore, controllano il loro stato (definito da un file xml) e agiscono di conseguenza, alcune delle azioni che possono eseguire sono:

- Fornire al giocatore un elemento
- Richiedere un elemento già raccolto dal giocatore per attivarsi
- Una volta interagito con essi inviare un segnale a un altro componente in gioco per attivarlo
- Fornire al giocatore l'elemento richiesto per vincere la partita

Il gioco è strutturato in modo che il giocatore possa iniziare interagendo con diversi oggetti, ma che proseguendo, il percorso percorribile divenga uno soltanto, portando alla fine all'interazione con l'ultimo oggetto che fornirà al giocatore il premio.

Sketch dell'interfaccia

A questo punto si sono realizzati diversi sketch cercando di definire la parte grafica.

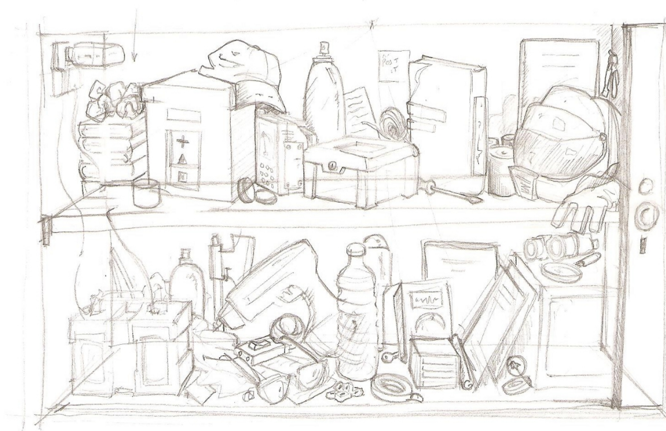

Illustrazione 4.15: Lo sketch definitivo dell'interfaccia di gioco

Definizione della classe

Ogni sottogioco è definito da una classe contenente tutte le funzioni necessarie al suo svolgimento, suddivisa in stati che rappresentano le varie fasi di una partita (inizio partita, vittoria, sconfitta etc.).

Questa classe si trova inoltre a dover comunicare con l'oggetto a cui è connessa, perciò include anche tutti gli handler necessari per gestire la comunicazione con il giocatore, l'oggetto che la contiene ed il menu da cui è stata lanciata.

Una volta avviata una partita il sottogioco acquisisce le caratteristiche del match dal file xml Cassini_Games, per poi salvarle in una tabella multidimensionale.

Attraverso questo file vengono definite le caratteristiche della partita, quali:

- L'immagine di background
- La lista degli oggetti che il giocatore deve trovare
- Tutte le informazioni necessarie a realizzare i bottoni con cui il giocatore può interagire
- Le informazioni per realizzare le label degli oggetti che fanno da sfondo e con cui il giocatore non poù interagire

```
<Background>Game_HiddenObjects_Background00</Background>
<Items><!--Items list -->
	<n0 Qt="2">Game_HiddenObjects_I000</n0>
	<n1>Game_HiddenObjects_I001</n1>
	<n1>Game_HiddenObjects_I002</n1>
	...
</Items>
<Buttons><!-- In Game items variables -->
	<Key>
		<States>1</States>
		<BtnCoords>25.5-24</BtnCoords>
		<BtnSize>3-6</BtnSize>
		<LabelCoords>27.5-30</LabelCoords>
		<LabelSize>8-12</LabelSize>
		<ZOrder>190</ZOrder>
		<LabelImage>Game_HiddenObjects_Items_Key</LabelImage>
		<LabelImageSize>0.74-1</LabelImageSize>
		<LabelImageCoords>
				<n0>0-0</n0>
		</LabelImageCoords>
		<Active>true</Active>
		<Actions>
			<n0>
				<n0>Store-Game_HiddenObjects_I001</n0>
				<n1>Disable</n1>
				<n2>Invisible</n2>
			</n0>
		</Actions>
	</Key>
```

```
            <BatterySlot><!--Torch battery slot 1-->
                    <States>3</States>
                    <BtnCoords>14-22.5</BtnCoords>
                    <BtnSize>21.3-21.3</BtnSize>
                    <LabelCoords>14-22.5</LabelCoords>
                    <LabelSize>20.5-20.5</LabelSize>
                    <ZOrder>180</ZOrder>
                    <LabelImage>Game_HiddenObjects_Items_BatterySlot</LabelImage>
                    <LabelImageSize>0.5-0.5</LabelImageSize>
                    <LabelImageCoords>
                            <n0>0-0.5</n0>
                             <n1>0.5-0.5</n1>
                             <n2>0-0</n2>
                    </LabelImageCoords>
                    <Active>true</Active>
                    <Actions>
                            <n0>
                                    <n0>Signal-TorchBulb-1-BatterySlot</n0>
                                    <n1>States</n1>
                            </n0>
                            <n1>
                                    <n0>Signal-TorchBulb-1-BatterySlot</n0>
                                    <n1>States</n1>
                                    <n2>Disable</n2>
                            </n1>
                    </Actions>
                    <NeedItems>
                             <n0>Game_HiddenObjects_I000</n0>
                             <n1>Game_HiddenObjects_I000</n1>
                    </NeedItems>     ...
    </Buttons>
    <Labels>
            <Binder>
                    <LabelCoords>63-73</LabelCoords>
                    <LabelSize>30</LabelSize>
                    <LabelImage>Game_HiddenObjects_BgItem_Binder</LabelImage>
                    <ZOrder>180</ZOrder>
            </Binder>...
     </Labels>
```

Come si può intuire dal file XML, il comportamento dei bottoni si basa sullo stesso approccio utilizzato per definire il comportamento degli oggetti interattivi in gioco, introducendo una serie di stati fra cui possono transitare, ognuno dei quali definito da una serie di prerequisiti per essere raggiungibile, ed una lista di azioni che attiva quando raggiunto.

Esempio

Durante la partita, il giocatore, per illuminare una parte di mappa fino a quel momento oscurata, si trova costretto a ricostruire il circuito di una torcia, sostituendone i vari componenti danneggiati.

Illustrazione 4.16: Due stadi del sottogioco Hidden Objects

4.7 Modellazione ed animazione degli oggetti

Una volta realizzata la struttura base del gioco è stato il momento di occuparsi della realizzazione dei modelli necessari, di seguito si descrive come essi sono stati creati, texturizzati ed animati.

Il workflow utilizzato durante la modellazione di qualunque oggetto non triviale che si è deciso di utilizzare è composto dalle seguenti fasi.

Stesura della descrizione iniziale e sketch

Per prima cosa si è redatto un piccolo documento, il cui compito è descrivere l'oggetto, le sue dimensioni, lo stato e le eventuali animazioni, dopodiché, basandosi su di esso si sono creati diversi sketch, fra questi si è scelto quello considerato più adatto procedendo poi a raffinarlo fino ad ottenere la forma finale dell'oggetto.

Locked Box

Scatola di metallo con codice che il giocatore dovrà aprire per ottenere qualche tipo di oggetto.

Sarà caratterizzata da una forma rettangolare ed un display in cui inserire un codice che una volta digitato la farà aprire e disattivare.

Deve trasmettere un senso di solidità, è una scatola che viene utilizzata sul lavoro, spesso in condizioni difficili, quindi deve essere robusta e deve esprimere la chiara idea che senza codice non sia possibile riuscire ad aprirla, allo stesso tempo deve avere uno stile moderno per rappresentare il fatto che ci troviamo nel futuro.

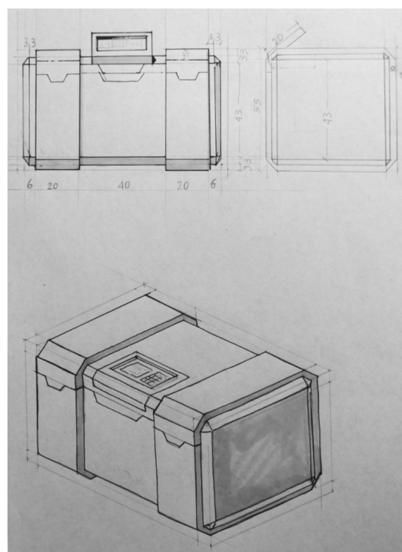

Illustrazione 4.17: Lo sketch della locked box basato sulla sua descrizione

Modellazione[9]

Durante la fase di modellazione si è realizzato l'oggetto, basandosi sia sul documento che sullo sketch prodotto e cercando di utilizzare il numero maggiore possibile di modifier*,* in modo da ridurre sensibilmente il tempo di sviluppo e rendere il modello più facilmente modificabile in futuro.

Definizione 4.2: **Modifier**. Script integrati in Blender che permettono di velocizzare la maggior parte delle operazioni di modellazione più comuni. Ad esempio, il *Mirror modifier* permette di modellare solo un lato dell'oggetto, la parte rimanente risulterà realizzata automaticamente dal modifier, come se fosse riflessa sugli assi da uno specchio. Un altro modifier estremamente utile è il *Decimate*, esso permette di ridurre il numero di facce totali di un modello in maniera automatica, lasciando così all'utente la libertà di realizzare un oggetto senza preoccuparsi del numero totale di triangoli e concentrandosi quindi unicamente sul suo aspetto. Questo modifier si è rivelato estremamente utile dato che il numero di poligoni totali renderizzabili a schermo su PC risulta estremamente più alto rispetto che su mobile, perciò, attraverso l'uso di questo script è stato possibile realizzare tutti i modelli high poly esportandone poi una versione low poly per i dispositivi mobile in automatico.

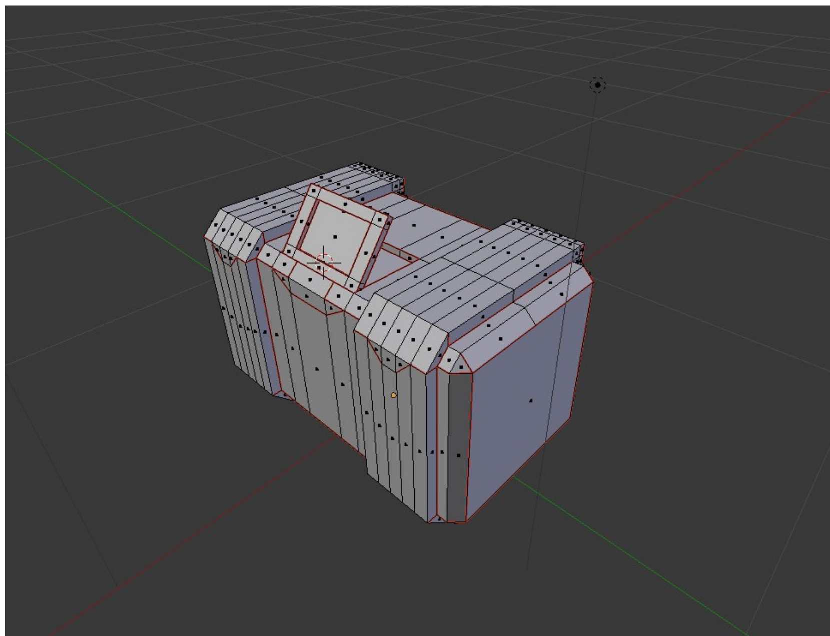

Illustrazione 4.18: La locked box modellata

Durante la modellazione degli ambienti di gioco più complessi oltre che allo sketch e la breve descrizione scritta si è anche fatta realizzare a Filippo una piantina in CAD, poi importata in Blender ed utilizzata come base per lo sviluppo.

Texturizzazione

La fase di texturizzazione consiste nel collegare ad ogni faccia del modello una parte di una texture, al fine di aggiungere dettagli visivi all'oggetto, questo procedimento prende il nome di Texture Mapping.

Una volta finito di modellare l'oggetto si è proceduto scollegando fra loro alcune delle sue facce in modo da riuscire poi a "stendere" il modello su di una superficie bidimensionale (la nostra texture).

Blender permette di automatizzare questo procedimento ma raramente con risultati soddisfacenti dato che, per rappresentare determinati dettagli in un modello spesso è necessario offrire una porzione maggiore di texture a determinate facce rispetto che ad altre, non considerando questo fatto, Blender, tende invece a cercare di offrire alle facce un'area correlata alla loro dimensione, rendendo questa funzione utilizzabile solamente in quei modelli in cui nessuna area necessita di maggior dettaglio rispetto alle altre.

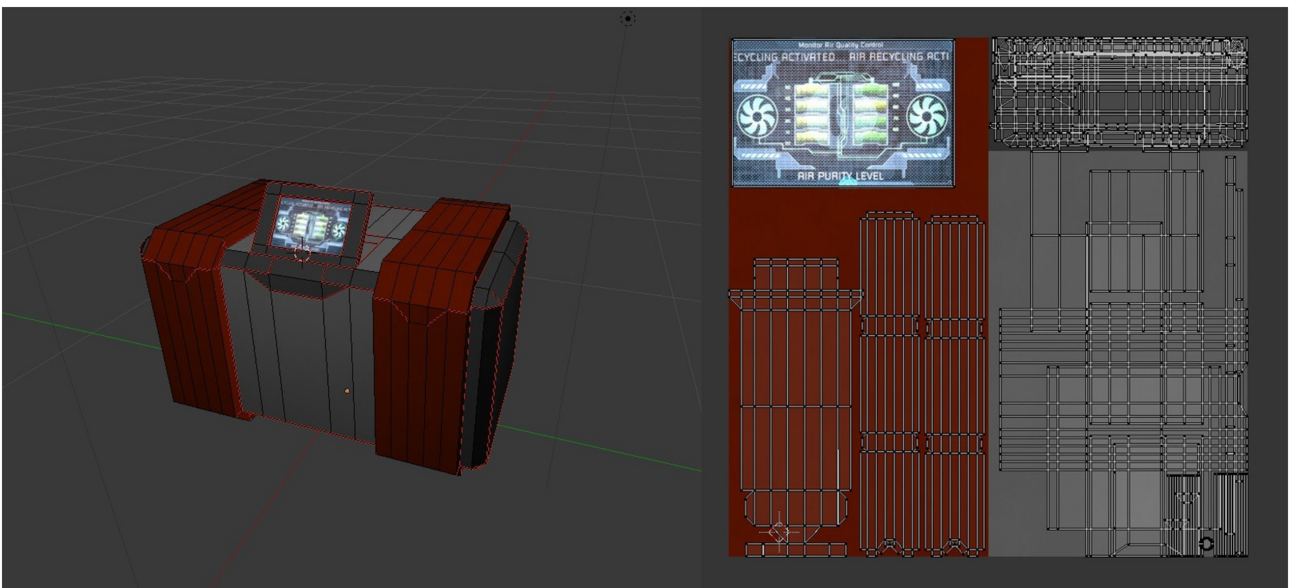

Illustrazione 4.19: Il modello di Locked Box texturizzato

Animazione

Nel caso l'oggetto fosse animato ne sono state definite le animazioni attraverso uno dei due metodi descritti di seguito, scelto in base alla tipologia di modello e di animazione richiesta.

Animazione di un intero modello

Con questo approccio è possibile traslare, ruotare e scalare nel piano di gioco un intero modello, questo tipo di animazione risulta molto semplice nel caso in cui l'oggetto da animare abbia un solo

elemento in movimento (come ad esempio una porta), ma si rivela estremamente complesso ed inefficiente con oggetti in cui più parti sono animate contemporaneamente.

Inoltre la necessità di considerare ogni parte animata di un oggetto come un sotto-modello a se, porta ad un aumento considerevole del numero di draw call totali, andando così a danneggiare le prestazioni, soprattutto in ottica mobile.

Questo tipo di animazioni risulta realizzabile all'interno di ShiVa tramite l'editor dedicato, ciononostante si è preferito utilizzare Blender, rivelatosi estremamente più versatile e semplice da usare.

Illustrazione 4.20: L'animazione in funzione

Animazioni tramite armatura

Il metodo migliore per animare modelli complessi consiste nel realizzare lo scheletro del modello (rigging), per poi collegare i vari vertex group alle ossa dello scheletro (skinning), così facendo sarà possibile muovere il modello semplicemente animando lo scheletro.

Definizione 4.3*:* **Vertex Group**[10]. I vertex group sono un sottoinsieme di vertici di un oggetto, utilizzati per identificarne determinate sezioni, ad esempio, nel caso di un modello di un essere umano, alcuni vertex group potrebbero essere l'insieme dei vertex che ne compongono la testa o il torso.

Questa tecnica anche se inizialmente molto più complessa di quella precedentemente descritta risulta drasticamente più potente e una volta padroneggiata, grazie all'utilizzo dei vertex goup, elimina la necessità di avere diversi sotto modelli mantenendo così il numero di draw call più basso.

Ad esempio, nel caso della locked box precedentemente modellata, si è deciso di animare due parti di essa contemporaneamente, il coperchio ed il suo schermo, per fare questo si sono creati tre vertex group separati, collegando il primo alla parte del modello non animata (il fondo) ed i rimanenti alle sue parti animate.

Fatto questo si è realizzato lo scheletro del modello, anch'esso composto da tre ossa, collegandole poi ai rispettivi vertex group, infine, si è definita la loro posizione in determinati keyframe

(posizione, rotazione e scala), lasciando poi a Blender il compito di interpolarli e definire una transizione fluida fra di essi.

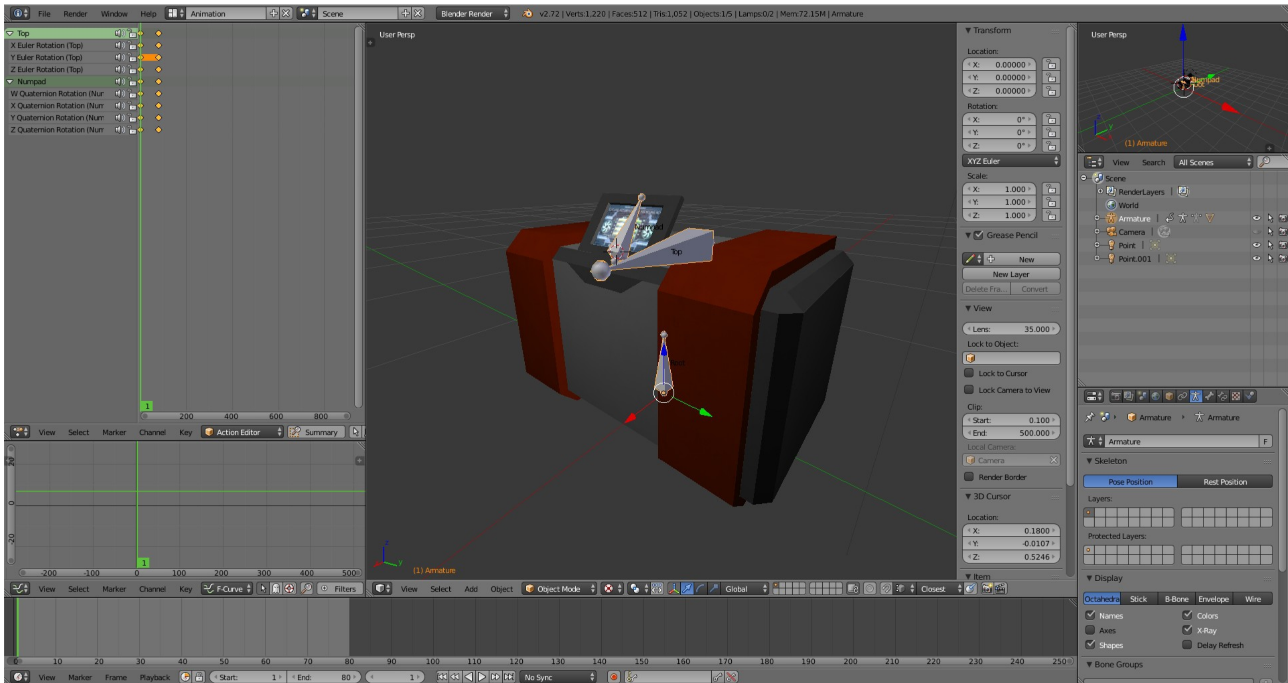

Illustrazione 4.21: Lo scheletro della locked box una volta definiti i suoi keyframe

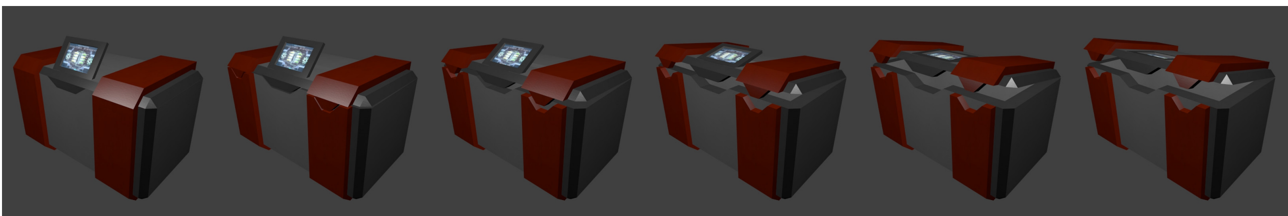

Illustrazione 4.22: L'animazione in funzione

4.8 Ottimizzazione

Durante l'intero sviluppo del gioco si è cercato di implementare tutte le risorse necessarie in modo tale da non scendere mai sotto una determinata soglia di frame per secondo:

- Maggiore di 20-25 per tutti dispositivi mobile aventi prestazioni medie dal 2011 in avanti

  Questo perché, la dimensione ridotta degli schermi mobile unita alle generali condizioni in cui vengono utilizzati (sul treno, in ufficio, in un parco) riduce l'importanza di avere quella perfetta fluidità offerta da un numero di frame maggiore o uguale a 60 al secondo.

- Maggiore di 60 per tutti i pc di basso livello prodotti dal 2008 ad oggi

  Rispetto ai dispositivi mobile la maggior parte dei personal computer possiede almeno uno schermo full HD[11] con una dimensione maggiore o uguale a 19 pollici, inoltre questo genere di piattaforma viene principalmente usato in ambienti in cui l'utente può più facilmente concentrarsi sulle immagini (in ufficio o a casa), per questa serie di motivi risulta necessario che le operazioni a schermo avvengano con un frame rate sufficiente a renderle perfettamente fluide.

4.8.1 Ottimizzazione delle facce dei modelli

Per ottimizzare i vari modelli in gioco si sono utilizzati due approcci diversi, nel primo caso si è semplicemente cercato di definire tutti i dettagli necessari tramite un numero di facce il più ridotto possibile[12], mentre nel secondo, ciò che si è fatto è stato procedere eliminando tutte le facce che il giocatore non sarebbe stato in grado di vedere (a causa della posizione del giocatore o del modello).

Questo significa che se nel primo caso ci si è concentrati sul realizzare l'intero oggetto come si procederebbe nella realtà, semplicemente limitandone i dettagli, nel secondo lo si è invece trattato come un elemento astratto ignorandone la struttura totale concentrandosi unicamente su ciò che sarebbe servito.

Un esempio molto chiaro di questo è il ponte di comando, dato che il giocatore non ha mai modo di vedere la nave da fuori, tutto ciò che si è modellato sono state le pareti interne ignorando completamente lo scafo.

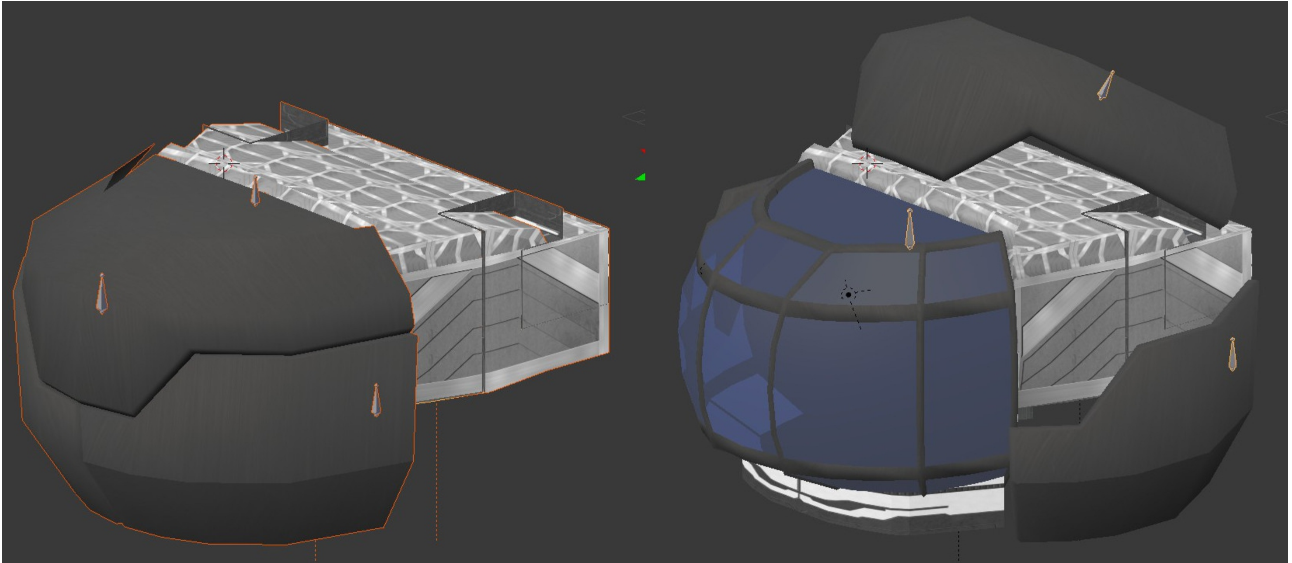

Illustrazione 4.23: L'esterno del ponte di comando della nave

4.8.2 Limitazione delle classi attive

Si è cercato inoltre di limitare il numero di classi contemporaneamente attive, facendo in modo che la maggior parte delle entità non si attivi se non in prossimità del giocatore, purtroppo la versione utilizzata di ShiVa per lo sviluppo di questa tesi (basic) non ha permesso l'utilizzo di funzioni avanzate (quale l'analisi del peso dei vari elementi ad ogni frame) che avrebbero sicuramente semplificato questa fase, si prevede comunque di acquistare la versione avanzata prima della release finale, al fine di poter effettuare un'ottimizzazione più accurata.

4.8.3 Gestione dei modelli extra in gioco

Per rendere il gioco più attraente agli utenti in possesso di un dispositivo più potente, si è deciso di introdurre una serie di oggetti extra, cioè modelli non necessari al proseguimento dell'avventura ma inseriti per rendere l'ambiente di gioco più interessante ed immersivo, affidandone la gestione ad una classe apposita chiamata *ExtraModelsLoader.*

Attraverso il menù delle opzioni video il giocatore può scegliere fra tre livelli di qualità dell'ambiente (dettagli: bassi, normali e alti), questa scelta definisce quali modelli extra saranno o meno resi visibili in gioco.

Questo approccio, che vede i modelli caricati indipendentemente dalla qualità grafica impostata, ha permesso di posizionare tutti gli oggetti extra attraverso l'editor grafico di ShiVa (rendendone così la manipolazione molto più immediata) nonché il calcolo delle loro lightmap (al momento Shiva non permette l'applicazione di lightmap via script ai modelli caricati a runtime), con, come unico

aspetto negativo, il fatto di avere una porzione della memoria occupata da questi modelli, anche se non impostati come visibili, ma, data la dimensione totale in memoria occupata da questi modelli, non si è ritenuto importante.

Questo approccio è stato utilizzato, ad esempio, per il caricamento dei modelli dei membri dell'equipaggio all'interno delle camere criogeniche, resi visibili solo a dettagli alti. Per evitare però che, ad inferiori livelli di dettaglio, le camere criogeniche risultassero vuote, andando così contro ciò che la storia racconta, si è introdotta in *ExtraModelsLoader* la possibilità di variare gli attributi dei materiali dei modelli, permettendogli così di rendere il vetro delle camere criogeniche non trasparenti quando necessario.

Illustrazione 4.24: Le camere criogeniche a confronto (dettagli normali a sinistra ed alti a destra)

```
-- Go through all the quality tables
    for nQ = 0, 2 do
        local bVisible = true
        -- If a table is highter than the current quality level, set the visibility to false
        if (nQ > nNewQuality) then
            bVisible =  false  end
        local tTargetQuality = table.getAt ( this.tExtraModels ( ), nQ )
        local nTargetQualityModels = table.getSize ( tTargetQuality ) - 1
        -- Update the visibility of all the models tagget inside the table
        for nModel = 0, nTargetQualityModels do
            local sModelTag = table.getAt ( tTargetQuality, nModel )
            local hModel = application.getCurrentUserSceneTaggedObject ( sModelTag )
            if ( hModel ) then
                object.setVisible ( hModel, bVisible )
            end
        end
    end
```

4.8.4 Definizione di oggetti come occluder ed occludable

Tramite questa tecnica è stato possibile definire dei modelli come *occludable*, *cioè* che non vengono renderizzati se completamente coperti al campo visivo del giocatore da altri modelli, impostati come *occluder*. Non si è potuta applicare questa opzione a tutti gli elementi in gioco perché il

motore di gioco è costretto ad eseguire svariati calcoli per verificare se ogni oggetto è occluso o meno, rendendola vantaggiosa unicamente su quegli oggetti con un'alta probabilità di venire coperti da altri.

Ad esempio, applicare quest'opzione ad un modello quale un tavolo potrebbe migliorare le prestazioni del gioco, essendo facilmente oscurabile da qualche altro oggetto non appena il giocatore lasci la stanza in cui è contenuto.

Al contrario impostare il pavimento del corridoio della nave come *occludable* porterebbe ad un decadimento delle prestazioni, dato che, la sua grossa dimensione richiederebbe calcoli molto complessi per determinarne l'occlusione o meno, ed inoltre il giocatore spende molto tempo in questi corridoi rendendo l'opzione quasi mai utilizzata.

Una volta fatto questo è stato possibile (previa la precedente definizione delle dimensioni) suddividere l'intero ambiente di gioco in sotto zone, per poi, attraverso uno dei moduli integrati in ShiVa, calcolarne le occlusioni statiche. Questo procedimento crea un k-d tree in cui ogni zona è rappresentata da una foglia, al cui interno risulta salvata una lista delle foglie dell'albero visibili da essa, durante l'esecuzione, il motore di gioco verifica in quale zona l'utente si trovi ed elimina dalla coda di rendering tutti gli oggetti non contenuti in zone visibili da quella attuale.

Si è inoltre impostata una distanza massima di visibilità a tutti gli oggetti (calcolata in base alla geometria del livello), escludendoli attivamente dal gioco non appena il giocatore si allontani troppo da essi.

4.8.5 Gestione del caricamento degli ambienti di gioco

Durante lo sviluppo di Adrift si è sempre mantenuto come obiettivo quello di sviluppare un gioco privo di schermate di caricamento per tutta la durata dell'avventura (questo per favorire l'immedesimazione del giocatore), ma, durante lo sviluppo del secondo livello della nave ci si è resi conto di come la quantità di memoria richiesta iniziasse a risultare piuttosto alta rispetto alla quantità media disponibile su dispositivi mobile.

Per far fronte a questo inconveniente si è deciso di modificare leggermente la struttura della nave, inizialmente si era pensato di collegare i tre livelli attraverso delle scalinate e di permettere al giocatore di spostarsi, sin dall'inizio dell'avventura, liberamente per tutti e tre i livelli, ciò che si è fatto invece è stato sostituire le scalinate con un'ascensore e renderlo parte del gameplay, ora infatti, l'utente, per potersi spostare di livello in livello, deve prima riparare l'ascensore e poi trovare delle

chiavi di accesso per sbloccare i vari piani della nave.

Attraverso l'uso dell'ascensore è stato poi possibile mascherare il cambio di livello facendo uso di un sistema di caricamento delle risorse asincrono, la nave è stata suddivisa in tre scene diverse, ognuna delle quali contenente un intero livello di essa ed una copia identica dell'ascensore, nel momento in cui il giocatore sale sull'ascensore e seleziona la destinazione, vede la porta chiudersi e sente un rumore ad indicare il movimento dell'ascensore verso il nuovo livello, in quel momento si avvia il caricamento asincrono della scena successiva e, appena terminato, sostituisce il nuovo livello al vecchio, il giocatore non si accorge di questo cambio dato che l'ambiente intorno a lui non muta, a questo punto le porte si aprono, dando l'illusione al giocatore di essersi fisicamente spostato attraverso la nave e di non aver dovuto caricare nulla.

4.8.6 Ottimizzazione in ottica multipiattaforma

Come detto in precedenza si è cercato di limitare la fase di ottimizzazione specifica per ogni piattaforma il più possibile, cercando di inserirne la maggior parte via codice.

Un esempio di questo può essere l'uso degli effetti particellari, i dispositivi mobile risultano in grado di supportare un numero di effetti molto più ridotto rispetto alla controparte PC, quindi, per evitare di dover modificare manualmente il numero e la qualità degli effetti particellari per ogni piattaforma, si è deciso di creare una classe apposita e di collegarne un'istanza ad ogni emettitore di particelle in gioco, al momento dell'avvio dell'applicazione, essa effettua un controllo sul sistema operativo utilizzato e modifica di conseguenza numero e qualità degli effetti ad essa collegati.

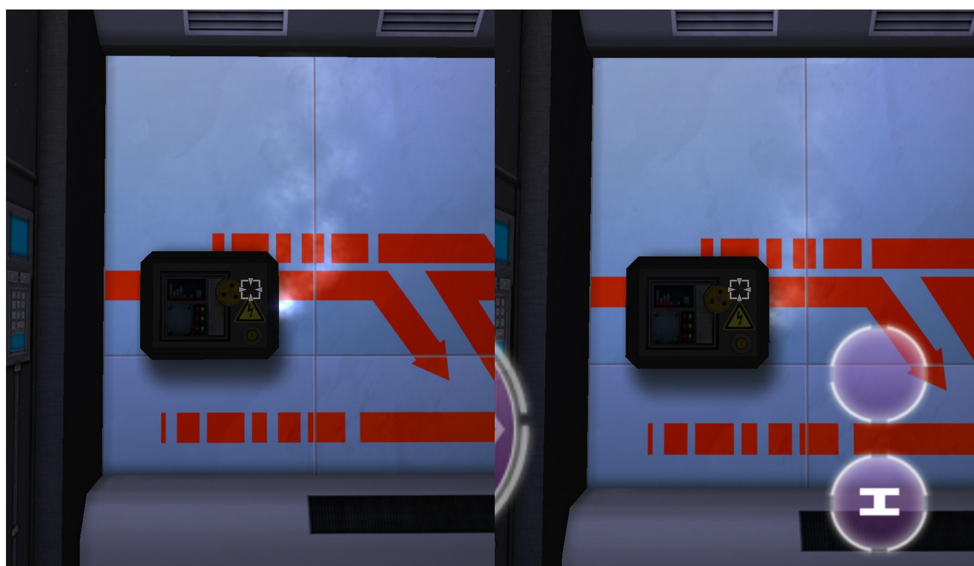

Illustrazione 4.25: Un effetto di fumo a confronto: a sinistra su PC e a destra su table Android

Un altro esempio di ottimizzazione automatizzata riguarda la dimensione delle texture utilizzate in gioco, come in precedenza descritto i dispositivi mobile riescono a supportare senza problemi texture fino all'incirca 1024×1024 pixel, su PC però le dimensioni standard oramai possono essere di 2048x2048 o addirittura 4096×4096, costringendoci quindi ad avere due set di tutte le texture in gioco a dimensioni differenti.

Per fortuna però questo si è potuto evitare, grazie ad una funzione contenuta all'interno dell'editor di ShiVa, che permette di realizzare profili legati alle piattaforme per cui si compila il gioco, al cui interno specificare le dimensioni massime delle singole texture e se voluto il livello di compressione da applicarvi, perciò è stato sufficiente caricare tutte le texture alla dimensione massima che si è ritenuta necessaria, lasciando poi all'editor il compito di ridimensionarle in base alla piattaforma per cui si sta esportando il gioco.

In altri casi invece si è scelto di affrontare il problema utilizzando un approccio compatibile con entrambe le piattaforme, ignorando funzioni che avrebbero velocizzato lo sviluppo su di una, ma richiesto la totale riscrittura per la seconda.

Un esempio di questo è la gestione delle musiche e dei suoni in gioco. Durante la realizzazione di uno dei prototipi si è scoperto che alcuni dei comandi offerti da ShiVa (come ad esempio il controllo della progressione di una traccia audio) non erano supportati appieno da alcuni dispositivi mobile, perciò, invece di utilizzare due approcci diversi si è deciso di scrivere un sistema più complesso ma in grado di girare su entrambe le piattaforme.

In questo preciso caso, quel che si è fatto è stato creare una tabella contenente la durata dei vari brani in gioco ed utilizzarla per lanciare, al momento dell'inizio di un nuovo pezzo, un timer con durata uguale alla canzone in riproduzione.

```
--------------------------------------------------------------------------------
--  Function......... : setMusicTimer
--  Author........... : Alex Grassi
--  Description...... : When starting a new song, create a timer that will call onMusicEnd exactly 
when the song end
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
function SoundAI.setMusicTimer ( nTimeTable )
--------------------------------------------------------------------------------

    local hTimer = scene.getTaggedObject ( application . getCurrentUserScene ( ), "Timer" )
    object.sendEventImmediate ( hTimer, "TimerAI", "onNewTimerUser", "musicTimer", "SoundAI",
                                "onMusicEnd", table.getAt ( this.tMusicTime ( ), nTimeTable ) )
--------------------------------------------------------------------------------
end
--------------------------------------------------------------------------------
```

Anche per quanto riguarda le luci è stato necessario predisporre il gioco ad entrambe le piattaforme, questo perché, se un ambiente di gioco con illuminazione statica viene considerato del tutto normale per mobile, su PC oramai l'utente medio esige sempre ombre ed illuminazione dinamica.

Per fare questo si sono configurate tutte le luci in gioco come dinamiche ma tenute disattivate e collegate ad una classe che, al momento dell'avvio di una partita, controlli la piattaforma presente e le attivi se necessario.

4.8.7 Modifiche al gameplay

In alcuni casi è stato necessario modificare il gameplay dei sottogiochi per adattarli meglio alle diverse piattaforme.

Ad esempio nel sottogioco "*Memory Restoration*" (in cui il giocatore deve identificare il simbolo corretto da una lista prima che scada il tempo), inizialmente i simboli con cui interagire erano stati posizionati senza testare il gioco su mobile e questo aveva portato ad un accentramento dei componenti (questo perché, su PC, solitamente, gli elementi con cui si può interagire tramite mouse vengono sempre posizionati al centro dello schermo), una volta rilasciato il prototipo per mobile al gruppo di tester interni è emerso immediatamente come questo posizionamento risultasse inefficiente nel caso di dispositivi touch, questo perché solitamente cellulari e tablet, durante una partita, vengono tenuti con entrambe le mani e vengono utilizzati i pollici per interagire con lo schermo, è chiaro ora come il centro dello schermo risulti essere il punto più difficile da raggiungere e quindi meno adatto a contenere elementi con cui interagire.

Per risolvere questo problema si sono spostati i vari elementi, con cui il giocatore deve interagire, dal centro dello schermo alla sua destra, in modo da essere facilmente raggiungibili dal pollice destro, dopodiché per mantenere il gioco fruibile su pc si è aggiunta la possibilità di selezionare i simboli anche tramite input da tastiera.

Illustrazione 4.26: Le due versioni dell'interfaccia di Memory Restoration a confronto

4.9 Menu

Un altro elemento che è stato necessario differenziare fra le due piattaforme è il menù delle opzioni, questo perché l'utente medio mobile desidera avere accesso ad un numero di opzioni estremamente ridotto e semplificato, a differenza di un utente PC, opzioni come il FOV (Field Of View), considerate obbligatorie su PC vengono del tutto ignorate su mobile o addirittura viste come fastidiose.

Per questo motivo è stato necessario realizzare due pagine delle opzioni completamente differenti, dove, in quella mobile è contenuta la possibilità di variare la qualità grafica tramite la scelta di tre profili (fast-balanced-beautiful) e poco più, mentre nella pagina relativa al PC sono contenute informazioni dettagliate quali la risoluzione delle ombre in gioco o il livello di Anti Aliasing.

4.10 Testing

Per verificare il corretto funzionamento dei componenti si sono utilizzate diverse tecniche di testing, per prima cosa si è inserito un controllo dei parametri passati alle funzioni, esso si assicura che il valore di tali parametri sia compreso nel range di quelli accettabili, altrimenti, annota l'errore nel log di debug, descrivendo il problema.

Per verificare il corretto transito degli oggetti fra i vari stati, si è introdotto un AIModel configurato appositamente per inviare all'oggetto target una lista di messaggi, contenenti tutti gli elementi da esso richiesti, questo, unito ad una funzione di log implementata all'interno di tutti gli AIModels sviluppati, ne ha permesso il testing in tempi rapidi. Lo stesso approccio è stato utilizzato, quando possibile, anche per il testing di alcuni sottogiochi (quale ad esempio Hidden Objects).

Per alcuni aspetti del gioco però, non è stato possibile introdurre metodi per il testing automatico, ed è proprio in questi casi che la disponibilità di un gruppo di tester capaci, si è rivelata molto importante.

Durante l'intero sviluppo di Adrift si è proceduto alla realizzazione di prototipi funzionanti per entrambe le piattaforme, con cadenza regolare (due settimane), questi prototipi sono stati consegnati di volta in volta ad un ristretto gruppo di tester i cui feedback sono stati utilizzati per migliorare il prodotto.

Ogni prototipo realizzato ha richiesto all'incirca un giorno di lavoro, rallentando così lo sviluppo generale, ciò nonostante si è convinti che il tempo dedicatovi abbia permesso di risolvere sul

nascere problematiche che avrebbero rischiato di intaccare profondamente la qualità e lo sviluppo del prodotto nelle fasi successive.

4.11 Creazione della demo

Una volta terminata la fase di sviluppo dell'architettura ed avendo quindi a disposizione tutti gli elementi principali del videogioco (realizzati in maniera più o meno definitiva), ci si è concentrati sulla realizzazione della demo.

Una demo può essere considerata come una versione completa di una porzione di un software, la quale viene utilizzata dalla software house per verificare le meccaniche principali del videogioco che si sta realizzando, ed inoltre testare l'interesse del mercato (solitamente le demo vengono rilasciate diversi mesi prima il prodotto completo).

Nel caso di Adrift si è deciso, che per rappresentare le potenzialità del titolo fosse sufficiente una porzione del gioco composta da un livello della nave, nello specifico il terzo (quello in cui il giocatore si risveglia), si è quindi proceduto dando a questa parte dell'ambientazione la massima priorità, utilizzando i restanti livelli per testare i modelli e le nuove variazioni di gameplay.

Nel dettaglio, l'ambiente di gioco della demo contiene:

- **Camere criogenica:** La camera in cui il giocatore inizialmente si risveglia, al suo interno si troverà il resto dell'equipaggio ancora in ibernazione e i primi sottogiochi.
- **Camera di stoccaggio:** In questa stanza il giocatore raccoglierà diversi oggetti e sperimenterà nuovi sottogiochi.
- **Ponte della nave:** Uno degli ambienti più ampi e complessi dell'intero gioco.
- **Stanza dei generatori:** Un'altra stanza dedicata principalmente all'introduzione di altri sottogiochi.
- **Stanza della manutenzione:** Al suo interno il giocatore verrà a conoscenza della possibilità di assemblare oggetti, diversificando ulteriormente il gameplay.
- **Serra:** L'ambiente più complesso e dettagliato dell'intero gioco, nella versione completa di Adrift sarà raggiungibile solo dopo metà avventura, ma si è deciso di inserirla comunque nella demo.

Lo sviluppo è proceduto senza problemi, richiedendo all'incirca un mese di lavoro e portando alla realizzazione di una demo contenente:

- Un livello della nave composto da 6 ambientazioni diverse
- 8 tipologie di sottogiochi differenti
- 94 modelli diversi
- 43 elementi con cui interagire
- 16 sottogiochi completabili
- Una durata complessiva del gioco di circa 45 minuti.
- 609 Scripts
- 73 Materiali

4.12 Screenshots

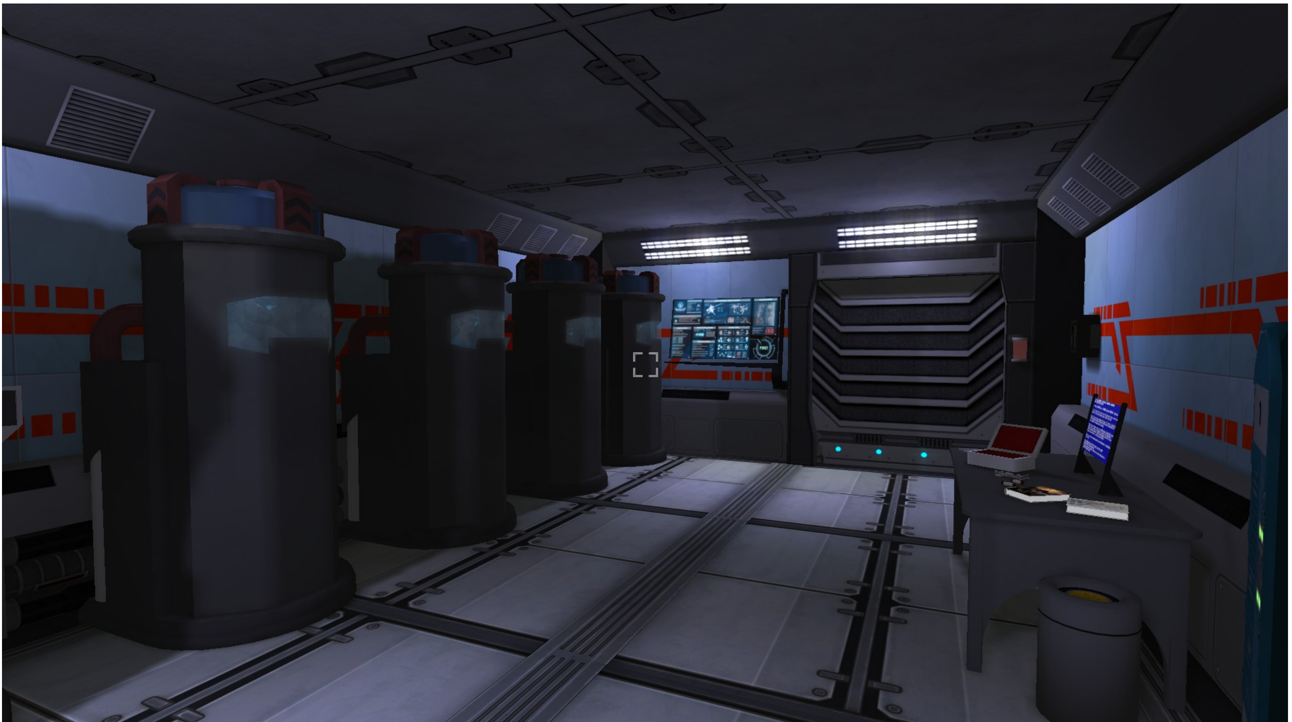

Illustrazione 4.27: Cryo room

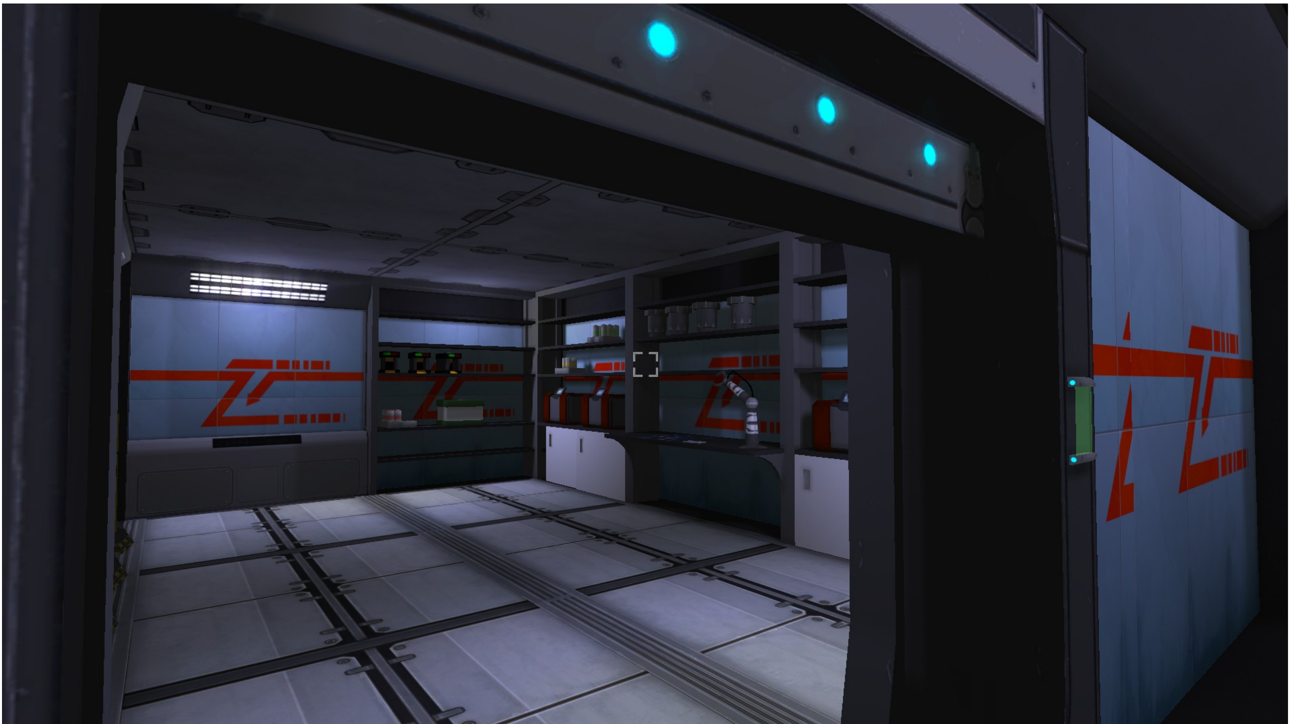

Illustrazione 4.28: Storage room

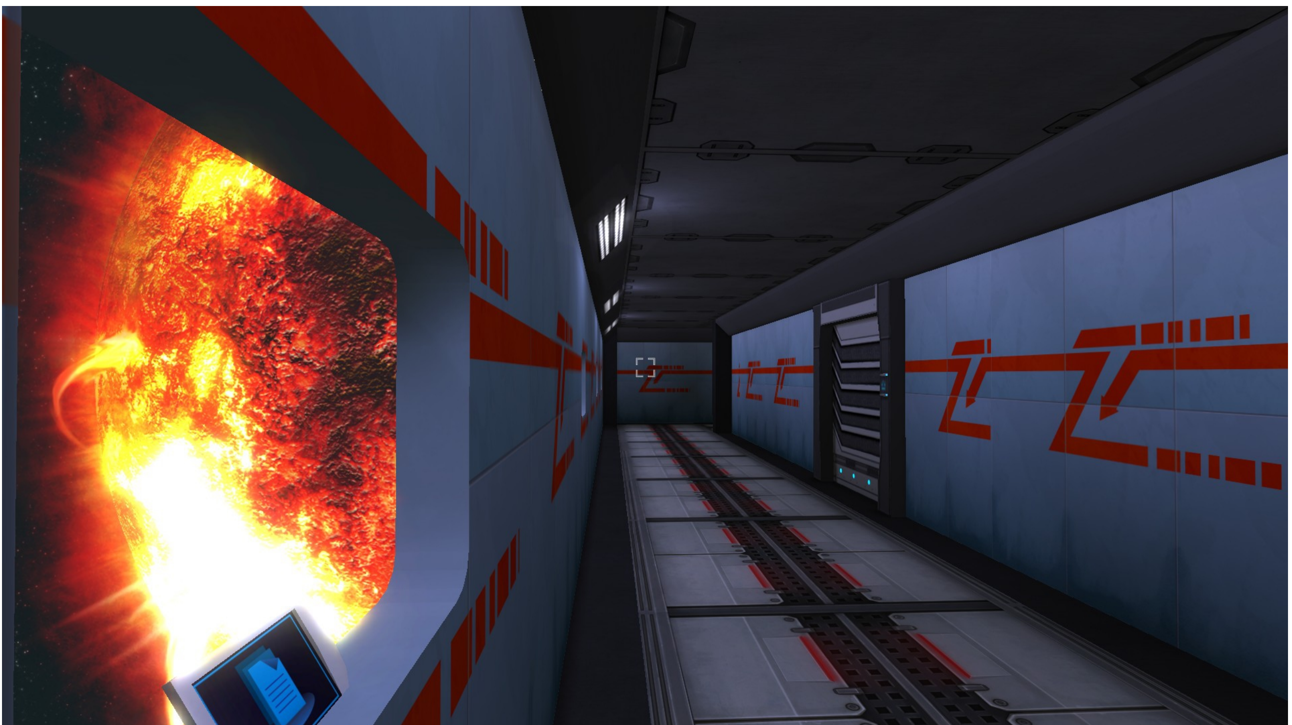

Illustrazione 4.29: Corridoio della nave

Illustrazione 4.30: Serra

Illustrazione 4.31: Sottogioco Packet Routing

5 CONCLUSIONI

Nel capitolo di chiusura si analizzano i risultati raggiunti e si descrivono quali saranno gli sviluppi futuri del gioco.

5.1 I risultati raggiunti

Ciò che ci si era prefissati è stato raggiunto appieno, la demo risulta stabile su entrambe le piattaforme, e permette agli utenti di provare tutte le caratteristiche principali del gioco.

Sono stati realizzati all'incirca il 70% dei modelli e delle texture richieste per il gioco completo, inoltre, sono già state scritte tutte le classi principali, e non si prevede la necessità di introdurre sostanziali modifiche al gameplay.

Il gioco è stato fatto testare ad una ristretta comunità di giocatori esperti (sia mobile che pc), ed ha riscosso pareri estremamente positivi, sia per quanto riguarda la qualità del prodotto, che la sua leggerezza.

La scelta di utilizzare un Game Engine come ShiVa si è rivelata corretta, permettendo di realizzare un prodotto piuttosto ampio e variegato in pochi di mesi.

5.1.1 Prestazioni

Una volta terminata la demo e finita la fase di ottimizzazione, il gioco è stato testato su diversi dispositivi al fine di verificarne il corretto funzionamento e le prestazioni su piattaforme aventi configurazioni diverse.
I test sono stati eseguiti sulle seguenti piattaforme.

Personal Computer

	Requisiti Minimi	Requisiti Raccomandati
CPU	Intel Core i3 330M	AMD FX 6300
GPU	ATI Mobility HD 5430	AMD HD 6850
RAM	4Gb	8Gb

Impostazioni video utilizzate

	Risoluzione	Anti Aliasing	Bloom	FOV	Anisotropic
Requisiti minimi	1366x768	4x	Si	90	x8
Requisiti racc.	1920x1080	4x	Si	90	x8

Mobile

	Requisiti Minimi	Requisiti Raccomandati
CPU	Dual-core 1 GHz Cortex-A9	Quad-core 2.5 GHz Krait 400
GPU	ULP GeForce	Adreno 330
RAM	1Gb	3Gb

Impostazioni video utilizzate

	Risoluzione	Anti Aliasing	Bloom	FOV	Anisotropic
Requisiti minimi	1280x800	No	No	60	No
Requisiti racc.	1920x1080	No	Si	60	No

Dai risultati dei test eseguiti, risulta chiaro come le prestazioni che ci si era imposti come obiettivo siano state raggiunte, e in alcuni casi, ampiamente superate, nel caso dei dispositivi mobile il frame rate si è dimostrato accettabile su piattaforme aventi i requisiti minimi indicati, andando invece a superare le aspettative sui dispositivi raccomandati, nei personal computer invece, le prestazioni hanno superato di gran lunga le aspettative su entrambe le configurazioni, suggerendo così la possibilità di aumentare notevolmente la quantità e la qualità dei modelli e degli effetti in gioco.

	Memoria Occupata	FPS - Requisiti minimi	FPS - Requisiti raccomandati
Personal Computer	~90 MB	100 - 200	200-490
Mobile	~50 MB	18 - 35	50 - 60

5.2 Riflessioni sullo sviluppo multi-piattaforma

L'obiettivo di questa tesi (oltre la realizzazione di un videogioco), era quello di verificare la fattibilità dello sviluppo, da parte di una casa indie, di un videogioco multi-piattaforma mediamente complesso, ed eventualmente identificare le principali problematiche legate al suo sviluppo, qui di seguito sono descritte le riflessioni a riguardo.

Attraverso l'uso della tecnologia più indicata, ed un'attenta fase di progettazione iniziale, lo sviluppo di titoli mediamente complessi, per piattaforme anche estremamente diverse quali quelle scelte, risulta affrontabile da una casa indie o da un singolo sviluppatore.

La fase di progettazione, deve includere un'attenta analisi critica delle capacità di ogni elemento del team e di ogni tecnologia che si intende utilizzare, dato che, ad esempio, la scelta sbagliata del motore grafico può facilmente determinare il fallimento di un intero progetto, oppure, sovrastimare le capacità di uno o più componenti del team può facilmente portare a sottostimare i tempi di

sviluppo necessari, causando la creazione di un prodotto non qualitativamente adeguato al mercato, risulta inoltre estremamente importante poter fare affidamento su un gruppo di tester esperti, in grado di identificare facilmente eventuali lacune nel gameplay, prima che esse diventino troppo complesse o troppo radicate nel nucleo del progetto per essere corrette facilmente.

La problematica maggiore dello sviluppo multi-piattaforma, che si sente di dover sottolineare, è lo sviluppo di una meccanica di gameplay adatta alle diverse tipologie di videogiocatori target, dato che, ciò che viene apprezzato in un titolo tende a variare di piattaforma in piattaforma, per questo, risulta cruciale al momento della definizione delle caratteristiche del prodotto che si intende sviluppare, riuscire a definire una meccanica di gioco in grado di offrire uno stile apprezzato da tutte le piattaforme su cui si ha intenzione di effettuare la release.

5.3 Piano di lavoro post laurea

Nei prossimi mesi si intende terminare lo sviluppo di Adrift e rilasciarlo sul mercato, inizialmente per piattaforma mobile (entro l'estate), per poi dedicarsi alle ultime ottimizzazioni per PC prima della sua release.

In generale tutte le caratteristiche principali del gioco sono già state implementate durante lo sviluppo della demo, perciò, ci si concentrerà unicamente sul realizzare le risorse ancora mancanti, ed estendere le funzionalità delle classi già implementate.

Alcuni degli interventi che si vogliono effettuare prima della versione finale sono riportati qui di seguito.

5.3.1 Miglioramento della classe adibita alle cut scene

Attualmente, le cut-scene realizzabili attraverso la classe implementata, risultano piuttosto statiche, basandosi unicamente sull'utilizzo di telecamere fisse, ciò che si ha intenzione di fare in futuro è sviluppare un sistema che permetta di definire parametri, quali una serie di coordinate nello spazio e la velocità di rotazione della telecamera, per poi lasciare al sistema il compito di effettuare le riprese, producendo uno stile simile alle panoramiche.

5.3.2 Estensione della classe Comp2dAI

Al fine di produrre sottogiochi sempre più complessi e dettagliati, risulta necessario che la classe adibita alla gestione degli elementi bidimensionali venga estesa ulteriormente, con funzioni quali,

ad esempio: la possibilità di collegare suoni a determinati stati dei componenti, la possibilità di effettuare il controllo delle collisioni sui singoli pixel, e non come avviene adesso, solo sulla base delle bounding box, ed ancora, realizzare effetti di rotazione, trasparenza e movimento più raffinati di quelli attuali.

5.3.3 Miglioramento della navigabilità dei menu tramite gamepad

Attualmente i menù e le varie interfacce in gioco risultano già navigabili tramite gamepad, ma la maggior parte degli elementi può essere raggiunta solamente tramite il cursore mosso con lo stick destro, risultando così in una navigazione piuttosto complessa e lenta.

Ciò che si ha intenzione di fare è introdurre la possibilità di spostarsi di elemento in elemento tramite movimenti orizzontali e verticali, semplicemente utilizzando lo stick sinistro, rendendo così la navigazione più rapida e precisa per tutti quegli utenti PC che preferiscono il gamepad a mouse e tastiera, e portando inoltre Adrift un passo più vicino ad un eventuale release per console.

5.3.4 Sottogiochi multi classe

Un'altra introduzione importante che si intende implementare, per permettere la realizzazione di sottogiochi più ampi e complessi, è la possibilità di avere più classi collaboranti allo stesso sottogioco.

Attualmente, come spiegato in precedenza, i sottogiochi vengono realizzati in maniera del tutto procedurale, con una sola classe che gestisce l'interazione di tutti i componenti in gioco.

Questo approccio permette la realizzazione rapida di giochi poco complessi, ma risulta estremamente ostico nel momento in cui si decida di sviluppare meccaniche di gioco più complesse, per far fronte a ciò si intende sviluppare un sistema in cui una classe principale gestisca lo stato del sottogioco, e svariate classi secondarie gestiscano i vari componenti in esso, simulando un approccio più object oriented.

5.3.5 Miglioramento del motore delle collisioni 2D

Un'altra funzione che si ha intenzione di raffinare è quella che gestisce il motore delle collisioni 2D.

Attualmente le collisioni vengono calcolate per ogni oggetto attivo in gioco con ogni altro oggetto, questo approccio può andare bene per sotto giochi in cui il numero di elementi totali risulti

contenuto, ma rischia di danneggiare le prestazioni nel caso di sottogiochi con un alto numero di elementi.

Per far fronte a questo sarà necessario implementare una gestione delle collisioni divisa in zone, facendo in modo che gli oggetti presenti in una determinata zona, controllino le collisioni solo con gli altri oggetti della stessa zona e limitrofe, così facendo si ridurrà sensibilmente il numero di controlli effettuati ad ogni iterazione.

5.3.6 Sistema di salvataggio\caricamento delle partite

Prima della release finale sarà necessario introdurre un sistema di salvataggio e caricamento delle partite.

Questo probabilmente si farà realizzando una classe apposita *SaveAI*, la quale, al momento del salvataggio di una partita, scorrerà tutti i modelli presenti in gioco, e, per ognuno di essi, salverà lo stato di tutte le classi ad esso collegate, nonchè il valore di tutte le loro variabili.

Nel momento in cui si vorrà caricare una partita, *SaveAI* procederà riapplicando ad ogni modello lo stato delle sue classi ed il valore delle variabili descritti all'interno del file di salvataggio.

Per l'archiviazione dei valori si utilizzerà la funzione offerta da Shiva "`saveCurrentUserEnvironment`", la quale permette il salvataggio di tutte le variabili inserite all'interno dell'ambiente di gioco, in un file di formato STK, leggibile e modificabile solo internamente alla partita (in modo da impedire al giocatore di "barare", modificando i propri salvataggi).

5.3.7 Introduzione di un secondo metodo per la creazione di sottogiochi bidimensionali

La realizzazione del sottogioco Packet Routing ha evidenziato come, tipologie di gameplay più action\arcade (cioè in cui, il fulcro della partita, consiste nell'interazione dei componenti fra di essi, influenzati solo parzialmente dall'input del giocatore) risultino molto complesse da implementare attraverso la classe per la gestione del 2D che si è realizzata.

Come detto in precedenza, il sistema utilizzato fino ad ora per i sottogiochi bidimensionali è basato sull'utilizzo di una form e dei suoi componenti per rappresentare gli oggetti di gioco.

L'aver scelto questo approccio ha permesso di iniziare rapidamente lo sviluppo dei sottogiochi, potendo già contare su funzioni complesse quali ad esempio: il clipping dei componenti contenuti nelle form, la gestione di sotto componenti all'interno di list box, o ancora, una gestione rapida del

draw order; tutto questo però a discapito di altre funzionalità offerte dal motore di gioco, escluse automaticamente durante l'utilizzo delle form perché considerate non necessarie.

Per far fronte a questo si è deciso in futuro di introdurre una classe adibita alla simulazione di interfacce bidimensionali attraverso l'uso di elementi a tre dimensioni, ciò che farà questa classe sarà creare piani perpendicolari alla camera di gioco, le cui textures saranno composte dagli sprite del sottogioco, così facendo si simulerà la bidimensionalità degli oggetti mantenendo però al contempo funzionalità quali: la gestione delle collisioni (molto più precisa di quelle che si è introdotta), la possibilità di applicare sensori ai piani, potendo così gestire più facilmente il sorgere di eventi, ed ancora la possibilità di avere facilmente più classi collegate allo stesso modello.

Tutto questo a discapito però delle funzionalità descritte in precedenza, per questo motivo, si prevede il mantenimento di entrambi i metodi, utilizzando quello già realizzato per la produzione di sottogiochi di stampo più Adventure, in cui cioè la maggior parte del gameplay viene definito dall'interazione del giocatore con gli oggetti, adibendo invece, il sistema appena descritto, alla realizzazione di tutti quei sottogiochi di stile più arcade\action.

5.3.8 Preparazione del gioco alla release sul market Android

Attualmente l'Android market permette il caricamento di applicazioni (nel singolo file .apk) con una dimensione massima di 50 Mb, mettendo poi a disposizione degli sviluppatori due slot di memoria extra da 2 Gb l'uno.

Dato che la sola demo di Adrift raggiunge quasi i 50 Mb, sarà necessario in futuro aggiungere le nuove risorse non internamente all'applicazione, ma invece caricandole in un file separato (nel formato runtime package di Shiva .stk), configurando poi il gioco in modo da fargli acquisire le risorse necessarie da esso.

Il file in questione verrà poi caricato alla release negli spazi di memoria extra del market e scaricato automaticamente quando il giocatore avvierà il download del gioco.

5.3.9 Modifica del sistema con cui vengono gestite le lightmap

Il sistema con cui Shiva gestisce le lightmap risulta al momento estremamente limitato, impedendone il caricamento e la modifica a runtime e persino la gestione delle dimensioni tramite profili, come invece possibile per il resto delle texture, questo limita di conseguenza le dimensioni massime utilizzabili per esse alla risoluzione massima supportata dal dispositivo meno performante

per cui si intende rilasciare il gioco.

Il problema è già stato fatto presente agli sviluppatori del game engine e ci si augura che venga risolto entro la prossima release.

Bibliografia

1: Statista, Video games revenue worldwide from 2012 to 2015 (in billion U.S. dollars), 2013, www.statista.com/statistics/278181/video-games-revenue-worldwide-from-2012-to-2015-by-source/

2: Will Freeman, How film and TV directors are making the jump to games, 2014, www.develop-online.net/analysis/how-film-and-tv-directors-are-making-the-jump-to-games/0195532

3: The ESA, Games and Family Life, 2014, www.theesa.com/wp-content/uploads/2014/11/Games_Familes-11.4.pdf

4: Lisa Galarneau, The State Of Gaming, 2014, http://www.bigfishgames.com/blog/2014-global-gaming-stats-whos-playing-what-and-why/

5: David Silverman, 3D Primer for Game Developers: An Overview of 3D Modeling, 2013, http://gamedevelopment.tutsplus.com/articles/3d-primer-for-game-developers-an-overview-of-3d-modeling-in-games--gamedev-5704

6: Bates, Bob, Game Design, 2004

7: Mirco Baragiani, Programmazione LUA: La guida completa per Mac, Windows e Linux, 2014

8: Gilliard Lopes,Rafael Kuhnen, Design Cognition: The Bottom-Up And Top-Down Approaches, 2015

8: Andrew Price, Blender Tutorials, 2012, www.blenderguru.com

10: Wiki, Blender Wiki, 2014, http://wiki.blender.org/index.php/Main_Page

11: Steam, Steam Hardware & Software Survey, 2015, http://store.steampowered.com/hwsurvey

12: Katsbits, optimising models for use, 2013, www.katsbits.com/tutorials/blender/basic-optimise-models-for-export-to-games.php

Ringraziamenti

Desidero innanzitutto ringraziare il Dott. Mirko Ravaioli ed il professor Matteo Golfarelli per avermi dato la possibilità di realizzare questa tesi, e per il prezioso aiuto offertomi durante il suo intero svolgimento.

Ringrazio poi la professoressa Damiana Lazzaro, grazie alla quale, durante la tesi triennale ho iniziato il percorso che mi ha portato qui oggi.

Intendo poi ringraziare i miei collaboratori, Filippo Maroni e Federico Fucci per avermi aiutato nello svolgimento ed i miei amici, in particolare Luca Polverelli per essere sempre stati disponibili con consigli e pronti a testare gli innumerevoli prototipi creati in questi mesi.

Infine, ho desiderio di ringraziare i miei famigliari, per il sostegno, la pazienza e la fiducia che mi hanno sempre dimostrato, ed in particolare Francesca, senza cui non avrei mai avuto il coraggio di imbarcarmi in un progetto così ambizioso.