Università degli Studi di Milano

Facoltà di Scienze e Tecnologie

Dipartimento di Informatica
Giovanni Degli Antoni

Corso di Laurea Magistrale in Informatica

Piattaforma di cloud gaming per giochi arcade

Relatore: Prof. Dario Maggiorini
Correlatore: Prof. Davide Gadia

Tesi di Laurea di:
Michele Maione
Matr. Nr. 931468

Anno Accademico 2020-2021

"Se non così, come? E se non ora, quando?"

—Primo Levi

Ringraziamenti

Rivolgo il primo ringraziamento al prof. Dario Maggiorini per il suo continuo supporto e per avermi proposto questo progetto con la codebase di C/C++ più grande e ben strutturata che abbia mai compilato e modificato. È stata un'esperienza entusiasmante e altamente formativa.

Ringrazio il prof. Davide Gadia per la sua guida utile durante i miei sforzi verso il completamento della presente tesi.

Milano, ottobre 2021

Sommario

Negli ultimi anni sono apparse molte piattaforme che sfruttano il paradigma del cloud computing per offrire servizi accessibili su richiesta e da remoto per archiviare file, utilizzare le suite per l'ufficio, vedere film e serie TV, ascoltare musica e a partire dal 2011 anche giocare.

Il cloud gaming è un servizio che unisce il cloud computing e il live streaming per rendere possibile giocare in remoto senza scaricare o installare il gioco sul device dell'utente, in pratica consente di archiviare ed eseguire i videogiochi su un server remoto e trasmettere l'output audio-video all'utente sul proprio dispositivo.

Per far conoscere alle nuove generazioni i videogiochi che hanno fatto la storia e dare la possibilità di poter giocare ancora a macchine che ormai hanno cessato di funzionare per motivi di obsolescenza, sfruttando due tecnologie entrate a far parte della quotidianità, il live streaming e il cloud computing, in questo lavoro si propone la creazione di una piattaforma di cloud gaming. La piattaforma permetterà lo streaming audio-video, direttamente e su richiesta, dei videogiochi da un server remoto ad un client (computer, console e telefono). Il gioco è archiviato, eseguito e renderizzato su un server remoto; l'input (tastiera e gamepad) viene inviato dal client al server e lì processato. Il cloud gaming permette d'iniziare a giocare immediatamente poiché il gioco è già installato sul server offrendo agli utenti un rapido accesso indipendentemente dal sistema operativo e dalle capacità hardware del client utilizzato. Infine la piattaforma, indirettamente, garantisce la gestione dei diritti digitali (DRM) per gli editori. Per questo progetto verrà ampliato il software MAME (rilasciato sotto licenza GNU-GPL) che è in grado di emulare oltre 7.000 giochi arcade, in modo che possa fungere da server di cloud gaming e comunicare con un front-end HTML, rimanendo sempre indipendente dal sistema operativo, così da rendere più agevole l'installazione di uno stand per il retrogaming.

Indice

Introduzione

Nel 1972 la società Atari pubblicava il primo videogioco della storia, *Pong*, vendendo 19.000 cabinati e presto molte altre società seguirono l'esempio. Alla fine del decennio iniziò l'epoca d'oro dei videogiochi arcade e la nascita delle console [Acks et al. 2020]. I videogiochi uniscono narrativa, animazione e musica all'interattività, ed è grazie a quest'ultima che riescono ad esercitare un potenziale d'immersione e attrazione che gli altri media non hanno, tanto da diventare un fenomeno culturale di massa con centinaia di milioni di persone che giocano regolarmente ogni giorno, rendendoli attori dominanti nel settore dell'intrattenimento. L'importanza economica dei videogiochi arcade, negli ultimi vent'anni, è notevolmente diminuita a favore dei videogiochi per personal computer, console e più recentemente per mobile.

Il cloud computing è un paradigma a cui siamo ormai abituati e ci risulterebbe difficile abbandonare servizi come Dropbox, Office 365, Spotify e Netflix per tornare alle loro versioni "precedenti": i rullini fotografici, i documenti aziendali negli archivi, i CD audio ed i DVD a noleggio. Dalla nascita dei primi servizi di cloud computing nel 2006 alcuni oggetti sono stati sostituiti con la loro controparte informatica portando al fallimento di aziende storiche come Blockbuster (nel 2013), Kodak (nel 2012) e Borders[1] (nel 2011) [Pitozzi 2019]. Il cloud computing consiste nella distribuzione on-demand delle risorse IT tramite internet su tre livelli di servizio che sono: l'accesso all'infrastruttura hardware tramite API (IaaS), la piattaforma software inclusa di sistemi di sviluppo (PaaS) e le applicazioni (SaaS). Con il cloud computing la potenza della macchina, sia essa fisica o virtuale, aumenta automaticamente all'esigenza permettendo di gestire i picchi di utilizzo; svincola gli utenti dal dover acquistare, manutenere e gestire fisicamente le infrastrutture IT; fornisce l'accesso alle risorse informatiche da qualsiasi device, da qualsiasi luogo e in modo collaborativo.

Unendo il paradigma del cloud computing con lo streaming nasce un nuovo tipo di servizio dedicato ai videogiochi: il cloud gaming. Questo servizio rende possibile giocare in remoto senza scaricare o installare il gioco sul device dell'utente. Con il cloud gaming i videogiochi sono archiviati ed eseguiti su un server remoto e l'output audio-video trasmesso al dispositivo dell'utente, permettendo di iniziare a giocare immediatamente, indipendentemente dal sistema operativo e dalle capacità hardware del dispositivo utilizzato. Infine, indirettamente, viene garantita la gestione dei diritti digitali (DRM) per gli editori. Il cloud gaming è l'unione di due modelli del cloud computing: il modello SaaS e il modello PaaS sia perché il videogioco viene offerto al giocatore come "applicazione" sia perché allo sviluppatore viene offerto il sistema operativo e i vari SDK, questo paradigma implementato dal cloud gaming è definito "gioco come servizio" (GaaS) che si divide in tre istanze: rendering remoto (RR-GaaS), rendering locale (LR-GaaS), allocazione delle risorse cognitive (CRA-GaaS) [D'Angelo, Ferretti e Marzolla 2015].

Lo scopo di questa tesi è creare una piattaforma di cloud gaming per far conoscere alle nuove generazioni i videogiochi che hanno fatto la storia e dare la possibilità di poter giocare ancora a macchine che ormai hanno cessato di funzionare per motivi di obsolescenza. Per far ciò verrà

[1] Borders Group era un rivenditore americano di libri e musica.

ampliato il software MAME (rilasciato sotto licenza GNU-GPL) che è in grado di emulare oltre 7.000 giochi arcade, in modo che possa fungere da server di cloud gaming e comunicare con un front-end HTML, rimanendo sempre indipendente dal sistema operativo, rendendo più agevole l'installazione di uno stand per il retrogaming.

Per la realizzazione del progetto sono state prese in considerazione le analisi fatte sulle piattaforme delle multinazionali come GeForce Now, Stadia e PlayStation Now in [Domenico et al. 2020] e [Maggiorini et al. 2016], come Amazon Luna in [Orland 2020], ma anche progetti open source come "Games on Demand" di [Karachristos, Apostolatos e Metafas 2008] che propone una piattaforma basata sul hooking di funzioni DirectX, codifica in MPEG-2 e trasmissione tramite UDP; "GamingAnywhere" di [Huang et al. 2014] che è un progetto multipiattaforma che trasmette tramite protocollo RTP ed esegue la cattura audio-video utilizzando la libreria SDL tramite polling.

Il sistema proposto è stato progettato con un'ottica incentrata sull'utilizzo in LAN con l'utenza connessa tramite WiFi, ad esempio in stand di retrogaming ad eventi di informatica e videogiochi, in aziende come servizio di svago per i clienti in sala d'attesa e per i dipendenti durante la pausa, ecc..., è importante ricordare che i videogiochi, nonostante siano stati pensati come fonte d'intrattenimento, migliorano diversi tipi di abilità chiave: abilità sociali e intellettuali, riflessi e concentrazione [Suznjevic e Homen 2020]; per questo motivo la piattaforma può essere installata anche nelle scuole.

Per ampliare il progetto MAME, lato server ho modificato le funzionalità di rendering video e missaggio audio per convogliare il loro output, che viene codificato in MPEG-TS, ad un modulo che esegue lo streaming tramite il protocollo WebSocket ad una pagina HTML. Lato client ho creato un modulo JavaScript per gestire l'input utente e decodificare il filmato MPEG-TS. La piattaforma utilizza un bit-rate tra $0,5$ Mbps e $3,5$ Mbps ed offre una risoluzione di 480p; per valutarne la latenza è stata testata su rete locale e su rete internet [Popovic et al. 2016]; mentre la qualità audio-video è stata classificata tramite "peek signal-to-noise ratio", "structural similarity index method" e "perceptual evaluation of audio quality" [Shea et al. 2013].

La tesi è strutturata nel modo seguente:

- il capitolo 1 fornisce un'introduzione sulla nascita dei videogiochi e dei ricavi globali dell'industria videoludica, dà una definizione di cloud computing e di cloud gaming, fà una panoramica delle piattaforme di gioco che si sono susseguite nel tempo e delle proiezioni di mercato del settore del cloud gaming;
- nel capitolo 2 verrà descritto il sistema proposto, il MAME e le sue funzioni di: rendering, missaggio audio e gestione dell'input utente;
- nel capitolo 3 verranno descritte le cinque fasi aggiuntive del cloud gaming e la loro implementazione in C++ come nuovi moduli del MAME: la cattura audio-video, la codifica, la trasmissione, la decodifica e la gestione dell'input utente;
- il capitolo 4 analizza le prestazioni del progetto relativamente ai tre difetti intrinseci del cloud gaming: riduzione della qualità audio-video, bit-rate richiesto e il problema della latenza;
- nelle conclusioni si riassumono gli scopi, le valutazioni di questi e le prospettive future;
- nell'appendice A si trova il manuale utente.

Capitolo 1

Stato dell'arte

Il capitolo che apre questa tesi fornisce un'introduzione sulla nascita dei videogiochi e dei ricavi globali dell'industria videoludica, dà una definizione di cloud computing e di cloud gaming, fa' una panoramica delle piattaforme di gioco che si sono susseguite nel tempo e delle proiezioni di mercato del settore del cloud gaming.

1.1 La nascita dei videogiochi

Nel 1952 nei laboratori dell'Università di Cambridge, come esempio a corredo di una tesi di dottorato sull'interazione uomo-macchina, fu creato *OXO*, la trasposizione del tris come gioco per computer. *OXO* è considerato tecnicamente il primo videogioco. Nel 1958 un professore di fisica del Brookhaven National Laboratory creò un gioco, *Tennis for Two*, che aveva il compito di simulare le leggi fisiche relative ad una partita di tennis, lo strumento utilizzato era un oscilloscopio.

Nel 1961, sei giovani scienziati del Massachusetts Institute of Technology su un PDP-1[1] crearono il primo videogioco a scopo di intrattenimento: *Spacewar!*.

Due mesi dopo due ingegneri elettrici, N. Bushnell e T. Dabney, terminarono la loro versione di *Spacewar!* su larga scala (1.500 copie), ma il gioco non ebbe un grande successo a causa dell'elevata difficoltà. Bushnell, dopo l'esperimento non particolarmente riuscito, decise però di insistere nel settore dando così vita alla società Atari. Il primo gioco arcade di Atari fu il primo grande successo del settore: *Pong*. Pubblicato alla fine del 1972, è un gioco che riproduce approssimativamente la meccanica del ping pong. Atari vendette 19.000 cabinati di *Pong* e presto molte altre società seguirono l'esempio [Acks et al. 2020]. Alla fine del decennio iniziò l'epoca d'oro dei videogiochi arcade e la nascita delle console (console che hanno fatto la storia in Fig. 1.1).

Figura 1.1: Console iconiche, fino alla nona generazione. Tutti i marchi riportati appartengono ai legittimi proprietari

[1]PDP-1: Programmed Data Processor-1, era un computer della Digital Equipment Corporation del 1959.

I videogiochi sono un mezzo di intrattenimento unico che combina le diverse forme d'arte, quali musica, narrativa e animazione, all'interattività. Ed è proprio questa caratteristica, l'interattività, che permette loro di esercitare un potenziale d'immersione e attrazione che altri media non hanno. Sono ormai diventati un fenomeno culturale di massa con centinaia di milioni di persone che giocano regolarmente ogni giorno, il che li rende attori dominanti nel settore dell'intrattenimento, settore in continua crescita che non ha mai subito interruzioni nel corso degli anni come mostrato in Fig. 1.2. Negli ultimi vent'anni l'importanza economica dei videogiochi arcade è notevolmente diminuita[2] (in viola nella figura) a favore dei videogiochi per personal computer, console e più recentemente per mobile.

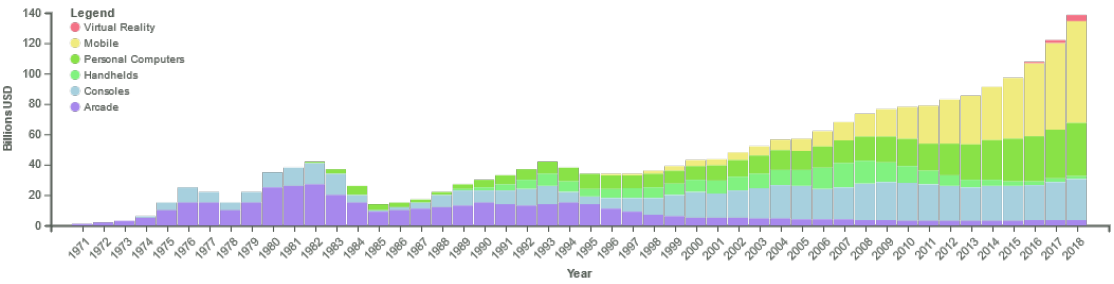

Figura 1.2: Ricavi globali dell'industria dei videogiochi dal 1971 al 2018 (non adeguati all'inflazione). Fonte: wikipedia.org

1.2 Cloud gaming

Negli ultimi anni sono apparsi molti tipi di servizi che sfruttano il paradigma del cloud computing: archiviazione (Dropbox, Drive, OneDrive), musica (Spotify, Amazon Music), cinematografia (Prime Video, Netflix), documenti (Google Workspace, Office 365) e più recentemente il cloud gaming. Il cloud gaming è un servizio che unisce il cloud computing e il live streaming per rendere possibile giocare in remoto senza scaricare o installare il gioco sul device dell'utente, in pratica consente di archiviare ed eseguire i videogiochi su un server remoto e trasmettere l'output audio-video all'utente sul proprio dispositivo.

Il cloud computing è un paradigma in cui un provider offre risorse fisiche e software accessibili da remoto, tramite la sottoscrizione di un abbonamento mensile/annuale oppure "pay-as-you go" (calcolato su: spazio di archiviazione utilizzato, tempo di utilizzo, cicli di CPU/GPU, ecc...), le cui caratteristiche essenziali sono:

- la capacità di fornire risorse hardware (processori, memoria, spazio di archiviazione, ecc...) automaticamente in base alle necessità software;
- l'accesso alle risorse attraverso la rete tramite protocolli standard;
- le risorse fisiche e virtuali possono essere distribuite dinamicamente agli utenti in base alle loro richieste;
- dal punto di vista dell'utente le risorse sono illimitate e possono essere acquistate in qualsiasi quantità ed in qualunque momento;

[2] Giappone, Cina e Corea mantengono una forte industria arcade ai giorni nostri.

- l'uso delle risorse e dei servizi è ottimizzato tramite il modello "pay-per-use" ed è monitorato e controllato in modo trasparente sia dal provider che dall'utente; alcune società offrono anche abbonamenti mensili e annuali, che invece limitano le risorse ad un tetto massimo.

Il cloud computing offre tre modelli di servizio: *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS) e *Software as a Service* (SaaS). Il cloud gaming è l'unione del modello SaaS poiché i videogiochi sono offerti come un servizio dal punto di vista del giocatore, e del modello PaaS dal punto di vista dello sviluppatore che necessita del sistema operativo, delle librerie e dei kit di sviluppo; per cui il paradigma implementato dal cloud gaming è definito *Gaming as a Service* (GaaS) e si divide in tre istanze [D'Angelo, Ferretti e Marzolla 2015], come mostrato in Fig. 1.3, che sono:

- *Remote Rendering* (RR-GaaS): il gioco viene eseguito e codificato sul server ed inviato all'utente come un filmato;
- *Local Rendering* (LR-GaaS): il gioco viene eseguito, codificato sul server ed inviato all'utente sotto forma di istruzioni di rendering. Il client invia le istruzioni di rendering alla scheda grafica dell'utente;
- *Cognitive Resource Allocation* (CRA-GaaS): il client riceve moduli eseguibili del gioco che vengono eseguiti sul dispositivo dell'utente.

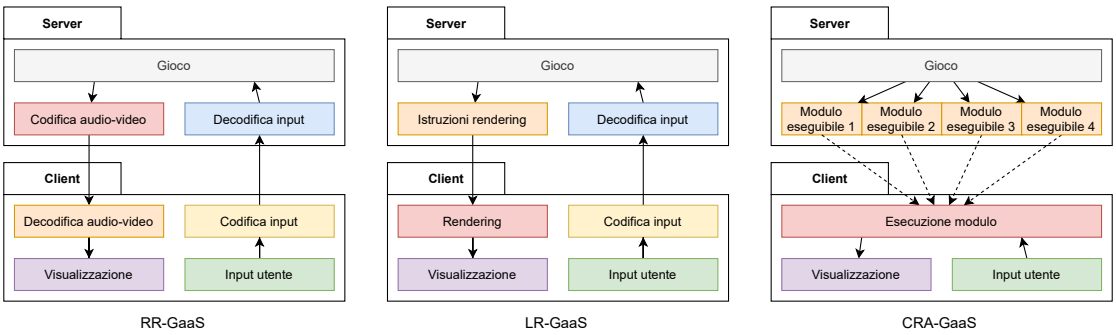

Figura 1.3: istanze di GaaS

Il rendering remoto (RR-GaaS) è attualmente l'istanza più utilizzata nelle soluzioni di cloud gaming sul mercato essendo la soluzione che ha tutta la computazione a carico del server ed offre all'utente la possibilità di usare videogiochi computazionalmente esosi su qualsiasi tipo di device. Dei tre protocolli è però quello che richiede il bit-rate maggiore. Alcuni esempi sono Stadia[3], PlayStation Now[4].

Il rendering locale (LR-GaaS) è l'istanza che richiede il bit-rate minore ma, come nel caso dell'allocazione delle risorse (CRA-GaaS), il rendering è eseguito sul device dell'utente, richiedendo così particolari caratteristiche hardware. Attualmente non ci sono importanti piattaforme sul mercato che offrono questo tipo d'istanza.

Alcune società propongono una versione dell'allocazione delle risorse cognitive (CRA-GaaS) mista al rendering remoto, in cui il gioco viene inizialmente servito come rendering remoto ed in contemporanea viene eseguito il download dei moduli eseguibili; al completamento del

[3]Stadia è una piattaforma di cloud gaming di Google.
[4]PlayStation Now è il cloud gaming targato Sony.

download dei moduli che riguardano l'attuale stato del gioco, lo streaming si interrompe e il gioco riprende dal device dell'utente; questa è una delle funzionalità che Project Atlas[5] dovrebbe offrire. Un'altra variante, che viene chiamata commercialmente come "progressive download", si basa sul concetto di scaricare inizialmente i moduli principali del gioco, solitamente il menù e il primo livello; gli store Origin[6] e Ubisoft Connect[7] implementano questa funzionalità.

Nel prossimo paragrafo vedremo le tecnologie per lo streaming maggiormente usati.

1.2.1 Tecnologie per lo streaming audio-video

I protocolli di comunicazione a livello di trasporto su cui sono costruiti servizi, tecnologie e Web API [Mozilla 2021a], schematizzati in Fig. 1.4, sono UDP, TCP e SCTP, le cui caratteristiche principali sono riassunte in Tabella 1.1.

Caratteristica	UDP	TCP	SCTP
Dimensione header	8 byte	20-60 byte	12 byte
Entità del pacchetto	datagramma	segmento	datagramma
Orientato alla connessione	no	sì	sì
Trasporto affidabile	no	sì	sì
Consegna ordinata	no	sì	sì/no
Controllo del flusso	no	sì	sì
Controllo della congestione	no	sì	sì
Flussi multipli	no	no	sì
Multihoming[8]	no	no	sì

Tabella 1.1: Comparazione tra protocolli di trasporto

Di questi solo quattro possono essere utilizzati per lo streaming utilizzando il browser web [Grigorik 2013]:

- Dynamic Adaptive Streaming over HTTP (DASH) è una tecnica di streaming con bit-rate adattivo del Moving Picture Experts Group (MPEG), che consente lo streaming di alta qualità di contenuti multimediali su protocollo HTTP;
- HTTP Live Streaming (HLS) è il protocollo di streaming ad alta latenza più popolare su HTTP per video on demand (video pre-registrato) sviluppato da Apple;
- WebSocket è un protocollo di comunicazione che fornisce un canale full-duplex su una singola connessione TCP, con una latenza inferiore rispetto ad HLS e DASH;
- Web Real-Time Communication (WebRTC) è un progetto per la comunicazione in tempo reale basato sul protocollo RTP (Real-time Transport Protocol).

[5] Project Atlas è un progetto di una piattaforma cloud gaming mista ad intelligenza artificiale della Electronic Arts.
[6] Origin è una piattaforma di distribuzione digitale di videogiochi sviluppata da Electronic Arts.
[7] Ubisoft Connect è un servizio di distribuzione digitale della società Ubisoft.
[8] Il multihoming è la funzionalità di poter connettere un host a più di una rete.

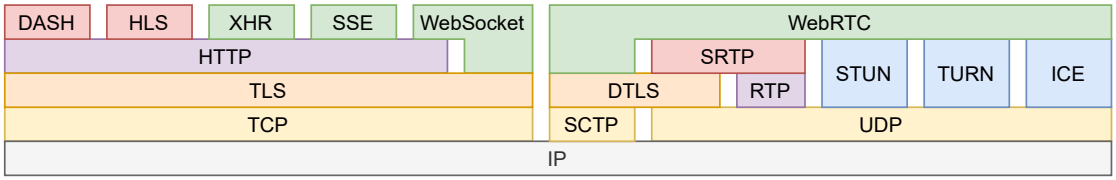

Figura 1.4: API, protocolli e servizi di rete del browser

Nel paradigma del cloud gaming la codifica audio-video avviene in real-time e il buffering non è usabile perché aumenterebbe la latenza, per questi motivi non tutte le tecnologie citate sono adatte. Come vedremo nelle caratteristiche delle piattaforme di streaming commerciali, descritte nel prossimo paragrafo, solo TCP, UDP ed RTP sono stati effettivamente usati. Solitamente la comunicazione tramite TCP è relegata alla parte di autenticazione e gestione dell'input utente. Infatti da test condotti in [Bielievtsov et al. 2018] sul frame rate in caso di perdita di pacchetti durante lo streaming, in Fig. 1.5 con una perdita di pacchetti dell'8%, si nota che con RTSP[9], schema (a) e (b), il frame rate è costante ma con alcune perturbazioni; invece HLS, schema (c), risulta essere la tecnologia che risente maggiormente della perdita dei pacchetti; mentre con RTMP[10], schema (d), la riproduzione si interrompe e il video si blocca per alcuni secondi.

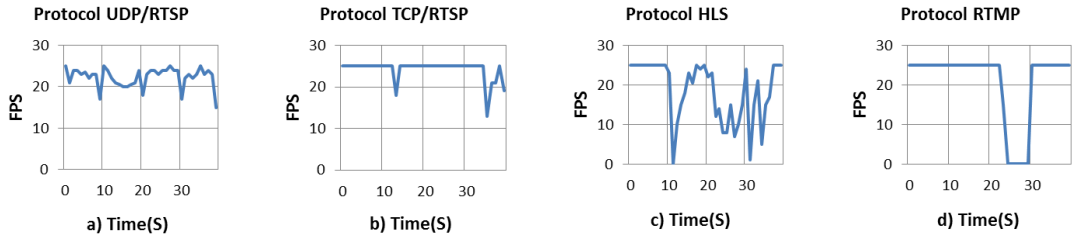

Figura 1.5: Streaming con perdita di pacchetti dell'8%.

Nel prossimo paragrafo viene fatta un'introduzione, in ordine cronologico, delle piattaforme di cloud gaming sul mercato.

1.2.2 Storia del cloud gaming

Una delle prime piattaforme di cloud gaming è stata OnLive di OL2, presentata alla GDC[11] 2009 e poi lanciata sul mercato a giugno 2010 negli Stati Uniti e a settembre 2011 nel Regno Unito. I giocatori, previo pagamento di un abbonamento mensile di 15$, potevano acquistare o noleggiare giochi sulla piattaforma oppure utilizzare quelli precedentemente acquistati su Steam[12]. Il servizio era ospitato su 5 data center situati sul suolo americano che servivano gli utenti per vicinanza geografica; il bit-rate richiesto era di $1,5$ Mbps per la qualità video SD[13] e 5 Mbps per la Stan-

[9]RTSP: Real Time Streaming Protocol serve a stabilire e gestire la sessione di streaming, la trasmissione dei dati avviene tramite RTP e la raccolta delle statistiche sulla qualità del servizio è affidata al Real-time Transport Control Protocol (RTCP).

[10]RTMP: Real Time Messaging Protocol era un protocollo sviluppato da Macromedia per il Flash player, ad oggi non più supportato dai browser.

[11]GDC: Game Developers Conference, una conferenza annuale per gli sviluppatori di videogiochi.

[12]Steam è un servizio di distribuzione digitale di videogiochi della società Valve.

[13]SD: Standard Definition include i formati video con rapporto 4:3 e 16:9 con 480 linee di risoluzione (in alcuni casi 576).

dard HD[14]. I protocolli di comunicazione utilizzati erano: TCP per il test della velocità verso i 5 data center, successivamente tramite TLS su TCP veniva fatta l'autenticazione al servizio; l'input utente veniva inviato tramite UDP e lo streaming, codificato tramite AVC[15], trasmesso tramite protocollo RTP [Manzano et al. 2014]. Era disponibile, oltre ad un client per Windows, macOS ed Android, anche una micro console da collegare alla TV. Il servizio fu acquistato ad aprile 2015 da Sony [JP Mangalindan 2020].

Al GDC 2010 Gaikai ha presentato il suo omonimo servizio di cloud gaming; A febbraio 2012 era distribuito su 24 data center e disponibile in 12 nazioni. La società si è concentrata principalmente su due modelli di business: sull'utilizzo del cloud gaming come forma di pubblicità online per i videogiochi, fornendo agli utenti la possibilità di accedere alle demo dei videogiochi sponsorizzati; fornire ad altre aziende l'infrastruttura per il cloud gaming, come per Wikipad[16], Electronic Arts e Samsung [Gaikai 2012]. La piattaforma era accessibile tramite browser (utilizzando il plugin di streaming disponibile in Adobe Flash, Java o Google Native Client), la qualità video HD richiedeva un bit-rate minimo di 5 Mbps. A luglio 2012 la società fu acquisita da Sony per integrare la loro tecnologia di streaming nella piattaforma di cloud gaming PlayStation Now [Hollister 2010].

Beijing Cloud Union al CES[17] 2012 ha presentato il suo omonimo servizio di cloud gaming con 64 videogiochi tra cui alcuni titoli per PlayStation 3, accessibile tramite browser sul PC e tramite app per smartphone. Il servizio era disponibile solo in Cina ed arrivò a 20 milioni di utenti chiudendo definitivamente a dicembre 2018 [Beijing Cloud Union 2018].

PlayStation Now è un servizio di cloud gaming basato sulla tecnologia cloud di Gaikai. È stato presentato durante il CES 2014 ed è stato reso disponibile a partire da gennaio 2015 in Nord America, da settembre in Giappone e Regno Unito ed ha iniziato a coprire il mercato europeo gradualmente a partire da agosto 2017. La piattaforma consente all'utente di giocare ai titoli PlayStation (attualmente 800 dal catalogo giochi della PS2, PS3 e PS4) su PS4, PS5 e Windows (tramite l'installazione del client "PS Now app" e l'uso di un controller compatibile). Per quanto riguarda i giochi PS2 e PS3 Sony ha costruito una scheda madre contenente la componentistica miniaturizzata di otto PS3, ognuna indipendentemente controllabile, inoltre l'hardware contiene un codificatore video AVC; lo stesso è avvenuto per i giochi PS4, utilizzando schede madri PS4 modificate ad hoc [Whitwam 2014]. Questo ha permesso di ridurre ulteriormente la latenza di cattura e codifica (argomento trattato nel Paragrafo 4.2). La risoluzione offerta è Standard HD con un bit-rate minimo richiesto di 5 Mbps e la tecnologia di trasmissione è basata su UDP [Domenico et al. 2020]. Gli abbonamenti proposti sono da 10$, 25$ o 60$ per 1, 3 o 12 mesi di utilizzo [Sony 2021].

GeForce Now è il servizio di cloud gaming di Nvidia lanciato in beta a gennaio 2017 e ufficialmente a febbraio 2020. I data center sono collocati negli Stati Uniti, Europa, Australia e Canada, e sono dotati di GPU Nvidia P40. GeForce Now consente agli utenti di accedere da remoto (tramite streaming) a un computer virtuale, dove possono installare giochi (attualmente il catalogo consta di 800 giochi) acquistati su Steam, Ubisoft Connect o Epic Games Store[18]. Il servizio può essere utilizzato su Windows, macOS, iOS, Android o Nvidia Shield TV[19]. Nvidia ha scelto di usare RTP come protocollo per lo streaming utilizzando il codec AVC, UDP per il test della velocità e l'invio dell'input dell'utente e TLS tramite TCP per l'autenticazione [Domenico

[14]Standard High Definition (chiamata anche HD ready) è un formato video 16:9 (disponibile anche con rapporto 4:3) con 720 linee di risoluzione.

[15]AVC è un formato di compressione video dello standard MPEG-4.

[16]Oggi Gamevice, è un produttore di tablet e periferiche specializzato in prodotti per il gaming.

[17]CES: Consumer Electronics Show è un evento annuale che ospita presentazioni di nuovi prodotti e tecnologie nel settore dell'elettronica di consumo.

[18]Epic Games Store è un negozio di videogiochi digitali gestito da Epic Games.

[19]Nvidia Shield TV è un lettore multimediale digitale basato su Android.

et al. 2020]. L'abbonamento mensile è di 10$ e le risoluzioni video offerte sono Standard HD e Full HD[20] con, rispettivamente, 15 e 25 Mbps di bit-rate richiesto [Nvidia 2021].

A maggio 2018 Electronic Arts ha svelato Project Atlas che va ad espandere il concetto di cloud gaming. La piattaforma mira a fornire un'esperienza di gioco nuova grazie al supporto dell'intelligenza artificiale, offrendo universi di gioco che cambiano con il passare del tempo, con l'interazione con altri giocatori e sotto l'influenza del mondo esterno. Dal punto di vista degli sviluppatori il progetto punta a far confluire il motore di gioco Frostbite[21], i servizi di gioco e l'intelligenza artificiale in una nuova piattaforma di sviluppo [Electronic Arts 2018].

Vortex della RemoteMyApp è un servizio di cloud gaming lanciato a novembre 2018, è disponibile per Android, Windows e macOS e offre tre piani mensili (12$, 23$ e 34$) che consentono all'utente di giocare per un massimo di 140 ore al mese ad un catalogo di 170 giochi. Sfortunatamente, alcuni giochi possono essere riprodotti solo acquistando la licenza del gioco. La società utilizza 13 data center con: processori Intel Xeon, 512GB di memoria RAM e schede grafiche NVIDIA. Il bit-rate richiesto è 10 Mbps per giocare in Standard HD [RemoteMyApp 2021].

Microsoft ha anticipato Xbox Cloud Gaming all'E3[22] 2018. La piattaforma è disponibile per gli abbonati a Xbox Game Pass Ultimate da settembre 2020 ed offre sia la libreria esistente di giochi per Xbox che per Xbox Series X (attualmente una selezione di 250 giochi). La piattaforma è ospitata su 54 data center Azure[23] che coprono 140 paesi [Bankhurst 2018] mentre l'hardware è basato su schede madri Xbox Series X ridisegnate ad hoc [Warren 2020b]. Il servizio è progettato per funzionare con gli smartphone (attualmente solo Android), con controlli touchscreen o usando un controller Bluetooth compatibile, ad una risoluzione Standard HD ed è richiesto un bit-rate di 10 Mbps; l'abbonamento ha un costo mensile di 10$ [Microsoft 2021c].

Google Stadia è una piattaforma di cloud gaming rilasciata a novembre 2019, ma è disponibile solo in Europa e negli Stati Uniti. Il servizio è ospitato sui data center Google che montano GPU personalizzate di AMD [Google 2019a] con supporto alle API Vulkan[24] in grado di sviluppare una potenza di oltre 10 teraflops [Google 2019b]. La piattaforma è eseguita su una versione personalizzata di Debian[25] e l'utilizzo dello Stadia SDK[26] è necessario per gli sviluppatori. Stadia è stata la prima piattaforma a sfruttare completamente WebRTC che fornisce: l'autenticazione tramite DTLS[27] e STUN[28], l'invio dell'input utente tramite DTLS e lo streaming tramite RTP, utilizzando uno dei seguenti codec video: AV1[29] , VP9[30] e AVC [Domenico et al. 2020]. Il costo mensile del servizio è di 10$ ed è accessibile tramite app su Android, tramite web app su iOS, su Chromecast[31] utilizzando il controller Stadia[32] e su computer tramite browser Chrome si può giocare con mouse e tastiera oppure usando un controller compatibile. Le risoluzioni video offerte sono Standard HD, Full HD e UHD[33], con bit-rate richiesti di 10, 25 e 35 Mbps. La piattaforma offre le seguenti funzionalità: live streaming su YouTube del proprio gameplay; "Crowd Play" che consente agli spettatori di unirsi ad una sessione di gioco in live stream (se invitati dall'host); "Stream Connect" che consente all'utente di condividere la schermata di gioco con altri giocatori

[20] Full HD è un formato video 16:9 con 1080 linee di risoluzione.
[21] Frostbite è un motore di gioco sviluppato da DICE, una società sussidiaria di Electronic Arts.
[22] E3: Electronic Entertainment Expo, un evento commerciale per l'industria dei videogiochi.
[23] Microsoft Azure è una piattaforma di cloud computing.
[24] Vulkan è un API per la grafica real-time 3D.
[25] Debian è una distribuzione Linux composta interamente da software libero.
[26] Il progetto Stadia è rilasciato sotto licenza GPL 2.0 ed è disponibile su Github.
[27] DTLS è un protocollo crittografico per UDP basato su TLS.
[28] STUN è un protocollo per il NAT traversal per comunicazioni in tempo reale.
[29] AV1 è un formato di codifica video open source della Alliance for Open Media.
[30] Google VP9 è un formato di codifica video open source.
[31] Google Chromecast è un lettore multimediale digitale per contenuti audiovisivi in streaming su Internet.
[32] Controller WiFi di Google con connessione diretta a Stadia.
[33] UHD: Ultra High Definition è un formato video 16:9 con 2160 linee di risoluzione.

nella stessa partita; "Condivisione dello stato" che consente di condividere il proprio salvataggio di gioco con gli amici [Google 2021].

Amazon Luna è stata annunciata a settembre 2020, con "accesso anticipato" a partire da ottobre 2020. Per motivi di compatibilità il S.O. scelto per la piattaforma è Windows ed è in esecuzione su istanze EC2 G4[34] in grado di sviluppare una potenza di 8,1 teraflops [Warren 2020a]. Il catalogo giochi proposto consta di più di 100 giochi ed il servizio offre l'integrazione con Twitch ed una partnership con Ubisoft che da accesso ai loro titoli al momento del rilascio. Si può accedere alla piattaforma tramite PC, Fire TV[35] e smartphone, utilizzando controller Luna[36], Xbox o PS. Come Stadia anche Luna utilizza WebRTC per l'autenticazione, la gestione dell'input utente e lo streaming [Orland 2020]. L'abbonamento è di 6$ al mese (15$ per la versione in partnership con Ubisoft) con una risoluzione Full HD e una banda richiesta di 10 Mbps [Amazon 2021].

La società Playkey ha realizzato un omonima piattaforma di cloud gaming distribuito, attualmente in alpha testing. Il sistema distribuito è formato da un server centrale che gestisce l'infrastruttura e dai computer dei cosidetti "minatori", coloro che mettono a disposizione il proprio computer come unità di calcolo del sistema distribuito, su cui viene eseguito il gioco, la codifica e lo stream tramite protocollo UDP. La piattaforma offre la Standard HD come risoluzione ed è richiesto un bit-rate di 10 Mbps. L'abbonamento è di 23$ mensili mentre i "minatori" guadagnano 10$ al giorno. Playkey da la possibilità di giocare solo i titoli precedentemente acquistati da Steam, Ubisoft Connect, Origin e Battle.net[37] [Playkey 2021].

Il caso Apple

A metà del 2020 Apple aveva cercato di bloccare le app di cloud gaming sull'App Store, ma a settembre 2020 decise di consentire il cloud gaming con alcune restrizioni: che i giochi offerti nel servizio dovessero essere scaricati direttamente dall'App Store e non da un'app all-in-one. I produttori di app sono autorizzati a rilasciare una cosiddetta "app catalogo" che si collega ad altri giochi nel servizio, ma ogni gioco dovrà essere una singola app e tutti i giochi e le "app catalogo" devono offrire l'acquisto solo tramite il sistema di elaborazione dei pagamenti "in-app purchases" di Apple, in base al quale la Apple ha un guadagno del 30% sugli acquisti fatti dall'utente [Leswing 2020].

Progetti a scopo didattico

Per la realizzazione di questo progetto sono stati presi in considerazione due progetti di cloud gaming a scopo didattico.

"Games on Demand" di [Karachristos, Apostolatos e Metafas 2008] è una piattaforma del 2007 per Windows basata sul hooking di funzioni DirectX tramite la libreria Taksi[38]. Lo streaming avviene tramite UDP utilizzando una codifica video in MPEG-2, mentre l'audio viene omesso.

GamingAnywhere di [Huang et al. 2014] è un progetto multipiattaforma del 2013 per Windows, Linux, macOS ed Android. La cattura audio-video avviene utilizzando la libreria SDL tramite polling settando un frame-rate per il video e uno per l'audio; se la velocità di aggiornamento dell'output video è maggiore del frame-rate alcuni frame intermedi vengono scartati;

[34] EC2 G4 è una istanza del cloud computing di Amazon progettata per il rendering e il machine learning. Le schede video installate sono le Nvidia T4.

[35] Fire TV è una linea di media center di Amazon.

[36] Controller WiFi di Amazon con connessione diretta a Luna.

[37] Battle.net è una piattaforma di distribuzione digitale e di gestione dei diritti digitali sviluppata da Blizzard Entertainment.

[38] Taksi è una libreria open source per la cattura video di applicazioni che utilizzano DirectX, OpenGL o GDI.

invece la cattura audio ha un altro problema, nel caso in cui il gioco non emetta nessun suono, il programma deve generare dei frame audio di silenzio. Questi due problemi in MAME CGP[39] sono stati affrontati in modo differente e verranno illustrati nel paragrafo 3.1.2. Lo streaming avviene tramite protocollo RTP, utilizzando la libreria LIVE555[40]; la codifica usata è VP8 con una risoluzione video Standard HD ed un bit-rate richiesto di 3 Mbps.

In Tabella 1.2 è riportato il riepilogo delle piattaforme disponibili ad oggi con caratteristiche, requisiti e protocollo di streaming utilizzato.

Piattaforma	Tipo	Protocollo	Risoluzione[41]	Bit-rate (Mbps)
Amazon Luna	Cloud gaming	RTP	1080	10
Games on Demand	Cloud gaming	UDP	?	?
GamingAnywhere	Cloud gaming	RTP	720	3
GeForce Now	Computer virtuale	RTP	1080	25
Google Stadia	Cloud gaming	RTP	2160	35
MAME CGP	Cloud gaming	WebSocket	480	1
Playkey	Computer virtuale	UDP	720	10
PlayStation Now	Cloud gaming	UDP	720	5
Vortex	Cloud gaming	UDP	720	10
Xbox Cloud Gaming	Cloud gaming	UDP	720	10

Tabella 1.2: Piattaforme disponibili

Nel prossimo paragrafo verranno illustrate le proiezioni di mercato relative al 2023.

Proiezioni di mercato

Secondo una ricerca di Newzoo[42] sull'industria dei videogiochi [Newzoo 2020], come mostrato in Fig. 1.6, nel 2020 il mercato del cloud gaming ha generato quasi $584,7$ milioni di USD di entrate, di cui il 39% e il 29% in Nord America e in Europa, e si prevede una crescita fino a $4,8$ miliardi di USD entro il 2023, se non maggiore. Per questo sono entrate nel mercato del cloud gaming anche aziende che non sono editrici o produttrici di videogiochi come Google e Amazon, come visto nel paragrafo 1.2.2.

Queste previsioni sono conseguenza sia di connessioni di rete, domestiche e mobili, sempre più veloci sia del paradigma del live streaming a cui oggi (soprattutto le nuove generazioni) siamo abituati. Dal punto di vista del giocatore il cloud gaming offre molti vantaggi tra cui: rendere il gioco facilmente accessibile senza la necessità di scaricarlo e installarlo localmente; compatibilità con computer, smartphone e anche con smart TV (se utilizzato con un gamepad WiFi) senza dover badare ai requisiti hardware; diverse modalità di pagamento tra cui l'acquisto di un gioco su richiesta e l'abbonamento mensile/annuale per l'utilizzo di tutta (o una parte) della libreria videoludica; funzionalità aggiuntive per sfruttare al meglio questo modello, come lo streaming della sessione di gioco, che siamo già abituati a vedere, con la possibilità di far entrare uno spettatore nella propria partita, funzionalità avanzate per il multiplayer come la condivisione della visuale di gioco e dei salvataggi di gioco, ecc.... Ci sono vantaggi anche per gli sviluppatori perché il cloud gaming riduce i costi di produzione limitando lo sviluppo e il testing ad una sola

[39] MAME CGP (Cloud Gaming Platform) è il nome che ho dato a questa versione modificata del MAME in grado di fungere da piattaforma di cloud gaming.
[40] LIVE555 Streaming Media è un set di librerie open source per lo streaming multimediale.
[41] Risoluzione video misurata in linee verticali.
[42] Società di analisi statistica del settore videoludico.

piattaforma ed inoltre risolve definitivamente un problema che esiste dai tempi delle audiocassette e dei floppy disk, la pirateria. Infine per i fornitori di servizi si viene a creare un nuovo modello di business su contenuti già esistenti. Tuttavia ci sono anche degli svantaggi, di cui parleremo nel capitolo 4: perdita della qualità audio-video a causa della compressione, bit-rate richiesto non soddisfabile dall'utente e l'ineliminabile latenza.

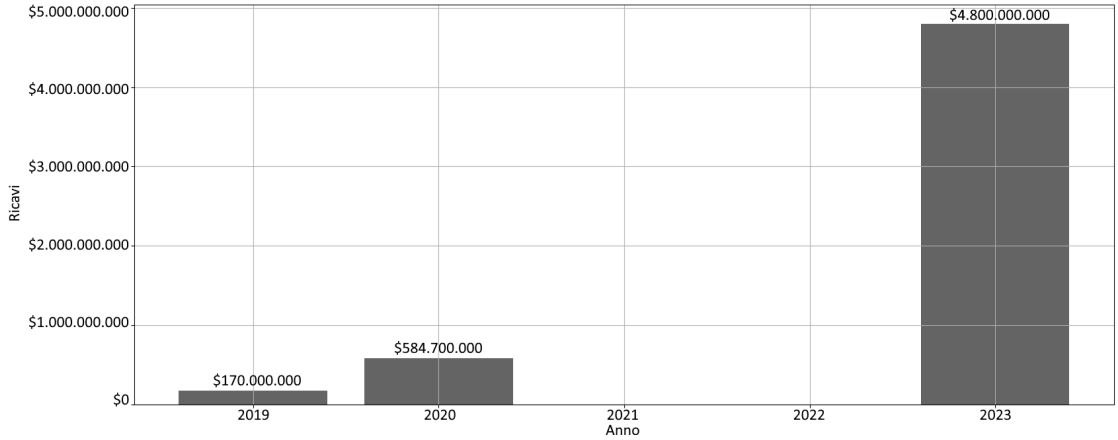

Figura 1.6: Previsioni per il mercato globale del cloud gaming (in dollari americani). Fonte: newzoo.com/global-cloud-gaming-report

Capitolo 2

Architettura del sistema

In questo capitolo verrà descritto il sistema proposto, il MAME e le sue funzioni di rendering, missaggio audio e gestione dell'input utente.

2.1 Sistema proposto

L'esigenza per la quale nasce questo progetto è far conoscere alle nuove generazioni i videogiochi che hanno fatto la storia e dare la possibilità di poter giocare ancora a macchine che ormai hanno cessato di funzionare per motivi di obsolescenza, sfruttando due tecnologie entrate a far parte della quotidianità, lo streaming e il cloud computing. In questo lavoro si propone la creazione di una piattaforma di cloud gaming, che permetta lo streaming audio-video direttamente e su richiesta dei videogiochi, da un server remoto, ad un client (computer, console, telefono). Per far ciò verrà ampliato il software MAME (rilasciato sotto licenza GNU-GPL) che è in grado di emulare oltre 7.000 giochi arcade. Le caratteristiche principali del progetto, che sono state vincolanti nella scelta delle tecnologie da utilizzare, sono la portabilità e la possibilità di utilizzare il sistema senza dover installare software aggiuntivi; per questi vincoli, lato client, la scelta è ricaduta sul browser web.

Il sistema è stato progettato con un'ottica incentrata sull'utilizzo in LAN con l'utenza connessa tramite WiFi, ad esempio in stand di retrogaming ad eventi di informatica e videogiochi, in aziende come servizio di svago per i clienti in sala d'attesa e per i dipendenti durante la pausa, nelle scuole, ecc. . . ; infatti nonostante siano stati pensati come fonte d'intrattenimento, i videogiochi migliorano diversi tipi di abilità chiave: abilità sociali e intellettuali, riflessi e concentrazione [Suznjevic e Homen 2020]. La tecnologia di streaming scelta è stata WebSocket poiché in questo contesto la differenza di velocità tra TCP e RTP può essere trascurata, è un protocollo di comunicazione standardizzato dal 2011, è pienamente supportato da tutti i browser moderni, ha una latenza inferiore rispetto ad HLS e DASH, è semplice da istanziare e non richiede l'utilizzo di protocolli aggiuntivi o configurazioni complesse a differenza di WebRTC.

Come mostrato in Fig. 2.1 il sistema è costituito dal server di gioco (Linux, macOS o Windows), su cui è installato il MAME CGP[1] con le rom dei giochi ed una pagina HTML5 che funge da front-end.

MAME CGP è un'applicazione server che rimane in ascolto in attesa di richieste di connessione da parte di un client, tramite il protocollo di comunicazione WebSocket. Quest'ultimo consente di aggiungere parametri (per es.: il nome del gioco, l'ID del player, l'ID della partita, ecc. . .)

[1] MAME CGP (Cloud Gaming Platform) è il nome che ho dato a questa versione modificata del MAME.

al URL di connessione. Una volta stabilita la connessione, il programma invia informazioni sulla risoluzione di rendering e avvia il gioco. Il rendering e il missaggio audio del gioco vengono generati utilizzando la libreria SDL, codificati e pacchettizzati nel contenitore MPEG-TS[2] usando la libreria FFmpeg[3] e inviati tramite WebSocket al client.

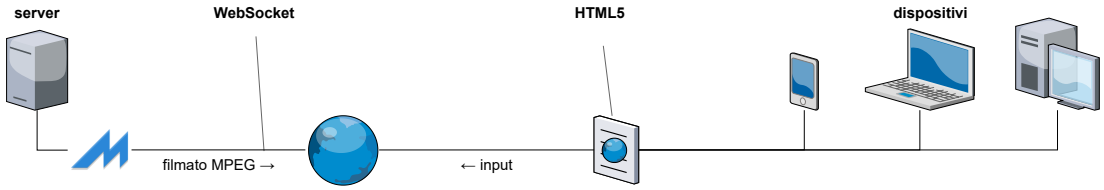

Figura 2.1: Panoramica del sistema

Lato client vari script si occupano di decodificare i dati audio-video ricevuti, catturare e inviare l'input dell'utente (sia dalla tastiera che dal gamepad) al server tramite WebSocket.

Nel prossima sezione introdurremo il software MAME su cui si basa questo progetto.

2.2 MAME

Multiple Arcade Machine Emulator (MAME) è un progetto open source (GNU-GPL) di Nicola Salmoria. La prima versione del MAME risale al febbraio 1997 ed è attualmente supportato da una vasta comunità di sviluppatori in tutto il mondo. Il suo scopo principale è quello di documentare l'hardware utilizzato da ogni gioco emulato, sia per scopi educativi che per scopi di conservazione, al fine di evitare che il software storico scompaia per motivi di obsolescenza. Il progetto MAME è stato realizzato in C e C++, inizialmente usando solamente la libreria standard e poi successivamente, negli anni, sono state aggiunte al progetto varie librerie open source per estenderne le funzionalità. Originariamente era disponibile solo per MS-DOS ma grazie alla vasta comunità di sviluppatori è stato compilato anche per i sistemi Windows e Unix-like [MAME Team 2021]. Logicalmente è suddiviso in quattro macro categorie, come mostrato in Fig. 2.2:

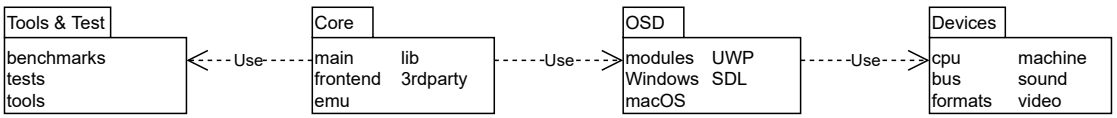

Figura 2.2: MAME, diagramma dei packages

- *Core* in cui ci sono i sotto-progetti indipendenti dal sistema e dal device emulato tra cui il programma principale (progetto main), il front-end grafico, il motore di emulazione (progetto emu), le librerie comuni (lib) ed i sorgenti delle librerie esterne (3rdparty);

[2]MPEG-TS: MPEG transport stream, è un contenitore digitale per la trasmissione e l'archiviazione audio-video.

[3]FFmpeg è una suite open source di librerie e programmi per la gestione di video, audio, e altri file multimediali e stream.

- *OSD* contenente le funzionalità dipendenti dal sistema operativo tra cui macOS, Windows, UWP[4] e SDLMAME, ed i moduli (progetto modules) di input, audio e video dipendenti da altre librerie come OpenGL, DirectX, SDL, CoreAudio, XAudio, XInput, ecc. . . ;
- *Devices* che contiene per ogni device emulato (ad esempio il Capcom CP System III) le classi che gestiscono le informazioni della macchina ed emulano cpu, bus, schede video, schede audio e i dischi (progetto formats);
- *Tools & Test* che contiene varie utility per la gestione delle rom e per la fase di testing e performance.

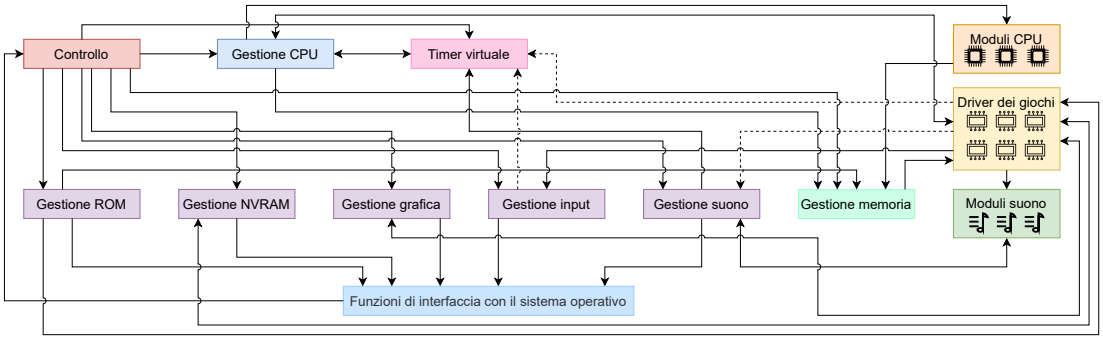

Figura 2.3: Schema della struttura del MAME

Il progetto ha una struttura modulare, schematizzata in Fig. 2.3, formata da un nucleo centrale che dirige le operazioni, gestisce l'interfaccia utente e mette a disposizione dei driver un buon numero di funzioni d'uso comune. Il nucleo delega a moduli esterni l'emulazione dei vari tipi di CPU e di chip audio supportati. In pratica il nucleo fornisce un ambiente operativo specializzato nell'emulazione di videogiochi arcade, che i driver possono sfruttare con poco codice aggiuntivo. Spesso la maggior parte del contenuto di un driver è costituita da strutture dati, gestite direttamente dal nucleo. Il driver deve fornire codice solo per alcuni compiti specifici, ad esempio l'aggiornamento del video [Salmoria 2002].

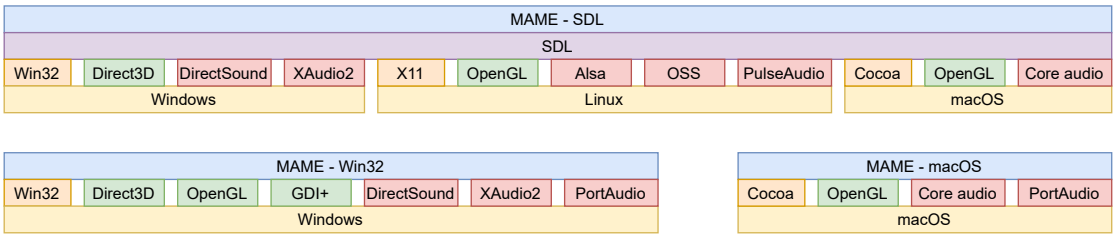

Figura 2.4: Librerie e interfacce utilizzate dal MAME sulle diverse piattaforme

Per quanto riguarda la portabilità, di cui c'è uno schema in Fig. 2.4, ci sono solo tre compilazioni native differenti e sono quella per macOS, Windows ed UWP. In aggiunta c'è la compilazione

[4]UWP: Universal Windows Platform è un'architettura applicativa della Microsoft per sviluppare applicazioni eseguibili su Windows 10, Xbox One e Hololens.

SDLMAME[5] in grado di funzionare su tutti i sistemi operativi supportati dalla libreria SDL. Quest'ultima è la compilazione obbligatoria per i sistemi Linux e per questo motivo è quella che si è scelta di utilizzare per questo server di cloud gaming.

SDL (Simple DirectMedia Layer) è una libreria multipiattaforma che fornisce accesso di basso livello ad audio, tastiera, mouse, gamepad, hardware 3D e framebuffer 2D. Come mostrato in Fig. 2.4 nello schema in alto, SDL è costruito sopra le API di visualizzazione video del sistema operativo (in arancione), le librerie di rendering (in verde) e le librerie che si interfacciano alla scheda audio (in rosso) [SDL Community 2020].

Nel prossimi tre paragrafi verrano descritti i tre moduli su cui sono state apportate le modifiche per trasformare il MAME in una piattaforma di cloud gaming.

2.2.1 Rendering

Nel contesto della computer grafica il rendering identifica il processo di "resa" ovvero di generazione di un'immagine a partire da una descrizione matematica di una scena (che può essere in due o tre dimensioni). La scena viene interpretata da algoritmi che vanno a definire il colore di ogni pixel nell'immagine finale. La descrizione di una scena contiene il punto di vista, le geometrie (sia 2D che 3D), informazioni sull'illuminazione e sulle caratteristiche ottiche delle superfici. Gli elementi di base del rendering 2D sono le textures[6] e le animazioni. Il motore grafico (il software che si occupa anche del rendering) prende gli elementi uno ad uno e li disegna nel framebuffer[7] generando l'immagine finale [Mileff e Dudra 2012]. Come mostrato in Fig. 2.5, questo processo è formato da quattro fasi [Forsyth et al. 2015]:

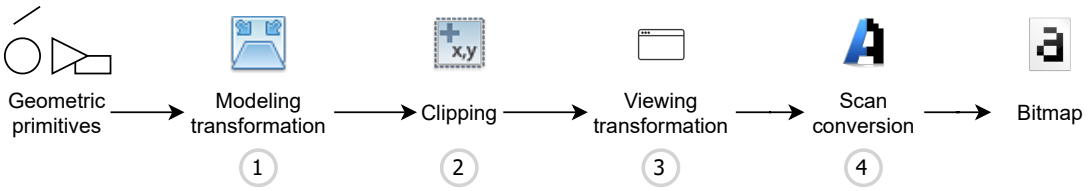

Figura 2.5: Pipeline di rendering 2D

1. trasformazione di modellazione: tramite una trasformazione trasporta le primitive geometriche in un sistema di coordinate universali (*world coordinate system*);
2. clipping: ritaglia porzioni delle primitive al di fuori della finestra di visualizzazione;
3. trasformazione di vista: tramite una trasformazione rigida detta "trasformazione di vista" trasporta le primitive ritagliate dalle coordinate universali a quelle schermo (*screen coordinate*);
4. *scan conversion*: attraverso algoritmi di rasterizzazione[8] genera l'immagine finale.

[5]SDLMAME era un port del MAME che utilizzava solamente la libreria SDL. Nel 2010 è stato incluso ufficialmente nel progetto MAME.

[6]Una texture è un'immagine rappresentata come una matrice bidimensionale di pixel colorati (in inglese bitmap). Ogni pixel è rappresentato tramite una quaterna formata dai tre colori primari più un valore che ne indica la trasparenza (RGB + A); servono 4 bit per memorizzare ogni elemento della quaterna, quindi 32 in totale per un singolo pixel.

[7]Il framebuffer è uno spazio di memoria presente sulla scheda video in cui si memorizza l'immagine che verrà successivamente mostrata a video.

[8]La rasterizzazione è il processo di approssimazione delle primitive geometriche in immagini bitmap.

Il MAME è in grado di emulare giochi sia 2D che 3D (ad es.: Tekken della Namco) ma sia per la volontà di emulare fedelmente l'hardware della macchina sia perché le varie schede ed API grafiche delle macchine emulate non lavorano esattamente come quelle moderne (ad es.: i poligoni della mesh utilizzano i rettangoli al posto dei triangoli) tutta la fase di rendering viene eseguita via software, per questo motivo ciò che viene inviato alla libreria grafica è un insieme di primitive e texture da disegnare sia nel caso di giochi 2D che 3D come spiegato nella FAQ ufficiale nel capitolo sulle performance: *«There are many things that are difficult to emulate without expending a large amount of CPU power. Some specific examples are: Games with 3D graphics. As of this writing, MAME does not pass polygons down to the video card of your system; instead, it renders all 3D graphics by hand in software. Although this code is generally optimized to take advantage of multiple CPUs, it is still quite taxing to do this. Some 3D games give you control of the output resolution; reducing it will reduce the CPU requirements.»* [MAME Team 2018]. La UI del MAME viene renderizzata con le stesse procedure utilizzate per l'emulazione per cui è possibile effettuarne lo streaming ed utilizzarla al posto del front-end HTML.

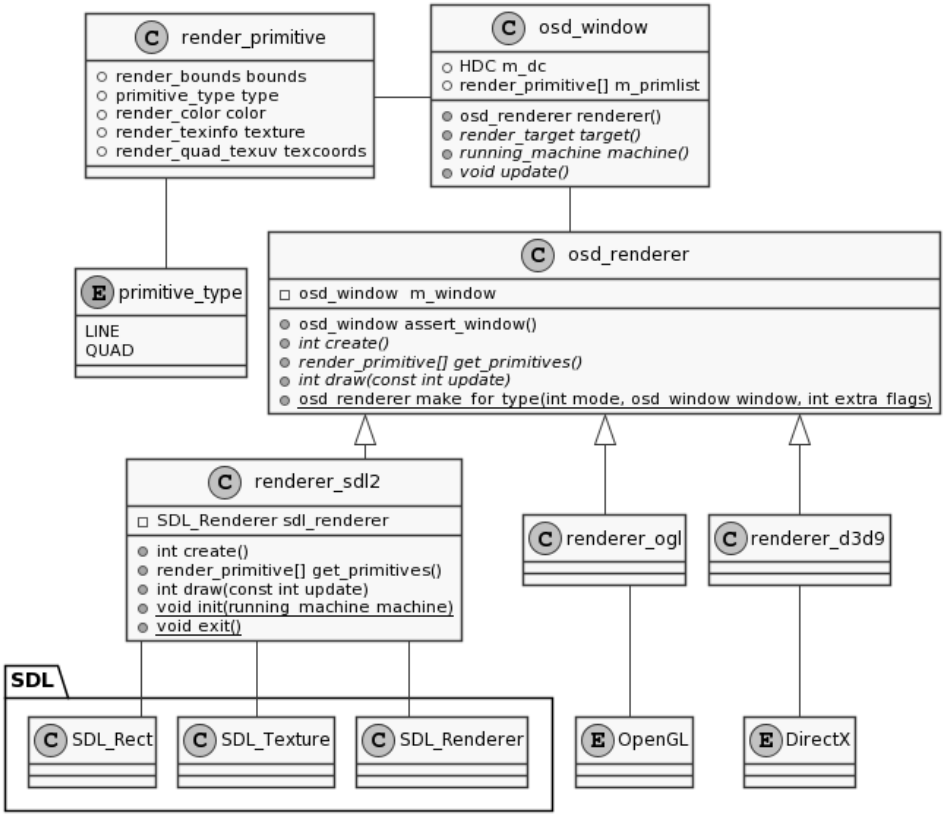

Figura 2.6: Diagramma delle classi relative al rendering

Come detto precedentemente il MAME supporta varie librerie multimediali per la fase di rendering e di missaggio audio. In Fig. 2.6 è visibile un diagramma delle classi relativo alla funzionalità di rendering. Quest'ultimo viene eseguito innanzitutto creando la finestra in cui verrà visualizzato il rendering (classe `osd_window`), tramite la funzione `osd_renderer::make_for_type` viene istanziata una delle classi per il rendering (es.: `render_ogl`, `render_d3d9`, `renderer_sdl2`, ecc. . .) che comunica direttamente con la libreria grafica. Alla creazione della classe per il rende-

ring viene chiamato il metodo `osd_renderer::create` che si occupa di inizializzare il necessario per il processo. Per ogni frame della macchina che viene emulato c'è una fase di disegno tramite il metodo `osd_renderer::draw`.

La classe che si occupa del rendering utilizzando la libreria SDL è `renderer_sdl2` che utilizza le seguenti funzioni SDL:

- `SDL_CreateRenderer`: crea un contesto di rendering 2D per una finestra;
- `SDL_SetRenderDrawColor`: imposta il colore utilizzato per le operazioni di disegno;
- `SDL_RenderFillRect`: riempe un rettangolo con il colore di disegno corrente; è usato per disegnare la primitiva `QUAD`;
- `SDL_RenderDrawLine`: disegna una linea con il colore di disegno corrente; è usato per disegnare la primitiva `LINE`;
- `SDL_RenderPresent`: aggiorna il contesto di rendering con il framebuffer corrente.

Di queste la prima viene utilizzata durante la fase di inizializzazione nella funzione `create`, mentre le altre vengono utilizzate durante la fase di disegno nella funzione `draw`.

2.2.2 Missaggio audio

Il missaggio audio è quel procedimento con cui due o più campioni audio sono missati in un unico output sonoro. Attualmente il MAME emula una cinquantina di chip audio, ma per alcuni giochi degli anni '70 e '80, invece di emulare i circuiti, si sono semplicemente utilizzati i suoni registrati dalla scheda originale poiché il sonoro era prodotto mediante circuiti analogici, la cui emulazione è più complessa. Sono supportate cinque librerie per la gestione della scheda audio: SDL (che è multi piattaforma), DirectAudio (dalla libreria DirectX) e XAudio2 per Windows, Core Audio per macOS, PortAudio per Windows e macOS [Salmoria 2002].

La classe che si occupa del missaggio audio utilizzando la libreria SDL è `sound_sdl2` che utilizza le seguenti funzioni SDL:

- `SDL_OpenAudio`: apre il dispositivo audio;
- `SDL_PauseAudio`: mette in pausa o ripristina la riproduzione audio;
- `SDL_CloseAudio`: interrompere l'elaborazione audio e chiude il dispositivo audio.

La funzione `SDL_OpenAudio` utilizza la struttura `SDL_AudioSpec` che contiene informazioni come il formato del buffer audio, il numero di canali, i bit per canale, ecc..., ed un puntatore ad una funzione di callback asincrona; questa funzione viene utilizzata per riempire il buffer audio con il suono da riprodurre. Come è visibile in Fig. 2.7 la funzione `sdl_callback` ha due parametri, `stream` e `len`, che sono rispettivamente il buffer da riempire con il suono e la dimensione del buffer [Pazera 2003]. Il MAME utilizza un buffer ad anello (classe `ring_buffer`) in cui la macchina emulata aggiunge campioni audio e da cui la funzione `sdl_callback` li rimuove e li copia nel vettore `stream` che viene poi inviato da SDL alla scheda audio.

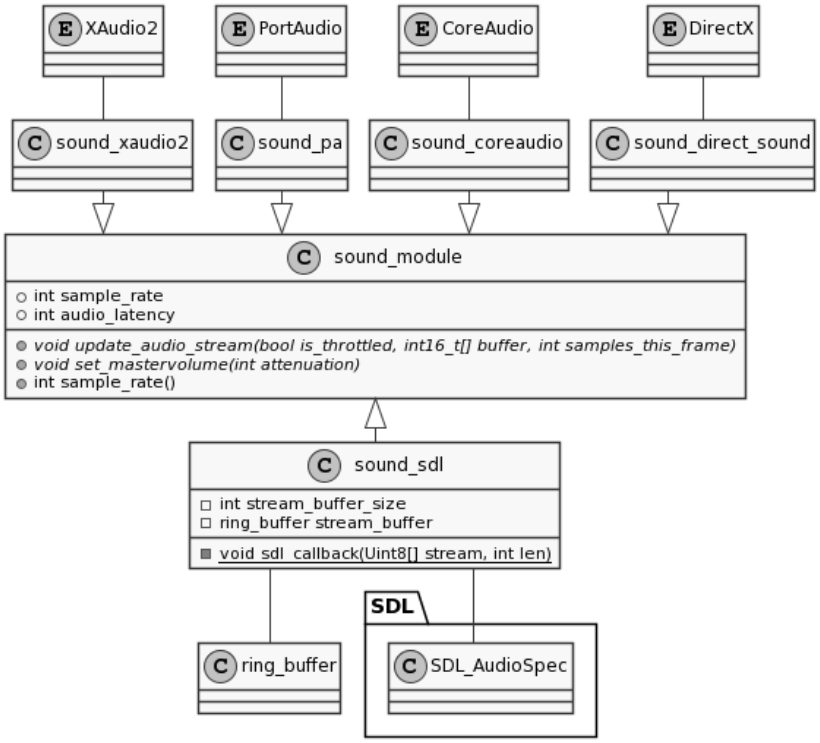

Figura 2.7: Diagramma delle classi relative al missaggio audio

2.2.3 Gestione input

Il MAME nasce come emulatore di videogiochi arcade e quindi ha bisogno di gestire l'input della gettoniera, dei controlli per il personale di servizio[9] e dei comandi del giocatore. Il dispositivo più usato è il joystick, tuttavia alcuni apparecchi montano volante e pedali, spinner, trackball, pistole e fucili ottici, ecc... [Salmoria 2002].

La libreria SDL gestisce la tastiera, il mouse e i joystick (pistola ottica, volante e pedali rientrano in questa categoria) tramite un sistema ad eventi:

- per la tastiera due tipi di eventi: pressione e rilascio dei tasti;
- per il mouse tre tipi di eventi: movimento del mouse, pressione e rilascio dei tasti;
- per il joystick tra i tre e i cinque tipi di eventi: pressione e rilascio dei tasti, movimento della leva di comando, movimento della levetta[10] e movimento della trackball.

Come mostrato in Fig. 2.8 il sistema di gestione degli eventi di SDL offre tre metodi diversi per l'acquisizione [Pazera 2003]:

- attesa: questo è il metodo utilizzato per la gestione dell'input nei software desktop, in cui il programma è in attesa dell'input utente per poi eseguire un'azione;

[9] I controlli per il personale di servizio sono quegli interruttori presenti su apparecchi non recenti che servono per testare il funzionamento dell'apparecchio, regolare il livello di difficoltà, ecc...; sono stati poi sostituiti dai menu interattivi.

[10] In inglese chiamato "hat switch" oppure "POV switch".

- polling: è il metodo solitamente utilizzato nei videogiochi. Solitamente il ciclo di esecuzione di un videogioco è: processare l'input, aggiornare lo stato, eseguire il rendering e il missaggio audio;
- diretto: questo metodo da la possibilità di leggere lo stato dei devices in qualsiasi momento in modo asincrono. Lo svantaggio di questo metodo è che bisogna comunque eseguire il polling dalla coda degli eventi.

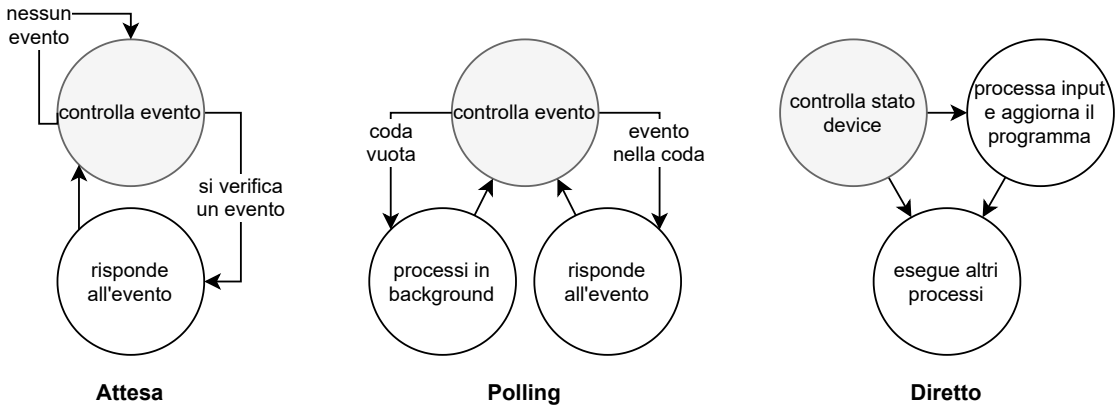

Figura 2.8: I tre metodi del gestore eventi della libreria SDL

Il sistema di gestione dell'input del MAME utilizza il metodo di polling attraverso le funzioni della libreria: `SDL_PollEvent` e `SDL_PumpEvents`; la prima esegue il controllo per gli eventi attualmente in sospeso e la seconda aggiorna la coda degli eventi e lo stato del dispositivo di input. Il sistema, schematizzato in Fig. 2.9, è formato dai device e dai moduli. La classe `input_module_base` che implementa un modulo di input dà la possibilità di sottoscrivere un device al sistema ad eventi e fornisce l'interfaccia per mapparlo con i comandi emulati dal MAME (ad es.: nel caso di SDL si mappa `SDLK_SPACE` con `P1_BUTTON1`). Il device, implementato tramite la classe `device_info`, offre le funzioni per gestire la coda dell'input e di impostarne lo stato (ad es.: tasto X premuto, tasto Y rilasciato).

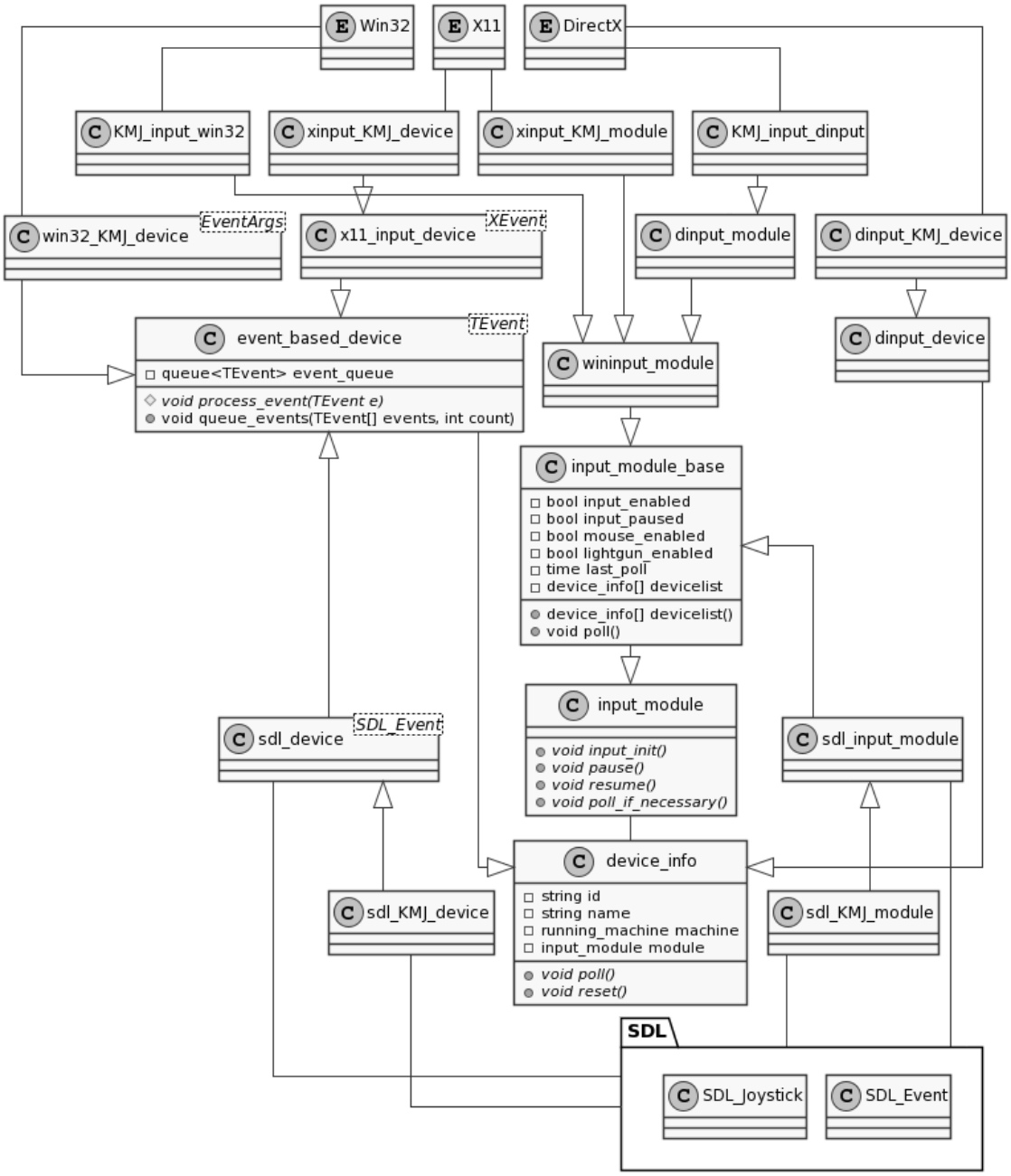

Figura 2.9: Diagramma delle classi relative ai moduli di input. Per ridurre la dimensione del diagramma sono state omesse le singole classi per gestire tastiera, mouse e joystick e sono state raggruppate con la sigla KMJ (Keyboard, Mouse, Joystick)

Capitolo 3

Implementazione

In questo capitolo verranno descritte le cinque fasi aggiuntive del cloud gaming e la loro implementazione in C++ come nuovi moduli del MAME: la cattura audio-video, la codifica, la trasmissione, la decodifica e la cattura dell'input utente. Il processo completo è mostrato in Fig. 3.1.

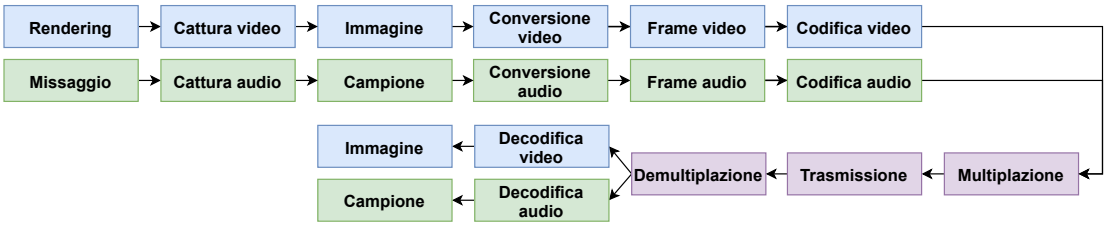

Figura 3.1: Processo completo delle fasi del cloud gaming: dalla cattura audiovisiva alla visualizzazione

3.1 Cattura audio-video

Nel paradigma del cloud gaming, per cattura si intende la fase in cui l'output audiovisivo del videogioco non viene visualizzato (l'immagine) ed emesso (il suono) ma viene inviato ad un codificatore. Bisogna distinguere tra due casi differenti: effettuare la cattura audio-video su giochi o emulatori già compilati oppure da compilare. Nei prossimi due paragrafi vedremo i metodi di cattura per la parte video e per quella audio.

3.1.1 Cattura video

Per la cattura video, nei giochi già compilati, esistono tre metodi di cattura: l'hook di funzioni della libreria multimediale, l'utilizzo delle API per il framebuffer virtuale e l'uso di schede video con funzionalità di codifica incorporata.

L'hook di funzione è una tecnica di programmazione con la quale si sostituisce il comportamento di una funzione di un programma o di una libreria già compilata. Questa operazione su una libreria multimediale consiste nel modificare il normale funzionamento della procedura che invia il framebuffer della scheda video all'uscita. Ad esempio per le librerie multimediali più usate le funzioni da catturare sono: `SDL_RenderPresent` per SDL, `Present` per DirectX e `SwapBuffer`

per OpenGL [Huang et al. 2014]. Questo metodo è stato usato ad esempio da OnLive, Gaikai e Vortex. Un esempio di codice per l'hook di funzioni per SDL è mostrato nel Lis. 3.1.

```
HOOK_TRACE_INFO hHook = { NULL };
HMODULE lib_handle = LoadLibrary("SDL2.dll");

FARPROC old_SDL_RenderPresent = GetProcAddress(
    lib_handle, "SDL_RenderPresent");

NTSTATUS result = LhInstallHook(
    old_SDL_RenderPresent, my_SDL_RenderPresent,
    NULL, &hHook);
```

Listato 3.1: Codice di esempio per hook della libreria SDL

Il framebuffer virtuale è una funzionalità offerta dalle API di sistema per accedere all'attuale framebuffer presente nella scheda video. Il loro utilizzo principale è per il remote desktop, ma alcune piattaforme come "GamingAnywhere" [Huang et al. 2014] e "Parsec"[1] [Dickson 2016] utilizzano questa funzionalità per la cattura video, il primo per la piattaforma di cloud gaming, il secondo per offrire un computer virtuale. Alcuni esempi nei moderni sistemi operativi sono: "X virtual framebuffer" del gestore grafico X11 su Linux [Wiggins 2012], "Desktop Duplication API" su Windows [Microsoft 2021b] e `CGDisplayStream` della libreria "Quartz Display Services" su macOS.

Le schede video con codifica incorporata non hanno una fase di cattura poiché il chip di codifica legge i dati direttamente dal framebuffer; l'argomento è trattato più avanti nel paragrafo 3.2.

Nel caso di giochi o emulatori da compilare si interviene direttamente nel codice sulla parte relativa alla chiamata della routine di rendering, come è stato fatto in questo progetto in cui ho scelto di intervenire sui sorgenti del MAME. Come mostrato in Fig. 3.2, sono state aggiunte alla classe `renderer_sdl2` le due funzioni (`init_streaming_render` e `free_streaming_render`) per allocare e deallocare le strutture dati per la cattura. Come mostrato nel Lis. 3.2 le funzioni della libreria SDL che sono state aggiunte al codice sono:

- `CreateRGBSurfaceWithFormat`: crea una superficie RGB specificando il formato pixel da utilizzare;
- `CreateSoftwareRenderer`: crea un contesto di rendering 2D per una superficie;
- `RWFromMem`: prepara un buffer di memoria di lettura-scrittura da utilizzare con la struttura dati `RWops` (read-write opaque pointer structure).

```
buffer_bytes_length = w * h * 4; // 4 canali (RGBA)
buffer_bytes = new char[buffer_bytes_length];

surface = SDL_CreateRGBSurfaceWithFormat(
    0, w, h, 32, SDL_PIXELFORMAT_RGBA32);

renderer = SDL_CreateSoftwareRenderer(surface);
buffer = SDL_RWFromMem(buffer_bytes, buffer_bytes_length);
```

Listato 3.2: Codice aggiunto per la cattura video: inizializzazione. File: draw13.cpp

[1] Parsec è un servizio di desktop remoto basato su Windows offerto dalla omonima società.

Nel Lis. 3.3 è mostrato il codice che è stato inserito per inviare alla classe statica che gestisce lo streaming (`streaming_server`) il frame da inviare al giocatore (che verrà codificato prima dell'invio). Poiché l'attuale contesto di rendering (`renderer`) è creato su una superfice RGB (`surface`) quando il contesto di rendering viene aggiornato con il framebuffer corrente (tramite `SDL_RenderPresent`) è la superfice a ricevere il frame attuale.

```
1 SDL_RenderPresent(renderer);
2
3 streaming_server::instance()
4     .send_video_frame(surface->pixels);
5
6 SDL_RWseek(buffer, 0, RW_SEEK_SET); // posizione inizio buffer
```

Listato 3.3: Codice aggiunto per la cattura video: disegno. File: draw13.cpp

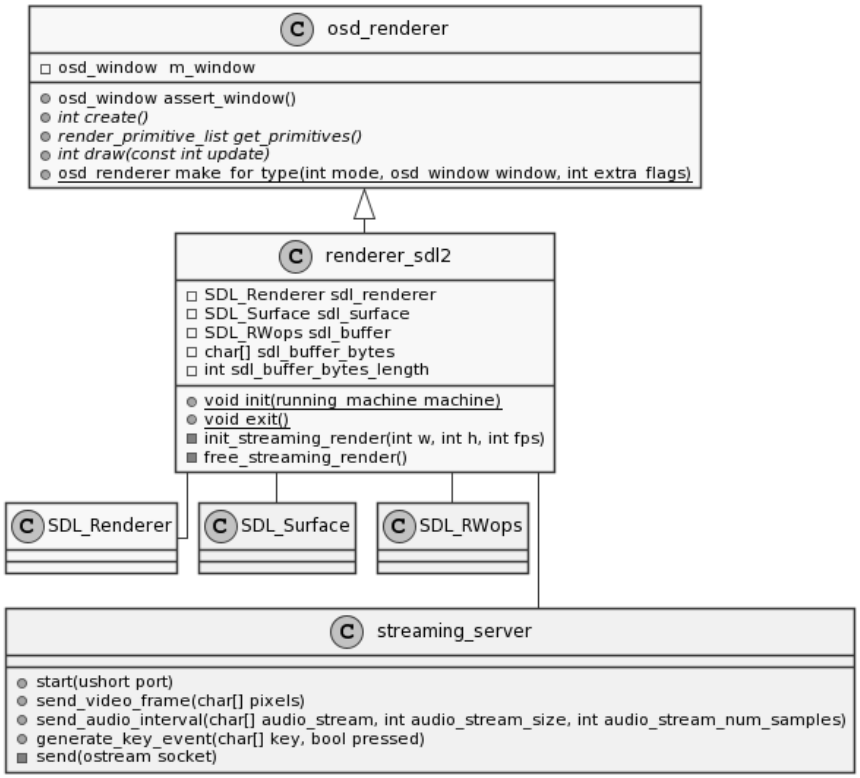

Figura 3.2: Diagramma delle classi relative alla cattura video

Dei quattro metodi appena visti, l'hook di funzione e la modifica dei sorgenti sono equipollenti poiché vanno ambedue a sostituire il comportamento software nella fase di rendering. Il framebuffer virtuale è implementato in maniera differente sui vari sistemi operativi e, su alcune piattaforme, il sistema operativo visualizza comunque il rendering nella finestra, rendendolo in alcuni casi equipollente o inferiore a l'hook di funzione e alla modifica dei sorgenti. La codifica incorporata è il metodo migliore perché, come vedremo nel paragrafo 3.2, dopo aver eseguito il rendering, converte il frame nel formato colore adatto e lo codifica.

3.1.2 Cattura audio

Come per la cattura video si può ricorrere all'hook di funzioni della libreria multimediale o alle API del sistema operativo per la gestione audio [Huang et al. 2014]; ad esempio: "Windows Audio Session API" per Windows [Microsoft 2021a], "Advanced Linux Sound Architecture" per Linux [ALSA team 2021] e "Core Audio" per macOS [Apple 2021].

Come nel caso della cattura video ho scelto di modificare il sorgente del programma. Come mostrato in Fig. 3.3 la classe `sound_sdl` implementa la funzione di callback necessaria per utilizzare il missaggio audio di SDL, di cui abbiamo parlato nel paragrafo 2.2.2. All'interno di questa funzione, tramite la funzione `send_audio_interval`, è stato inviato alla classe che si occupa di gestire il server di streaming (`streaming_server`) il buffer sonoro (`stream`) e la sua dimensione (`len`). La modifica apportata è mostrata nel Lis. 3.4.

```
1 // 512 campioni audio (per canale)
2 streaming_server::instance()
3     .send_audio_interval(stream, len, 512);
4
5 memset(stream, 0, len); // svuota buffer sonoro
```

Listato 3.4: Codice aggiunto per la cattura audio. File: sdl_sound.cpp

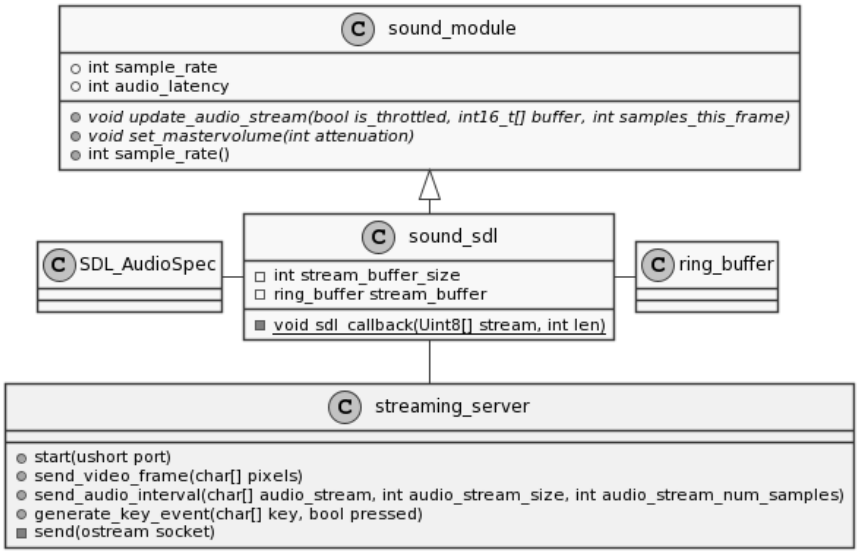

Figura 3.3: Diagramma delle classi relative alla cattura audio

Nel caso della cattura audio i due metodi risultano equipollenti.

3.2 Codifica

Nel paradigma dello streaming la codifica è un processo essenziale sia perché stabilisce il formato audiovisivo che verrà trasmesso e poi decodificato sia perché riduce l'elevata quantità di dati che compongono il flusso audiovisivo. La codifica avviene tramite un codificatore (hardware o software) che descrive e comprime i dati (solitamente con perdita dell'informazione). Di seguito

parleremo dei codificatori video e audio e di come è stata utilizzata la libreria *FFmpeg* per la codifica.

3.2.1 Codificatori video

I codificatori video si divino in due gruppi:

- intraframe: codificano ogni fotogramma indipendentemente dagli altri;
- interframe: descrivono i cambiamenti tra fotogrammi contigui; sono più efficienti su scene statiche poiché i cambiamenti da descrivere sono minori.

Gli attuali monitor (CRT, LCD, PDP e OLED) utilizzano RGB come modello di colori (e di conseguenza gli attuali sistemi operativi), ma gli algoritmi di codifica video utilizzano il modello YCbCr. Sfruttando l'innata sensibilità alla luminosità del sistema visivo umano questo modello suddivide i dati del colore in componenti separati di luminanza e crominanza, come mostrato in Fig. 3.4, possiamo comprimere selettivamente solo i componenti di crominanza per ottenere risparmi di spazio con una perdita minima di qualità.

Utilizzando le costanti dello standard NTSC e PAL: K_R, K_G e K_B che valgono rispettivamente $0,299$, $0,587$ e $0,114$ [Rupp 2008], la conversione da YCbCr a RGB avviene utilizzando l'Eq. 3.1:

$$\begin{aligned} Y &= K_R R + K_G G + K_B B, \\ C_B &= \frac{0,5}{1 - K_B}(B - Y), \\ C_R &= \frac{0,5}{1 - K_R}(R - Y). \end{aligned} \tag{3.1}$$

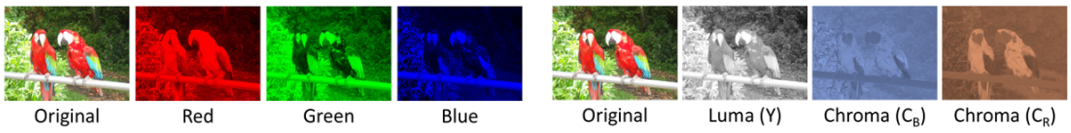

Figura 3.4: Componenti di RGB e YCbCr

Come mostrato in Fig. 3.5, nel modello RGB i dati per un singolo componente di colore sono intercalati tra i pixel e ciascun pixel è archiviato in memoria in modo contiguo, mentre nel modello YCbCr le componenti Cb e Cr sono intercalate e memorizzate insieme, mentre la componente Y rimane nel proprio piano, per un totale di due piani.

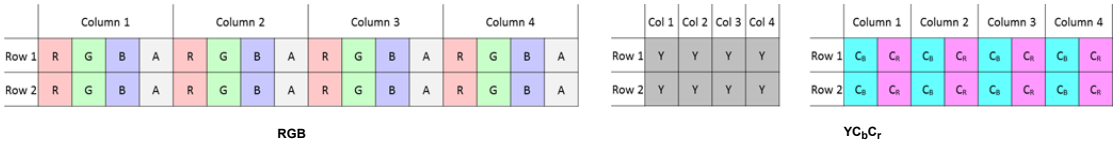

Figura 3.5: Matrice di memorizzazione di RGB e di YCbCr

Come detto precedentemente la codifica può essere sia software che hardware, per quanto riguarda la codifica software nel paragrafo 3.2.3 parleremo della libreria FFmpeg che supporta

la codifica e decodifica di tutti i codec presenti in Tab. 3.2 e Tab. 3.3, invece per la codifica hardware è importante sottolineare quali soluzioni hanno utilizzato le varie piattaforme di cloud gaming.

La prima società che modificò una scheda video adattandola per il cloud gaming fu Sony che incluse un chip di codifica sulle GPU Nvidia ed AMD (rispettivamente per il rendering PS3 e PS4) [Whitwam 2014]. Altre piattaforme di cloud gaming che usano schede video con funzionalità di encoding incluse sono "Nvidia GeForce Now" su schede Nvidia [Nvidia 2021], "Microsoft XBox Cloud Gaming" su schede AMD [Warren 2020b], "Google Stadia" su AMD [Google 2019b] e "Amazon Luna" su Nvidia [Warren 2020a]. La lista delle aziende che creano schede video con questa funzionalità è riportata in Tab. 3.1 [AMD 2021] [Intel 2020] [Nvidia 2020] [Qualcomm 2020].

Produttore	Funzionalità	Codec supportati
AMD	Video Core Next	AVC, HEVC
Intel	Quick Sync Video	AVC, HEVC, MPEG-2, VP8, VP9
Nvidia	NVENC	AVC, HEVC
Qualcomm	Hexagon	AVC, HEVC, H.263, MPEG-4, VP8, VP9

Tabella 3.1: Schede video con funzionalità di codifica video

Una varietà di codec audio e video sono stati implementati fino ad oggi con licenza proprietaria o libera e caratteristiche diverse. Una lista dei codec video più utilizzati[2] è mostrata in Tab. 3.2 [Mozilla 2021d].

Codec	Contenitore	Perdita dei dati	Max bit-rate[3]	Licenza[4]
AV1	MP4, WebM	sì	800	libera
AVC	MP4	sì/no	arbitrario	proprietaria
HEVC	MP4	sì	800	proprietaria
MPEG-1	MPEG	sì	1,5	scaduta
MPEG-2	MP4, MPEG	sì	100	scaduta
Theora	Ogg	sì	2000	libera
VP8	Ogg, WebM	sì	arbitrario	libera
VP9	Ogg, WebM	sì	arbitrario	libera

Tabella 3.2: Lista codec video

3.2.2 Codificatori audio

Come per i codificatori video anche i codificatori audio sono di tipo hardware o software. I codificatori hardware sono integrati nelle schede audio e sono in grado di trasformare il segnale da analogico a digitale e viceversa. Il segnale è rappresentato digitalmente come modulazione a impulsi codificati (PCM) e non è compresso.

Come mostrato in Fig. 3.6, la linea arancione rappresenta i campioni prelevati (in questo esempio viene utilizzato il punto medio) dalla forma d'onda audio. Alla riproduzione queste

[2] I codec video più utilizzati sono di categoria interframe.
[3] In Mbps.
[4] Alla scadenza dei brevetti il software può essere utilizzato liberamente.

ampiezze vengono utilizzate per generare un'approssimazione della forma d'onda originale. Il numero di campioni prelevati al secondo è chiamato frequenza di campionamento.

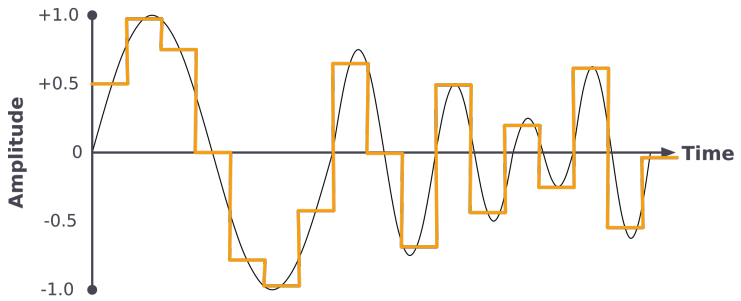

Figura 3.6: Conversione audio da analogico a digitale

Esistono diversi formati per rappresentare i campioni all'interno di un file audio: naturali a 8 bit (uchar), interi a 16 bit (short), interi a 32 bit (int), reali a 32 bit (float) e reali a 64 bit (double); sia *planar*[5] che *packed*[6].

La compressione audio avviene solo tramite codificatori software. Anche questi hanno tre formati di codifica: con perdita, senza perdita e nessuna compressione. Il formato di codifica audio con perdita (l'unico usato per lo streaming) riduce la risoluzione in bit del suono e rimuove le frequenze che l'essere umano non può sentire o sente poco (secondo un modello psicoacustico). Altri fattori che influenzano la dimensione dell'audio codificato sono il numero di canali (che influisce solo sulla percezione della direzionalità e non sulla qualità) e il numero di campioni disponibili al secondo (che influiscono sulla fedeltà audio codificata) [Spanias 2008].

Una lista dei codec audio più utilizzati è mostrata in Tab. 3.3 [Mozilla 2021c].

Codec	Contenitore	Perdita dei dati	Max bit-rate[7]	Licenza[8]
AAC	ADTS, MP4	sì	512	proprietaria
ALAC	MP4	no	variabile	proprietaria
FLAC	FLAC, MP4, Ogg	no	variabile	libera
MP2	MPEG	sì	320	scaduta
MP3	ADTS, MP4, MPEG	sì	320	scaduta
Opus	MP4, Ogg, WebM	sì	510	libera
PCM	WAV	no	64	scaduta
Vorbis	Ogg, WebM	sì	500	libera

Tabella 3.3: Lista codec audio

3.2.3 La codifica tramite le librerie di FFmpeg

Il progetto FFmpeg è un insieme di librerie (per il linguaggio C) e programmi open source per la gestione di video, audio, file e flussi. Include i codec per la codifica e la decodifica della maggior parte dei formati di file audio e video conosciuti [FFmpeg Team 2021]. Le librerie sono:

[5] *Planar*: ogni canale audio si trova su un piano dati separato. Tutti i piani dati hanno la stessa dimensione.
[6] *Packed*: viene utilizzato un unico piano dati e i campioni per ciascun canale vengono interlacciati.
[7] In kbps.
[8] Alla scadenza dei brevetti il software può essere utilizzato liberamente.

- `SW Resample` offre funzioni per il ricampionamento audio;
- `SW Scale` offre funzioni per il ridimensionamento dell'immagine video e le routine di conversione dello spazio colore e del formato pixel;
- `AV Codec` contiene tutti i codificatori e decodificatori audio/video FFmpeg nativi;
- `AV Format` contiene demuxer e muxer per formati di contenitori audio e video;
- `AV Util` è una libreria di supporto contenente routine comuni a diverse parti di FFmpeg, come funzioni hash, cifratura, decompressore LZO e codificatore/decodificatore Base64;
- `AV Filter` permette di modificare o esaminare il video/audio tra il decoder e l'encoder;
- `Post Proc` è una libreria contenente le vecchie routine di post-elaborazione video basate su H.263.

La classe che si occupa della codifica è `encode_to_movie` che utilizza le seguenti librerie FFmpeg: `SW Scale` per la conversione dal modello RGB a YCbCr, `SW Resample` per il ricampionamento audio, `AV Codec` per la codifica audio e video e `AV Format` per l'incapsulamento nel contenitore. `encode_to_movie` è stata progettata per codificare in vari formati, come mostrato in Tab. 3.4, anche se per motivi di decodifica si è scelto MPEG-TS (di cui parleremo nel paragrafo 3.6).

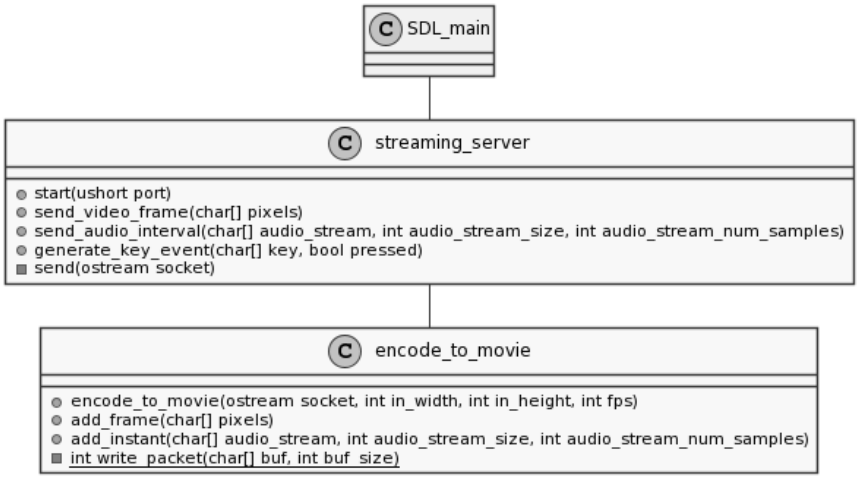

Figura 3.7: Diagramma delle classi relativo alla codifica

Contenitore	Codec video	Codec audio	Modello colore	Modello audio
MP4	AVC	AAC	YCbCr	float, planare
MPEG-TS	MPEG-1	MP2	YCbCr	short
WebM	VP9	Vorbis	YCbCr	float, planare

Tabella 3.4: Lista dei codificatori utilizzabili

Come mostrato in Fig. 3.7 il costruttore di `encode_to_movie` richiede il parametro `socket` su cui verrà inviato il filmato al client. Tramite le funzioni `add_frame` e `add_instant` vengono aggiunti il frame video e quello audio per essere codificati.

Al termine di ogni fase di incapsulamento la libreria `AV Format` invia alla funzione di callback `encode_to_movie::write_packet` il pacchetto (`buf` di lunghezza `buf_size`) appena codificato, che viene scritto sulla `socket`.

Codifica video con FFmpeg

Con riferimento al List. 3.5, la codifica video inizia trasformando il frame video da SDL (`uint8_t[] pixels`) al frame video FFmpeg (`AVFrame`) tramite la funzione `avpicture_fill`. Come detto precedentemente i codificatori video utilizzano il modello di colore YCbCr, la conversione da RGB avviene tramite la funzione `sws_scale`; al termine di questa operazione `yuv_frame` conterrà il frame convertito in YCbCr. Tramite le funzioni `avcodec_send_frame` e `avcodec_receive_packet` si comunica alla libreria FFmpeg di voler codificare il frame YCbCr. Tramite la funzione `av_interleaved_write_frame` avviene la multiplazione del pacchetto video nel formato contenitore.

```
1  void add_frame(uint8_t[] pixels)
2  {
3      // Riempimento AVPicture con frame generato da SDL
4      avpicture_fill(
5          rgb_frame,                          // output
6          pixels,                             // input
7          AV_PIX_FMT_BGR32,                   // formato input
8          width, height);                     // dimensione input
9
10     // conversione da RGB a YUV
11     sws_scale(
12         video_converter_context,            // contesto conversione
13         rgb_frame->data,                    // input
14         rgb_frame->linesize, 0, height,     // dimesione input
15         yuv_frame->data,                    // output
16         yuv_frame->linesize);               // dimensione output
17
18     av_init_packet(&video_packet);          // pacchetto video
19
20     // inizio codifica
21     avcodec_send_frame(
22         video_codec_context,                // contesto codifica
23         yuv_frame);                         // frame YCbCr
24
25     // fine codifica
26     avcodec_receive_packet(
27         video_codec_context,                // contesto codifica
28         &video_packet);                     // pacchetto video
29
30     // multiplazione pacchetto video
31     av_interleaved_write_frame(
32         muxer_context,                      // contesto multiplazione
33         &video_packet);                     // pacchetto video
34 }
```

Listato 3.5: Codice per la codifica video. File: encode_to_movie.hpp

Codifica audio con FFmpeg

Con riferimento al List. 3.6, la codifica audio inizia trasformando il frame audio da SDL (`uint8_t[] stream`) al frame audio FFmpeg tramite la funzione `avcodec_fill_audio_frame`. Tramite la funzione `swr_convert` si converte dal formato audio SDL (PCM rappresentato tramite interi a 16 bit con segno per ogni canale) al formato audio di destinazione (come riportato in Tab. 3.4). La funzione `avcodec_fill_audio_frame` viene utilizzata nuovamente per riempire il frame audio FFmpeg con i dati appena convertiti (`convertedData`). Tramite la funzione `avcodec_encode_audio2` si comunica alla libreria FFmpeg di voler codificare il frame audio. Tramite la funzione `av_interleaved_write_frame` avviene la multiplazione del pacchetto audio nel formato contenitore.

Tramite la funzione `avformat_write_header`, che scrive l'intestazione per il flusso e alloca i dati del contesto, ha inizio la multiplazione del pacchetto audiovisivo. Questa termina chiamando la funzione `av_write_trailer` che scrive il trailer per il flusso, genera il pacchetto e libera i dati del contesto; questa funzione chiama internamente la callback `write_packet` di cui abbiamo parlato precedentemente, che andrà a scrivere sulla nostra `socket` il pacchetto.

```
1  void add_instant(uint8_t[] stream, int stream_size, int num_samples)
2  {
3      wav_frame->nb_samples = num_samples;
4      // Riempimento frame con audio da SDL
5      avcodec_fill_audio_frame(
6          wav_frame,                          // output
7          2,                                  // canali input
8          AV_SAMPLE_FMT_S16,                  // formato input
9          stream,                             // input
10         stream_size);                       // dimensione input
11
12     // conversione audio
13     swr_convert(
14         audio_converter_context,            // contesto audio
15         &convertedData,                     // output
16         output_frame->nb_samples,           // campioni output
17         wav_frame->data,                    // input
18         wav_frame->nb_samples);             // campioni input
19
20     // riempimento frame destinazione
21     avcodec_fill_audio_frame(
22         output_frame,                       // output
23         2,                                  // canali output
24         audio_codec_context->sample_fmt,    // formato output
25         convertedData,                      // input
26         buffer_size);                       // dimensione input
27
28     av_init_packet(&audio_packet);          // pacchetto audio
29     // codifica
30     avcodec_encode_audio2(
31         audio_codec_context,                // contesto codifica
32         &audio_packet,                      // pacchetto audio
33         output_frame_frame);                // output
```

```
34      // multiplazione pacchetto audio
35      av_interleaved_write_frame(
36          muxer_context,                  // contesto multiplazione
37          &audio_packet);                 // pacchetto audio
38  }
```

Listato 3.6: Codice per la codifica audio. File: encode_to_movie.hpp

3.3 Trasmissione

La trasmissione è la fase del cloud gaming che influisce maggiormente sulla latenza e l'unica ad avere un tempo d'esecuzione condizionato dalla qualità della connessione dell'utente. Nel paragrafo 1.2.2 abbiamo parlato delle piattaforme che hanno fatto parte del panorama del cloud gaming, come mostrato in Tab. 1.2 i protocolli di trasmissione più usati sono stati UDP e RTP. Come detto nel paragrafo 2.1 in questo progetto si è scelto di utilizzare come protocollo di comunicazione WebSocket.

3.3.1 WebSocket

WebSocket è un insieme di standard formato dalle API WebSocket implementate dai browser web e dal protocollo di comunicazione WebSocket. Il protocollo di comunicazione fornisce canali di comunicazione full-duplex su una singola connessione TCP. È supportato nativamente da tutti i browser e il suo utilizzo è simile alle normali socket. Per questi motivi è il protocollo di comunicazione generico più utilizzato sul web.

WebSocket: le API

Come mostrato nel Lis. 3.7 tratto da [Grigorik 2013] le API WebSocket fornite dal browser sono relativamente semplici. La connessione viene creata istanziando la classe `WebSocket`, al costruttore oltre al URL di una risorsa WebSocket è possibile passare un vettore di stringhe che specificano dei sotto-protocolli definiti dall'utente. La classe `WebSocket` espone 4 callback: errore, chiusura, apertura e ricezione messaggio. Tramite la proprietà `binaryType` si può controllare il tipo di dati binari utilizzati per la ricezione (di tipo `Blob` o `ArrayBuffer`). La classe `WebSocket` espone solo due metodi: `close` per chiudere la connessione e il metodo asincrono `send` per inviare dati (di tipo `String`, `Blob`, `ArrayBuffer` o `ArrayBufferView`) [Mozilla 2021f].

```
1  var ws = new WebSocket(
2      'ws://www.maionemiky.it/mamecgp',
3      ['protocollo_utente1', 'protocollo_utente2']);
4
5  ws.binaryType = 'arraybuffer'; // or 'blob';
6
7  ws.onerror = function (error) { ... }
8
9  ws.onclose = function () { ... }
10
11 ws.onopen = function () {
12     ws.send('Connection established. Hello server!');
13 }
14
```

```
15 ws.onmessage = function(msg) {
16     if(msg.data instanceof ArrayBuffer)
17         processArrayBuffer(msg.data);
18     else if(msg.data instanceof Blob)
19         processBlob(msg.data);
20     else
21         processText(msg.data);
22 }
```

Listato 3.7: Esempio di codice delle API WebSocket

WebSocket: il protocollo

Il protocollo WebSocket è composto da due componenti: l'handshake utilizzato per negoziare i parametri della connessione, ed un meccanismo di framing dei messaggi binari per consentire un basso sovraccarico [Grigorik 2013].

Per stabilire la connessione il client invia una richiesta di handshake prestabilita (come mostrato in Lis. 3.8) a cui il server risponde (come mostrato in Lis. 3.9). All'interno della richiesta vengono specificati opportuni campi. Il client invia un guid casuale codificato in base64 nel campo `Sec-WebSocket-Key` a cui il server risponde restituendo lo stesso guid con l'aggiunta di una costante[9] codificando il tutto con SHA-1 prima e base64 poi. Al termine di questa procedura la comunicazione full-duplex può iniziare [Mozilla 2021g]. L'implementazione server all'interno del progetto è mostrata nel Lis. 3.10 relativo alla classe `SocketServerBase`.

```
1 GET /mamecgp HTTP/1.1
2 Host: www.maionemiky.it
3 Upgrade: websocket
4 Connection: Upgrade
5 Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
6 Sec-WebSocket-Protocol: cloudgaming
7 Sec-WebSocket-Version: 13
8 Origin: http://www.maionemiky.it
```

Listato 3.8: Richiesta handshake da parte del client

```
1 HTTP/1.1 101 Switching Protocols
2 Upgrade: websocket
3 Connection: Upgrade
4 Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
5 Sec-WebSocket-Protocol: cloudgaming
```

Listato 3.9: Risposta handshake da parte del server

```
1 bool generate_handshake(Connection &c, ostream &handshake)
2 {
3     auto ws_magic_string =
4         "258EAFA5-E914-47DA-95CA-C5AB0DC85B11";
5
6     auto header_it =
7         connection->header.find("Sec-WebSocket-Key");
```

[9] La stringa magica utilizzata è definita in RFC 6455.

```
8       auto sha1 =
9           sha1_encode(header_it->second + ws_magic_string);
10
11      handshake
12          << "HTTP/1.1 101 Web Socket Protocol Handshake" << endl
13          << "Upgrade: websocket" << endl
14          << "Connection: Upgrade" << endl
15          << "Sec-WebSocket-Accept: "
16          << base64_encode(sha1) << endl << endl;
17
18      return true;
19  }
```

Listato 3.10: Handshake implementato in SocketServerBase. File: server_ws_impl.hpp

Come detto precedentemente WebSocket implementa un meccanismo di framing dei messaggi binari che consente la comunicazione orientata ai messaggi. Quando viene inviato un messaggio il destinatario riceve una notifica quando l'intero messaggio è stato ricevuto. Questo meccanismo divide ogni messaggio in più frame, mostrato in Fig. 3.8, che vengono trasportati, riassemblati e poi viene inviata una notifica al destinatario [Grigorik 2013].

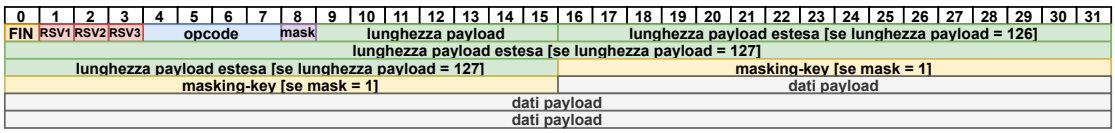

Figura 3.8: Frame del WebSocket

Le parti più importanti che compongono un frame sono:

- *FIN* che indica se è l'ultimo frame;
- *Opcode* che indica il tipo di messaggio: testo (1), binario (2), chiusura connessione (8), ping (9) e pong (10);
- *mask* indica se il *payload* è mascherato (i messaggi dal client al server hanno questo bit a 1);
- *lunghezza payload* indica quanto è lungo il *payload*: 7 bit, $7 + 16$ bit o $7 + 64$ bit;
- *masking-key* contiene un valore a 32 bit usato per masherare il *payload*.

Con riferimento alla Fig. 3.9, la classe che si occupa di inizializzare il server e comunicare con la classe per la codifica è `streaming_server` la quale utilizza la classe `ws_server` per comunicare tramite il protocollo WebSocket con la pagina HTML.

La classe `ws_server` implementa la classe generica `SocketServer`, tipizzandola come socket TCP della libreria Asio[10].

`SocketServer`, che eredita da `SocketServerBase`, implementa solo il metodo `accept` che serve per accettare una nuova connessione, come mostrato nel Lis. 3.11.

[10] Asio è una libreria open source per la programmazione di rete.

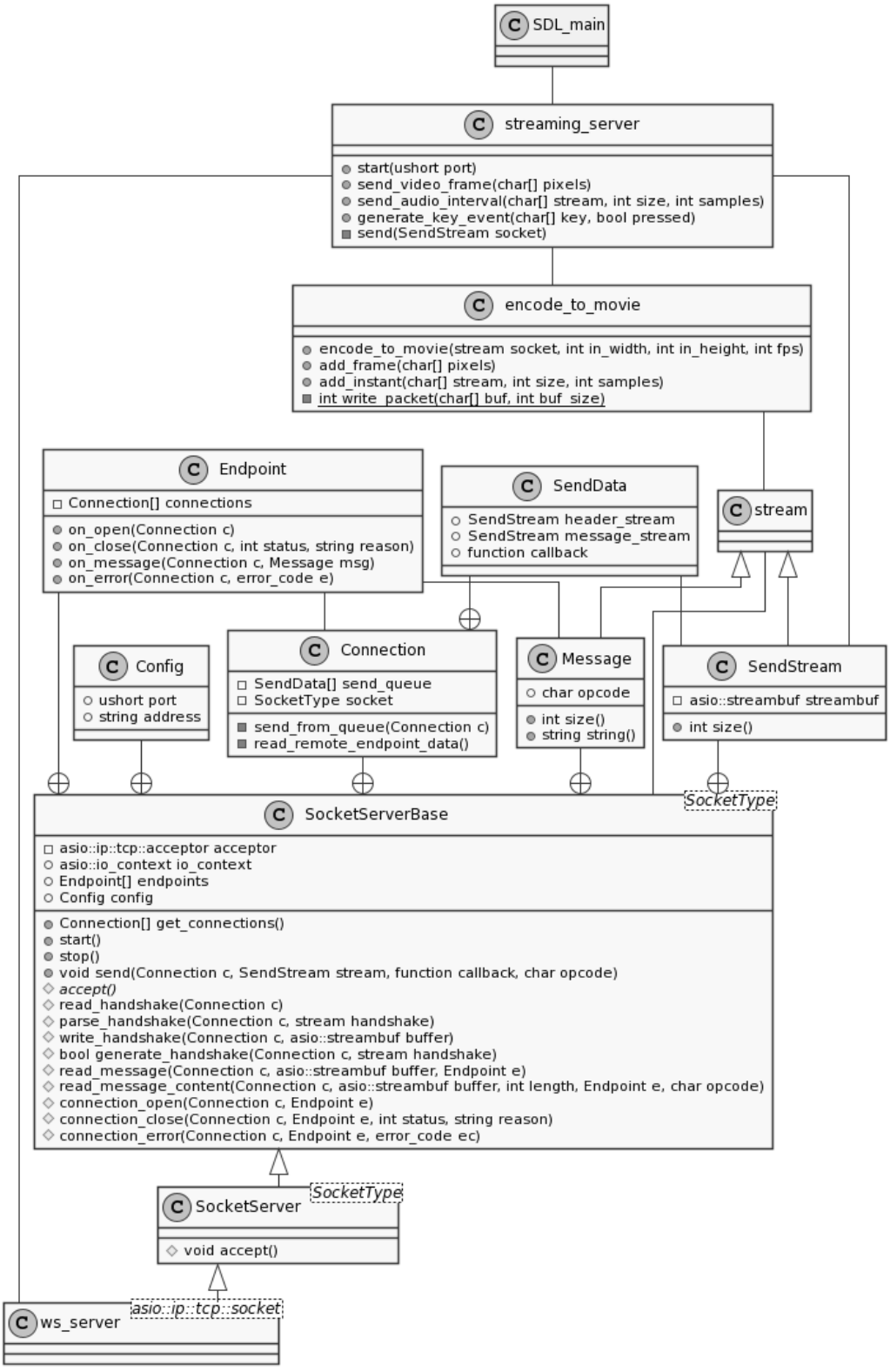

Figura 3.9: Diagramma delle classi del server

```
void accept() override
{
    auto connection = new Connection(new SocketType(io_context));
    acceptor->accept(connection->socket);
    read_handshake(connection);
}
```

Listato 3.11: Funzione accept della classe SocketServer. File: server_ws_impl.hpp

`SocketServerBase` offre le funzioni per avviare e fermare il server, gestire la connessione, scrivere e leggere tramite socket e gestire l'handshake.

La classe `SocketServerBase` espone gli stessi quattro eventi delle API WebSocket che vengono gestiti da `SocketServerBase` ed utilizzati da `streaming_server`, come mostrato in Lis. 3.12:

- *open*: viene avviato il thread per eseguire l'emulazione del gioco selezionato;
- *close*: viene spenta la macchina emulata;
- *error*: viene stampato a video il messaggio d'errore, dopodiché il browser spegne la connessione e si verifica l'evento *close*;
- *message*: alla ricezione di un messaggio viene gestito tramite la funzione `process_key` l'input utente oppure il meccanismo di pausa dell'emulazione in caso di bit-rate insufficiente (è meglio mettere in pausa che far perdere il giocatore).

```
endpoint.on_open = [this](auto connection)
{
    game_thread = new thread(on_accept, connection->parameters);
};

endpoint.on_close = [this](auto connection, auto status, auto reason)
{
    machine->schedule_exit();
};

endpoint.on_error = [](auto connection, auto code)
{
    std::cout
        << "-Error on connection from "
        << connection->remote_endpoint_address << ":"
        << connection->remote_endpoint_port << std::endl << ": "
        << code.message() << std::endl;
};

endpoint.on_message = [this](auto connection, auto message)
{
    const auto msg = message->string();
    const auto values = split(msg, ":");

    if (values[0] == "ping")
        process_pausing_mechanism();
```

```
27      else if (values[0] == "key")
28          process_key(values);
29  };
```

Listato 3.12: Codice relativo alle callback WebSocket. File: streaming_server.hpp

Nel `main` del programma è implementata la funzione `on_accept` che avvia l'emulazione della macchina arcade selezionata, come mostrato in Lis. 3.13.

```
1  streaming_server::instance().on_accept = [&](auto parameters)
2  {
3      main_sdl(argc, argv, parameters["game"]);
4  };
```

Listato 3.13: Codice relativo alla callback di avvenuta connessione. File: sdlmain.cpp

3.4 Decodifica

Nel paradigma del cloud gaming la decodifica viene eseguita lato client. Questa fase è vincolata dai codec installati sul sistema operativo del giocatore, per questo motivo quasi tutte le piattaforme di cloud gaming, come mostrato in Tab. 3.5, necessitano dell'installazione di un software client proprietario. Amazon Luna e Google Stadia, che utilizzano la tecnologia WebRTC, sono attualmente le uniche piattaforme ad offrire la possibilità di poter giocare tramite il browser (solo Chrome è supportato). Come detto precedentemente questo progetto mira ad offrire il più semplice accesso alla piattaforma, per questo motivo si è scelto come client il browser web e quindi la fase di decodifica viene eseguita dal browser utilizzando una libreria JavaScript.

Piattaforma	Protocollo	Installazione app.	Client
Amazon Luna	RTP	sì/no	Chrome, iOS, Android
GeForce Now	RTP	sì	Windows, macOS, Android
Google Stadia	RTP	sì/no	Chrome, iOS, Android
MAME CGP	WebSocket	no	Browser
Playkey	UDP	sì	Windows, macOS
PlayStation Now	UDP	sì	PS4, PS5, Windows
Vortex	UDP	sì	Windows, macOS, Android
Xbox Cloud Gaming	UDP	sì	Android

Tabella 3.5: Piattaforme disponibili

I moderni browser web offrono due API per la grafica, le Canvas API e WebGL, e le Web Audio API per la riproduzione sonora.

Le Canvas API forniscono un mezzo per disegnare grafica tramite JavaScript, si concentrano principalmente sulla grafica 2D ma quando vengono utilizzate dalle API WebGL possono disegnare grafica 2D e 3D con accelerazione hardware. Sono completamente supportate da tutti i browser [Mozilla 2021b].

WebGL è una API JavaScript, progettata e gestita dal gruppo no-profit Khronos, per il rendering di grafica 2D e 3D. WebGL 2.0 è supportato da tutti i moderni browser[11] [Mozilla 2021e].

[11] Il supporto a WebGL 2.0 su Safari è disponibile nella versione *Technology Preview 15* su macOS e abilitando le *Experimental Feautures* nella versione *14.6* su iOS.

Su queste API sono state create delle librerie JavaScript per la decodifica di vari formati audiovisivi.

3.4.1 Le librerie JavaScript per la decodifica

Per questo progetto sono state provate varie librerie open source:

- *jMuxer* per la decodifica audiovisiva dei formati AVC e AAC. Necessita di due stream separati, uno per l'audio ed uno per il video;
- *ogv.js* in grado di decodificare i seguenti formati audiovisivi open source: Vorbis, Theora, Opus, VP8, VP9 e AV1;
- *JSMpeg* in grado di decodificare il formato MPEG-TS con i codec MPEG-1 per il video e MP2 per l'audio.

jMuxer, come mostrato nel Lis. 3.14, richiede che lo stream audio e video siano separati e che venga specificata la durata in millisecondi del pacchetto audio/video.

```
let player = new JMuxer({
    node: 'myCanvas',
    mode: 'both', // available values are: both, audio and video
});

player.feed({
    audio: audio,
    video: video,
    duration: duration // in ms
});
```

Listato 3.14: Esempio di utilizzo di jMuxer

La libreria *ogv.js*, come mostrato nel Lis. 3.15, è anche essa molto semplice da usare ed è in grado di decodificare molti formati open source; purtroppo nella versione attuale può leggere solamente file completi.

```
let player = new OGVPlayer();

myCanvas.appendChild(player);

player.src = URL;
player.play();
```

Listato 3.15: Esempio di utilizzo di ogv.js

JSMpeg si è rivelato relativamente facile da usare. Come mostrato nel Lis. 3.16, si possono specificare le dimensioni del buffer video e di quello audio. Questa funzionalità è stata molto utile per ridurre l'effetto di latenza percepito dall'utente, poiché non è possibile usare buffer nel paradigma del cloud gaming. La libreria è in grado di acquisire il filmato tramite connessione WebSocket (questa operazione ha richiesto una modifica al sorgente della libreria).

```
let player = new JSMpeg.Player(URL, {
    canvas: 'myCanvas',
    autoplay: true,
```

```
4       loop: false,
5       pauseWhenHidden: false,
6       videoBufferSize: 64 * 1024,
7       audioBufferSize: 48 * 1024,
8   });
```

Listato 3.16: Esempio di utilizzo di JSMpeg

La scelta finale è ricaduta su *JSMpeg* sia perché offre le funzionalità per gestire i buffer sia perché supporta il protocollo WebSocket. *JSMpeg* è una libreria composta da un demuxer MPEG-TS, un decoder video MPEG-1 e audio MP2, con un sistema di rendering basato sia su WebGL che su Canvas2D, ed un sistema di output audio basato su Web Audio. La libreria consente lo streaming a bassa latenza (∼50 ms) tramite WebSocket, ed è rilasciata con licenza MIT [Szablewski 2021]. MPEG verrà descritto nel paragrafo 3.6.

3.5 Cattura dell'input utente

L'ultima fase del cloud gaming è la cattura dell'input utente da inviare al server. Questa fase viene eseguita lato client, nel nostro caso sarà il browser ad eseguirla. MAME CGP consente di poter giocare sia tramite tastiera che tramite joystick a più giocatori sullo stesso dispositivo. Per far ciò sono state utilizzate due librerie JavaScript open source per la gestione dell'input:

- Keypress è una libreria per la cattura dell'input da tastiera specializzata per l'uso in contesti videoludici, rilasciata con licenza Apache 2.0 [Mauro 2021];
- GameController.js è una libreria che estende le Web API per il gamepad, rilasciata con licenza MIT [Montoro 2021].

Come mostrato nel Lis. 3.17, tratto dal file `MAME.js`, la funzione `InitKeyboard` si occupa di inizializzare la libreria *Keypress* mentre la funzione `InitGamepad` la libreria GameController. I tasti mappati sono: le quattro direzioni, gli otto tasti azione e i tasti funzione start, gettone e pausa. Nel listato è mostrata la mappatura del tasto pausa per la tastiera e del tasto R2 per il joystick. In entrambi i casi viene gestita la pressione e il rilascio.

```
1   function InitKeyboard()
2   {
3       keypress_Listener = new window.keypress.Listener();
4
5       if (!keypress_Listener)
6           return; // tastiera non presente
7
8       keypress_Listener.register_many([
9           {
10              "keys": "p",
11              "on_keydown": ()=> InputSend(keyboard, 'PAUSE', true),
12              "on_keyup": (e)=> InputSend(keyboard, 'PAUSE', false),
13          },
14          // continua ...
15      ]);
16  }
17
```

```
18 function InitGamepad()
19 {
20     gameControl.on('connect', gamepad =>
21     {
22         gamepad.before('r2', ()=> InputSend(gamepad.id, 'R2', true));
23         gamepad.after('r2', ()=> InputSend(gamepad.id, 'R2', false));
24         // continua ...
25     }
26 }
```

Listato 3.17: Gestione input lato client. File: MAME.js

3.5.1 Invio dell'input alla libreria SDL

Come detto nel paragrafo 2.2.3 l'input è gestito tramite la libreria SDL. Come mostrato nel Lis. 3.12 alla ricezione di un messaggio di tipo "key" il server chiama la funzione `process_key` che si occupa di analizzare i dati contenuti nel messaggio e chiamare la funzione `generate_key_event`, mostrata nel Lis. 3.18, che invia alla classe device l'input ricevuto tramite WebSocket. Nel Lis. 3.19 è mostrata l'unica modifica apportata alla classe `sdl_keyboard_device` che serve a passare per riferimento il device SDL alla classe `streaming_server`.

```
1 void generate_key_event(char[] key, bool down)
2 {
3     SDL_Event e;
4     e.type = down ? SDL_KEYDOWN : SDL_KEYUP;
5     e.key.keysym.scancode = SDL_GetScancodeFromName(key);
6     e.key.keysym.sym = SDL_GetKeyFromScancode(e.key.keysym.scancode);
7
8     keyboard->queue_events(&e, 1);
9 }
```

Listato 3.18: Gestione input lato server: modulo server. File: streaming_server.hpp

```
1 // costruttore
2 sdl_keyboard_device(running_machine &r, char[] n, char[] id,
3     input_module &m): sdl_device(r, n, id, DEVICE_CLASS_KEYBOARD, m)
4 {
5     streaming_server::instance().set_keyboard(this);
6 }
```

Listato 3.19: Gestione input lato server: modulo SDL. File: input_sdl.cpp

3.6 MPEG-1

MPEG-1 è uno standard (ISO/IEC 11172) del *Moving Pictures Experts Group* del 1991 per la compressione audiovisiva con perdita. È stato progettato per lo standard NTSC (240p a 30 fps) per un bit-rate di $1,5$ Mbps ma supporta risoluzioni fino a 4095×4095 px ed un bit-rate fino a 100 Mbps. In questo progetto è stato utilizzato con una risoluzione di 480p a 30 fps per un bit-rate massimo di $2,1$ Mbps (dipende dalla dinamicità delle schermate del gioco e dalla presenza di musica); questa impostazione può essere cambiata a riga di comando come mostrato nel paragrafo A.3. Di seguito vedremo il processo di codifica video ed audio e il trasporto.

3.6.1 Video

Un filmato *MPEG-1 Video* è suddiviso in 6 layer, come mostrato in Fig. 3.13, su cui si applicano dei processi di codifica e decodifica formati da vari algoritmi. Il processo di codifica, mostrato in Fig. 3.10, utilizza l'algoritmo di compressione (con perdita) JPEG:

1. L'immagine viene elaborata in blocchi da 8×8 px;
2. Viene applicata, su ogni blocco, la trasformata discreta del coseno (DCT) prima orizzontalmente e poi verticalmente;
3. Ogni blocco viene compresso tramite quantizzazione.

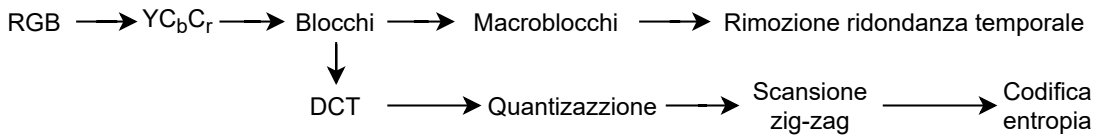

Figura 3.10: Fasi della codifica video

Il processo completo di codifica e decodifica è mostrato in Fig. 3.11.

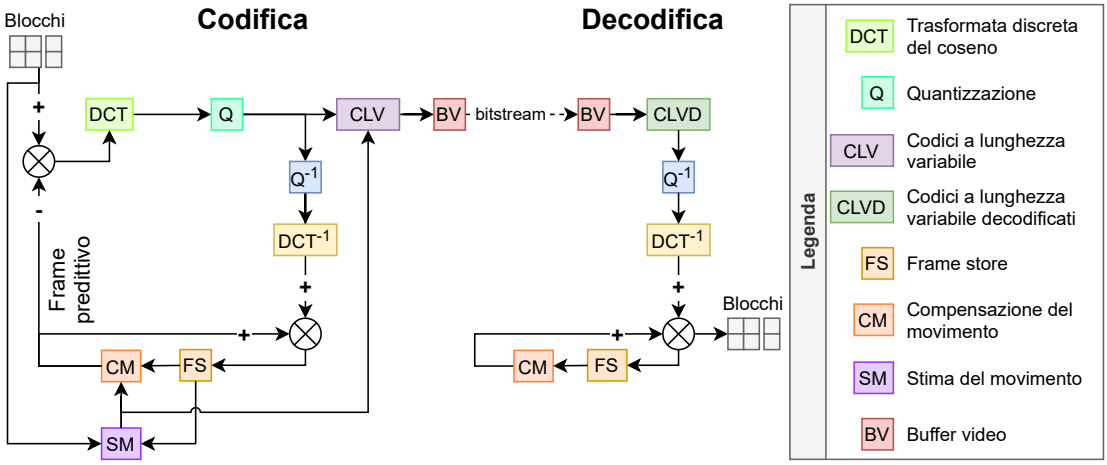

Figura 3.11: *MPEG-1 Video*: schema di codifica e decodifica

Frame e macroblocchi

In *MPEG-1 Video* la sequenza video è composta da "gruppi di immagini" (GOP: *group of pictures*); ogni GOP è composto da un *I-frame*, da una sequenza di *P-frame* e *B-frame* e da un *D-frame*, come mostrato in 3.12.

Una configurazione tipica è formata da 9 frame di cui 2 *P-frame* e 6 *B-frame*. Ogni GOP è formato da quattro diversi tipi di frame [Rupp 2008]:

- *I-frame* (*Intra-frame*) sono chiamati anche *key-frame* perché sono codificati senza utilizzare informazioni diverse da quelle contenute nell'immagine stessa;

- *P-frame* (*Predicted-frame*) sfruttano la ridondanza temporale per aumentare la compressione. Memorizzano solamente le differenze nell'immagine rispetto al frame precedente;
- *B-frame* (*Bidirectional-frame*) come i *P-frame* fanno una predizione però sfruttando sia il frame precedente che quello successivo;
- *D-frame* mentre gli altri tre sono presenti in quasi tutti i codec video i *D-frame* sono presenti solamente in *MPEG-1 Video*. La loro funzione è di fornire un'anteprima del video in un punto (usata nei visualizzatori video). Sono frame a bassa qualità (proprio perché le anteprime solitamente sono molto piccole) ed oggi la loro funzione è stata sostituita dagli *I-frame*.

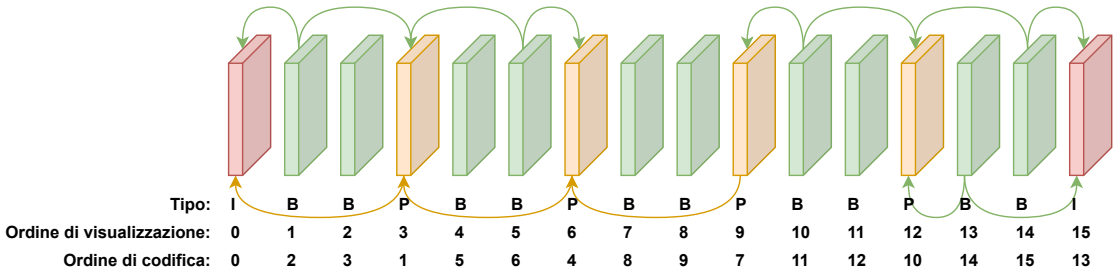

Figura 3.12: GOP: ogni *I-frame* inizia un nuovo GOP; le frecce indicano da che frame prendono informazioni

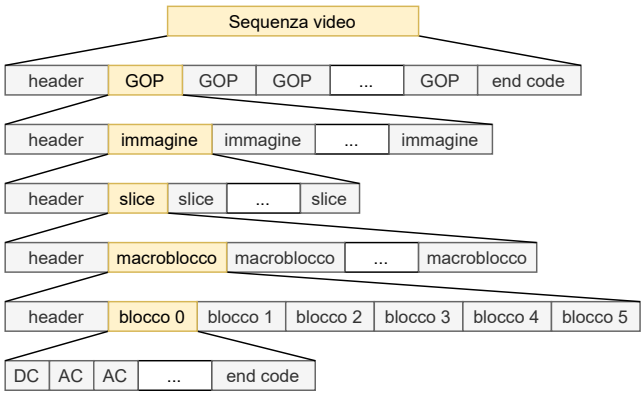

Figura 3.13: Layer di *MPEG-1 Video*

Come mostrato in Fig. 3.13, ogni immagine (frame) è divisa in *slice*. Le slice sono di 3 tipi: Y, Cb e Cr. Ogni *slice* è scomposta in macroblocchi, che sono l'unità d'elaborazione della compressione. I macroblocchi sono formati da 16×16 px e sono necessari ai fini del calcolo dei vettori di movimento e dei blocchi di errore per la compensazione del movimento. I macroblocchi a loro volta, utilizzando un campionamento 4:2:0[12], sono formati da 4 blocchi per la luminanza (Y di YCbCr) e 2 per la crominanza (Cb e Cr) come mostrato in Fig. 3.14. Essi si dividono in [Del Bimbo 2018]:

[12]Il campionamento 4:2:0 riduce di un fattore 2 in ambedue le direzioni.

- *macroblocchi I* codificato indipendentemente da altri macroblocchi (dalla trasformazione coseno discreta 2D come nei blocchi JPEG);
- *macroblocchi P* codificare non la regione ma il vettore di movimento e il blocco di errore del fotogramma precedente (macroblocco previsto in avanti);
- *macroblocchi B* come sopra tranne che il vettore di movimento e il blocco di errore sono codificati dal frame precedente (macroblocco previsto in avanti) o successivo (macroblocco previsto all'indietro).

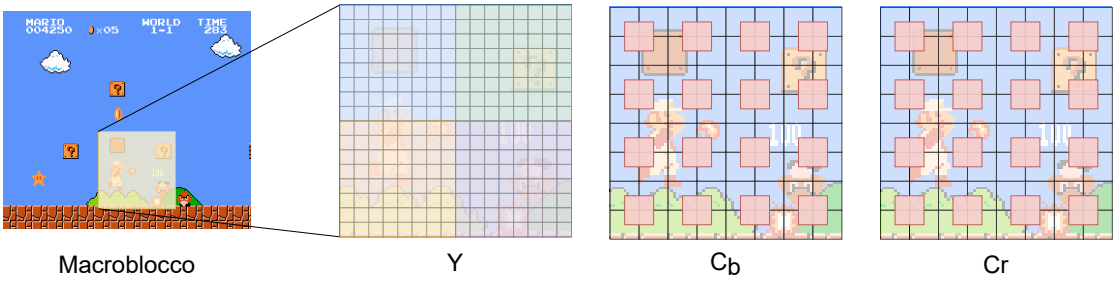

Figura 3.14: Disposizione blocchi per il campionamento 4:2:0. © Nintendo

I frame possono contenere differenti tipi di macroblocchi:

- *I-frame* contiene solamente macroblocchi *I*;
- *P-frame* può contenere macroblocchi *I* o *P*;
- *B-frame* può contenere tutti i tipi di macroblocchi (*I*, *P* o *B*).

Per ogni *I-frame* i macroblocchi vengono ricalcolati. Per i *P-frame* e *B-frame* i macroblocchi possono essere: ricalcolati, copiati, spostati o modificati [The Moving Picture Experts Group 2003].

Dal dominio spaziale a quello delle frequenze: la DCT

La trasformata discreta del coseno (DCT)[13] divide la forma d'onda nelle sue componenti frequenziali, di per se non effettua nessuna compressione ma trasforma i pixel in una forma nella quale è possibile identificare la ridondanza, poiché i coefficienti della DCT sono correlati alle frequenze spaziali dell'immagine. Inoltre non tutte le frequenze spaziali sono presenti e per questo si avranno, come risultato, molti coefficienti prossimi allo zero.

I blocchi degli *I-frame* sono trasformati dal dominio spaziale a quello delle frequenze utilizzando la DCT, definita in Eq. 3.2, che viene eseguita sui blocchi da 8×8 px della luminanza e della crominanza. Questo processo divide l'immagine in parti (in sottobande spettrali) di diversa importanza [Lim, Piuri e Swartzlander 2000].

$$
\begin{aligned}
y(k) &= \sqrt{\frac{2}{N}} \cdot u(k) \cdot \sum_{n=0}^{N-1} x(n) \cdot cos \frac{(2n+1)k\pi}{2N}, \quad \forall k = 0, 1, \ldots, N-1 \\
u(k) &= \begin{cases} \frac{1}{\sqrt{2}}, & \text{se } k = 0 \\ 1, & \text{altrimenti} \end{cases}
\end{aligned}
\tag{3.2}
$$

[13] La forma più comune è la DCT-II che è quella utilizzata in questo contesto.

La quantizzazione e la scansione zig-zag

Successivamente la matrice di 64 frequenze del blocco DCT viene uniformemente quantizzata (dividendo per lo *step-size*) utilizzando la matrice di quantizzazione in Eq. 3.3. Il sistema visivo umano è più sensibile agli errori di ricostruzione relativi alle frequenze spaziali più basse per questo si migliora la qualità visiva dell'immagine decodificata utilizzando dei pesi tarati sulle proprietà fisiche del sistema visivo umano. La quantizzazione è un processo che introduce perdita di qualità del segnale [Del Bimbo 2018].

$$\begin{bmatrix} 8 & 16 & 19 & 22 & 26 & 27 & 29 & 34 \\ 16 & 16 & 22 & 24 & 27 & 29 & 34 & 37 \\ 19 & 22 & 26 & 27 & 29 & 34 & 34 & 38 \\ 22 & 22 & 26 & 27 & 29 & 34 & 37 & 40 \\ 22 & 26 & 27 & 29 & 32 & 35 & 40 & 48 \\ 26 & 27 & 29 & 32 & 35 & 40 & 48 & 58 \\ 26 & 27 & 29 & 34 & 38 & 46 & 56 & 69 \\ 27 & 29 & 35 & 38 & 46 & 56 & 69 & 83 \end{bmatrix} \tag{3.3}$$

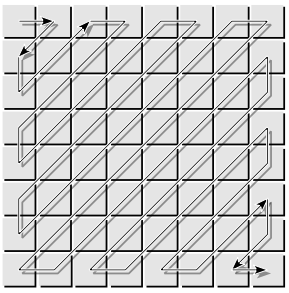

Figura 3.15: Scansione zig-zag

Dopo la quantizzazione, il coefficiente DCT con la varianza più alta (coefficiente DC) viene trattato in modo diverso dai coefficienti rimanenti (coefficienti AC). Il coefficiente DC corrisponde all'intensità media del blocco. I valori del quantizzatore diversi da zero dei restanti coefficienti DCT vengono scansionati "a zig-zag", come mostrato in Fig. 3.15, e codificati con l'entropia della lunghezza di esecuzione utilizzando tabelle di "codici a lunghezza variabile". La scansione zig-zag ispeziona in successione ed in ordine crescente gli elementi.

Stima predittiva e compensazione del movimento

La stima predittiva del movimento mira a ridurre i dati trasmessi calcolando il movimento degli oggetti in una sequenza video, risparmiando tra il 50% e il 80% di bit da memorizzare. I passaggi di questo processo sono:

1. trova il macroblocco più adatto nel frame di riferimento cercando in una determinata area, questo è il macroblocco predetto;
2. invia i coefficienti DCT quantizzati della differenza tra i due frame, chiamata errore di predizione;
3. codifica e invia il vettore movimento tra la posizione del macroblocco predetto e quello del frame di riferimento.

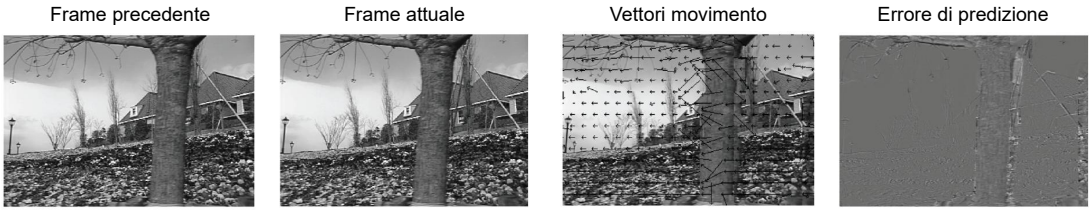

Figura 3.16: Compensazione del movimento: vettori di movimento ed errore di predizione. Fonte: David A. Forsyth et al. (2015). *Computer vision: a modern approach*. Pearson Education Limited.

Il processo di applicare i vettori movimenti ad un frame per sintetizzare la trasformazione al frame successivo è chiamato compensazione del movimento e si applica solamente ai *P-frame* e *B-frame*.

Un esempio molto semplice dei vettori movimento è mostrato in Fig. 3.17, mentre in Fig. 3.16 è mostrata la compensazione del movimento.

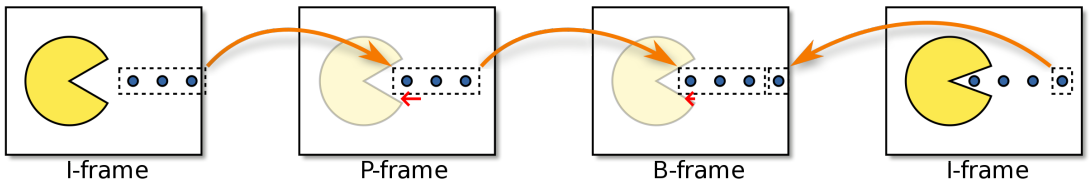

Figura 3.17: Vettori movimento. Le frecce arancioni che partono dal frame di riferimento al frame di predizione indicano i frame che l'algoritmo prende in considerazione; le frecce rosse indicano il vettore movimento calcolato. Fonte: wikipedia.org

L'errore di predizione può essere calcolato tramite *mean absolute error*, *sum of squared error* o *mean squared error*. Solitamente implementato utilizzando *mean absolute error*, utilizzando l'Eq. 3.4, dove C_{ij} è il frame corrente e R_{ij} quello di riferimento [Del Bimbo 2018]:

$$\frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|C_{ij}-R_{ij}| \tag{3.4}$$

Run-length encoding e la codifica di Huffman

Il *run-length encoding* (RLE) è un algoritmo di compressione senza perdita di informazione. Sostituisce nell'input le sequenze di caratteri uguali con il numero di occorrenze consecutive, il carattere ed un simbolo.

Ad esempio la stringa `AAAAAAFDDCCCCCCCAEEEEEEEEEEEEEEEE` diventa `6A:1F:2D:7C:1A:16E`.

La codifica di Huffman è anche esso un algoritmo per la compressione senza perdita di informazione. L'algoritmo crea un albero binario di simboli ordinati in base al conteggio delle loro occorrenze. Agli elementi che si ripetono più spesso viene assegnato il codice più breve.

I coefficienti AC sono codificati senza perdite secondo il RLE e memorizzati utilizzando la codifica di Huffman, mentre i coefficienti DC codificano le differenze tra blocchi dei macroblocchi.

Al termine di queste due operazioni si ottiene il bitstream del video [Jurgem 1997].

3.6.2 Audio

MPEG-1 Audio riduce il quantitativo di dati da utilizzare per l'audio tramite psicoacustica, eliminado frequenze che l'orecchio umano non può sentire. Lo standard sonoro è diviso in 3 layer:

- Layer I: è progettato per la codifica in tempo reale tramite hardware con un bit-rate di 384 kbps. L'estensione del file è `.m1a`;
- Layer II: è un formato con perdita proggettato per l'alta qualità a 192 kbps. L'estensione del file è `.mp2`;
- Layer III: anche esso un formato con perdita progettato per una buona qualità a 128 kbps; L'estensione del file è `.mp3`.

Come detto nel paragrafo 3.4.1, la libreria *JSMpeg* supporta *MPEG-1 Audio Layer II (MP2)*.

La tecnica di codifica

La codifica, come mostrato in Fig. 3.18, opera su dati di tipo PCM a 16 bit campionati a 32, 44,1 o 48 kHz, che vengono decomposti in 32 sottobande utilizzando un filterbank PQMF[14]. Le sottobande sono sottoposte a compansione a blocchi dimodoché l'ampiezza massima dei campioni sia unitaria.

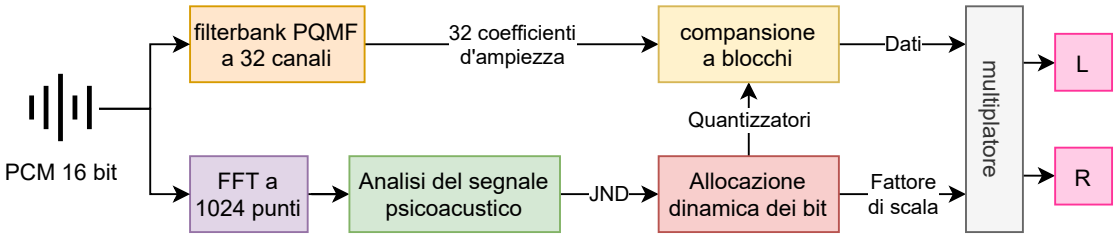

Figura 3.18: *MPEG-1 Audio*: schema di codifica

Parallelamente l'input PCM viene inviato ad una FFT[15] da 1024 punti, che scompone il segnale nelle sue frequenze costituenti, come mostrato in Fig. 3.19. Successivamente l'analisi del segnale psicoacustico scarta tutti i suoni sotto la *soglia assoluta della percezione sonora* (ATH[16]) insieme a quelli difficilmente audibili. Quindi una procedura di allocazione dinamica dei bit (iterativa) applica la soglia di "distorsione appena rilevabile" (JND[17]) per selezionare il quantizzatore migliore da un insieme predeterminato per ciascuna sottobanda [Spanias 2008].

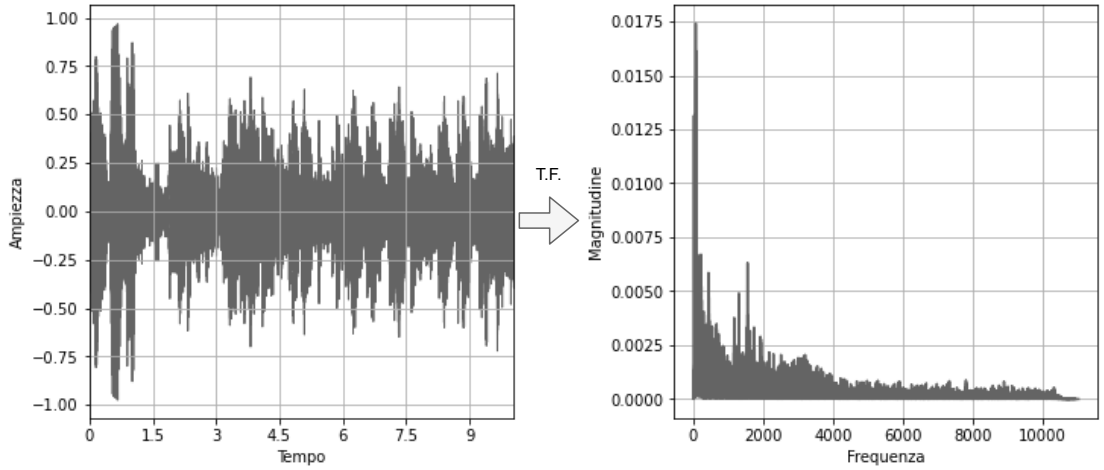

Figura 3.19: Applicazione della trasformata discreta di Fourier (1D) su 10 secondi di audio (PCM a 16 bit) tratti da *Street Fighter III*

[14]PQMF: *pseudo quadrature mirror filter* è un filterbank in cui la cancellazione dell'aliasing avviene solo tra bande adiacenti.

[15]FFT: *fast Fourier transform* (trasformata di Fourier veloce) è un algoritmo per calcolare la trasformata discreta di Fourier.

[16]ATH: *Absolute Threshold of Hearing*, stimata attorno i $10^{-12} W/m^2$.

[17]JND: *just noticeable distortion*.

3.6.3 Trasporto

Il trasporto definisce il modo in cui il filmato viene compresso in byte per la trasmissione da una parte all'altra, utilizzando un protocollo. MPEG, nel layer di sistema, definisce *Program Stream* (PS) e *Transport Stream* (TS), il primo usato per la memorizzazione sui DVD ed il secondo per i sistemi di trasmissione per la TV via cavo.

MPEG-TS è composto da più flussi audio e video (PES[18]), non necessariamente sincronizzati, che vengono multiplessati assieme e trasmessi sotto forma di pacchetti, come mostrato in Fig. 3.20.

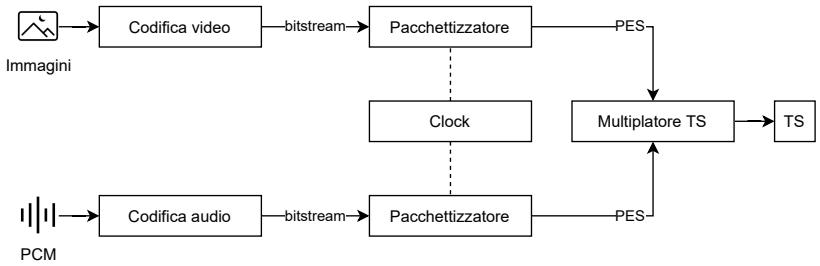

Figura 3.20: Schema di funzionamento del *Transport Stream*

I pacchetti TS sono formati da un header di 4 bytes e un payload di 184 bytes, come mostrato in Fig. 3.21, e sono composti da [Paris 2011]:

- *Synchronization Byte*: identifica l'inizio del pacchetto (è la costante 0x47);
- *Transport Error Indicator*: indica un errore nel pacchetto;
- *Payload Unit Start Indicator*: indica se il contenuto è un pacchetto PS;
- *Transport priority*: indica se ha priorità sugli altri pacchetti;
- *Packet Identifier*: identificatore del pacchetto;
- *Transport Scrambling Control*: indica se il pacchetto è mischiato e se con chiave pari o dispari;
- *Adaptation field control*: indica se il campo AF è settato;
- *Continuity counter*: numero di sequenza del pacchetto;
- *Adaptation field*: informazioni aggiuntive sul pacchetto;

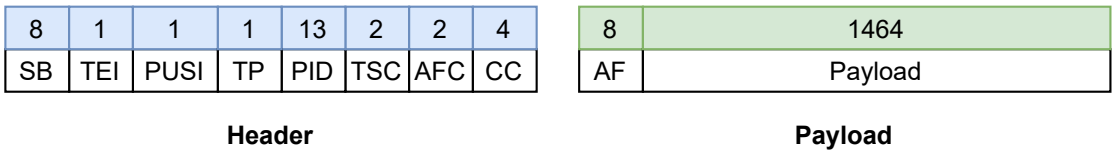

Figura 3.21: Pacchetto *MPEG-TS*

[18]PES: *Packetized elementary stream* definisce il trasporto dei flussi elementari (ES), cioè l'output di un codificatore audio o video.

Capitolo 4

Prestazioni

Questo capitolo analizza le prestazioni del progetto relativamente ai tre difetti intrinseci del cloud gaming: riduzione della qualità audiovisiva, il problema della latenza ed il bit-rate richiesto.

4.1 Riduzione della qualità audiovisiva

Nell'ambito videoludico la qualità audiovisiva (soprattutto quella visiva) influisce sulla qualità generale percepita dall'utente, per questo nel cloud gaming è necessario garantire un livello di qualità quasi identico alla situazione in cui il gioco viene eseguito in locale. È possibile preservare la qualità percepita durante lo streaming anche se si utilizzano bassi bit-rate, che sono particolarmente adatti per l'utilizzo in reti cellulari e WLAN [Rupp 2008].

4.1.1 La qualità visiva

Qualsiasi elaborazione applicata ad un'immagine può causare una perdita di qualità. La qualità dell'immagine si riferisce al modo in cui viene riprodotta la scena ripresa (o quella generata digitalmente) e può essere descritta dal degradarsi di alcune caratteristiche tra cui: nitidezza, gamma dinamica, fusione dei colori, contrasto, sfocatura, ecc.... I metodi di valutazione della qualità dell'immagine sono suddivisi in due categorie: soggettivi ed oggettivi. I metodi soggettivi sono composti da test psicofisici per la valutazione, da parte di osservatori, della qualità percepita. Questi test sono dispendiosi in termini di risorse e di tempo, sia per la preparazione che per l'esecuzione. Le valutazioni vengono poi utilizzate come termini di paragone per i metodi oggettivi [Shang et al. 2017]. I metodi oggettivi si basano su confronti utilizzando criteri numerici espliciti. L'errore quadratico medio (MSE), il *peak signal-to-noise ratio* (PSNR) e la *structural similarity index measure* (SSIM) sono esempi di misure di qualità utilizzate nell'analisi delle immagini [Horé e Ziou 2013].

Peak signal-to-noise ratio

Il PSNR è comunemente usato come misura della qualità della compressione e della riduzione del rumore nelle immagini digitali a scala di grigi. Esistono diversi approcci per calcolare il PSNR delle immagini a colori, ad esempio (il più semplice) utilizzando immagini in formato YCbCr e calcolando il PSNR solo sul canale della luminanza. Il PSNR è espresso solitamente in decibel con valori tipici per la compressione video per lo streaming tra 20 dB e 30 dB [Thomos, Boulgouris

e Strintzis 2006] e si calcola utilizzando l'Eq. 4.1, con $MSE(f, g)$ l'errore quadratico medio e $L = 255$ l'intervallo dinamico [Shang et al. 2017].

$$PSNR(f, g) = 10 \log_{10} \frac{L^2}{MSE(f, g)} \tag{4.1}$$

Structural Similarity Index Measure

SSIM è un metodo per predire la qualità percepita nelle immagini digitali. Invece di utilizzare i tradizionali metodi di somma degli errori, SSIM è progettato modellando qualsiasi distorsione dell'immagine come una combinazione di tre fattori: la perdita di correlazione, la distorsione della luminanza e la distorsione del contrasto. L'indice SSIM è un valore decimale compreso tra 0 e 1 e si calcola utilizzando l'Eq. 4.2 con [Horé e Ziou 2013]:

- $k_1 = 0,01$ e $k_2 = 0,03$ costanti;
- $L = 255$ l'intervallo dinamico;
- $c_1 = (k_1 L)^2$, $c_2 = (k_2 L)^2$ e $c_3 = c_2/2$ variabili di stabilizzazione della divisione;
- $l(f, g)$ funzione di comparazione della luminanza tra le medie μ_f e μ_g;
- $c(f, g)$ funzione di comparazione del contrasto tra le deviazioni standard σ_f e σ_g;
- $s(f, g)$ funzione di comparazione della struttura che misura il coefficiente di correlazione; σ_{fg} è la covarianza tra f e g.

$$\begin{aligned} &SSIM(f, g) = l(f, g)c(f, g)s(f, g) && l(f, g) = \tfrac{2\mu_f\mu_g + C_1}{\mu_f^2\mu_g^2 + C_1} \\ &c(f, g) = \tfrac{2\sigma_f\sigma_g + C_2}{\sigma_f^2 + \sigma_g^2 + C_2} && s(f, g) = \tfrac{\sigma_{fg} + C_3}{\sigma_f\sigma_g + C_3} \end{aligned} \tag{4.2}$$

Risultati ottenuti

Tramite lo script in Lis. 4.1 è stata calcolata la qualità video per la codifica *MPEG-1 Video* a 1024 kbps che ha dato come risultati un valore di PSNR di $27,703$ dB e un indice SSIM del $89,9\%$. Alcune immagini d'esempio del risultato ottenuto sono visibili in Fig. 4.1.

```
1 imgRGB = skimage.io.imread("frameRGB.bmp")
2 imgMPEG1 = skimage.io.imread("frameMPEG1.bmp")
3
4 PSNR = skimage.metrics.peak_signal_noise_ratio(imgRGB, imgMPEG1)
5
6 SSIM = skimage.metrics.structural_similarity(imgRGB, imgMPEG1,
7     multichannel=True)
```

Listato 4.1: Script Python per il calcolo di PSNR e SSIM

Figura 4.1: Alcuni frame d'esempio: a sinistra i frame renderizzati dal MAME (RGBA), a destra i frame codificati in *MPEG-1*. © Capcom, Data East, Taito

4.1.2 La qualità uditiva

Gli algoritmi sviluppati per misurare oggettivamente la qualità audio percepita simulano le proprietà percettive dell'orecchio umano ed applicano modelli informatici per stimare le similarità tra due segnali audio. Tra questi algoritmi abbiamo *Perceptual Evaluation of Audio Quality* (PEAQ), *Perceptual Objective Listening Quality Assessment* (POLQA) ed *Evaluation of Audio Quality* (EAQUAL). Questi algoritmi sono brevettati e protetti da licenza ma per PEAQ è dispo-

nibile un'implementazione per uso accademico a cura di Giuseppe Gottardi[1] [Pocta e Beerends 2015].

Perceptual Evaluation of Audio Quality

Perceptual Evaluation of Audio Quality (PEAQ) è un algoritmo sviluppato dal ITU-R[2] che simula le proprietà percettive dell'orecchio umano tramite modelli informatici, per determinare il livello di rumore che si può introdurre in un segnale acustico prima che esso diventi udibile. Per PEAQ sono stati creati due modelli, *basic* e *advanced*, il modello avanzato viene utilizzato per test più approfonditi [Ulovec e Smutny 2018]. Gli output dell'algoritmo sono il *Objective Difference Grade* (ODG) e l'*indice di distorsione* (DI) che vengono assegnati ad ogni frame audio, in questo modo si può intervenire precisamente sull'algoritmo di codifica per migliorarlo. Facendo la media degli ODG di tutti i frame si può dare una stima della qualità di tutto il segnale. La scala ODG è basata su un test uditivo soggettivo (ITU-R BS.1116) ed è la seguente: *molto fastidioso* (1), *fastidioso* (2), *leggermente fastidioso* (3), *percettibile ma non fastidioso* (4), *impercettibile* (5) [Pocta e Beerends 2015].

Il metodo, illustrato in Fig. 4.2, inizia applicando un allineamento temporale tra i due segnali in ingresso (alcuni codificatori slittano leggermente il segnale in avanti), questi vengono inviati al modello percettivo. I segnali audio vengono tagliati in frame di $0,042$ s con una sovrapposizione del 50%. Il modello percettivo trasforma i segnali utilizzando una funzione finestra e una *FFT*. Successivamente vengono applicati dei pre-processamenti per calcolare: i modelli di eccitazione, i modelli di volume, i modelli di modulazione e il segnale di errore. Utilizzando questi modelli una funzione calcola le *Model Output Variables* (MOVs) che vengono mappate sul corrispettivo *ODG* [D. Câmpeanu e A. Câmpeanu 2021].

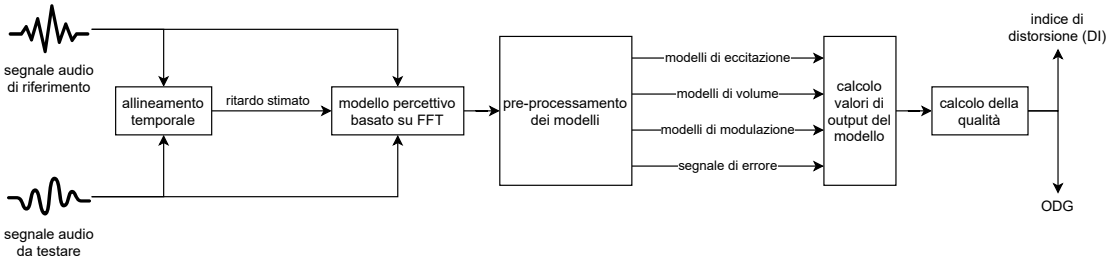

Figura 4.2: Diagramma del funzionamento di PEAQ

Risultati ottenuti

In questo progetto è stata usata un'implementazione per scopi accademici di PEAQ, con cui è stato calcolato per una partita di 2 minuti di *Street Fighter III*, codificato con *MPEG-1 Audio Layer II* (MP2) a 64 kbps, un grado PEAQ di 2 (*fastidioso*).

4.2 Il problema della latenza

Alle fasi di base di un videogioco (ricezione input, esecuzione, rendering e display) nel cloud gaming vanno ad aggiungersi: invio al server dell'input utente, codifica, trasmissione e decodifica.

[1] Il repository del progetto è disponibile al seguente indirizzo: sourceforge.net/projects/peaqb.

[2] ITU-R: *International Telecommunication Union - Radiocommunication Sector* è un organizzazione internazionale che si occupa degli standard nel campo delle telecomunicazioni, settore radio.

Queste fasi aggiuntive aumentano il tempo che intercorre tra l'input del giocatore e l'azione visualizzata sullo schermo.

Un leggero ritardo in un film in streaming o in una videochiamata molto probabilmente passa inosservato, ma durante una partita la latenza può rendere il gioco non fruibile, fortunatamente il rapido sviluppo delle reti a banda larga hanno reso questo problema meno evidente.

Secondo diversi studi, affinché il gameplay sia fluido e senza jitter[3], il ritardo tra la trasmissione dell'input utente e la visualizzazione dell'azione corrispondente, chiamato anche *round trip delay*, deve essere inferiore a una soglia tollerabile, altrimenti la reazione dell'utente viene visualizzata con un ritardo che rende il gameplay difficoltoso e irritante, soprattutto nei giochi in cui la velocità è un fattore molto importante, come gli sparatutto in prima persona e i giochi di strategia in tempo reale. In tabella 4.1 sono riportate le soglie di ritardo tollerabili per tipo di gioco [Shea et al. 2013].

Tipo di gioco	Prospettiva	Soglia di ritardo (ms)
FPS	Prima persona	100
RPG	Terza persona	500
RTS	Onnipresente	1000

Tabella 4.1: Ritardo tollerato per tipo di gioco

Risultati ottenuti

La latenza è stata calcolata cronometrando il *round trip delay* e calcolando le medie al termine della partita, lato server della codifica[4] e della creazione del pacchetto di rete, e lato client della decodifica. Per calcolare il *round trip delay* è stata eseguita una sessione di gioco di *Street Fighter III* ed è stato effettuato lo screencast utilizzando il programma *OBS Studio*[5] a 60 fps. Utilizzando una *tastiera su schermo* si preme il tasto *P* per mettere il gioco in pausa, e successivamente viene analizzato manualmente lo screencast, calcolando il *round trip delay* medio tra il frame in cui avviene la pressione del tasto e la visualizzazione a video del frame del gioco in pausa[6], come mostrato in Fig. 4.3. Il tempo medio di trasmissione[7] è stato calcolato come la differenza tra il *round trip delay* medio e le altre operazioni. I test sono stati effettuati su rete LAN utilizzando i computer descritti in Tab. 4.4 e Tab. 4.5. I tempi medi ottenuti sono mostrati in Tab. 4.2.

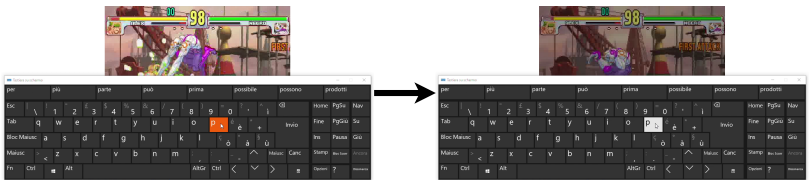

Figura 4.3: Frame di gioco (sx) e frame di pausa (dx). © Capcom

[3] Il jitter è la variazione del ritardo tra i pacchetti inviati.
[4] Il tempo medio di codifica contiene anche il tempo di cattura.
[5] OBS Studio è un software open source per lo streaming e la cattura video.
[6] In MAME il frame di pausa è un frame a cui viene abbassato il canale alfa su sfondo nero.
[7] Il tempo medio di trasmissione contiene anche il tempo di visualizzazione del frame sullo schermo.

Operazione	Tempo (ms)
Codifica	$6,71$
Creazione pacchetto di rete	$0,07$
Trasmissione	$\sim 174,00$
Decodifica	$1,04$
Round trip delay	$\sim 182,00$

Tabella 4.2: Tempi medi d'elaborazione per *Street Fighter III*

Sono stati effettuati dei test per la stima del tempo di *round-trip*[8] da vari comuni italiani utilizzando il server descritto in Tab. 4.4. I risultati ottenuti sono mostrati in Tab. 4.3.

Comune	Distanza (km)	Min (ms)	Max (ms)
Milano (MI)	~ 1540	20	28
Lamezia Terme (CZ)	~ 740	20	40
Olgiate Olona (VA)	~ 1620	28	38
Bologna (BO)	~ 1160	30	60
Campobasso (CB)	~ 300	30	60
Casalpusterlengo (LO)	~ 1440	35	40
Treviso (TV)	~ 1480	35	45
Basiano (MI)	~ 1580	30	50
Verona (VR)	~ 1420	30	70
Novara (NO)	~ 1640	30	80

Tabella 4.3: Tempo di *round-trip* da vari comuni italiani

Componente	Valore
CPU	AMD Ryzen 5 2500U (8 threads) @ $3,6$ GHz
GPU	AMD Radeon Vega 8 Mobile Graphics 4 GB
RAM	8 GB
Connessione internet	*download*$41,5$ mbps, *upload*$10,5$ mbps
S.O.	Fedora 34
Posizione	Sant'Anastasia (NA)

Tabella 4.4: Server utilizzato per il test

Componente	Valore
CPU	Intel Core2 Quad Q6600 (4 threads) @ $2,4$ GHz
GPU	Nvidia GeForce 9500 GT 1 GB
RAM	8 GB
S.O.	Windows 10
Browser	Firefox 88

Tabella 4.5: Client utilizzato per il test

[8]Con il termine *round-trip* si intende il messaggio che il client invia al server e che il server invia al client. In questo caso viene inviata anche una marca temporale.

4.3 Bit-rate richiesto

È necessario stimare il bit-rate richiesto per offrire all'utente un'esperienza di gioco priva di ritardi e con la miglior qualità audiovisiva. Il bit-rate richiesto è l'unico parametro che è influenzato dalla compressione video, e differisce tra i vari giochi. Calcolando i differenti bit-rate massimi per le differenti qualità audiovisive si può scegliere il miglior bilanciamento tra qualità offerta e bit-rate richiesto all'utente [Chen et al. 2014].

Risultati ottenuti

Il MAME CGP può essere avviato specificando la risoluzione video da utilizzare e come mostrato in Tab. 1.2 è stata usata una risoluzione di 480p, che con la compressione *MPEG-1 Video* ed *MPEG-1 Audio Layer II*, necessitano di un bit-rate medio di 1 Mbps (con picchi in alcuni casi fino a 3.5 Mbps). Utilizzando lo script in Lis. 4.2 sono stati estrapolati i dati mostrati in Tab. 4.6 e Fig. 4.4.

```
try:
    while True:
        new_value = psutil.net_io_counters().bytes_recv
        new_seconds += 0.1
        seconds.append(new_seconds)

        if old_value
            bw = (new_value - old_value) * 8. / 1024. / 1024.
        else:
            bw = 0

        bandwidth.append(bw)
        old_value = new_value
        time.sleep(0.1)
except KeyboardInterrupt:
    pass

print("Media:", statistics.mean(bandwidth))
print("Dev. std:", statistics.stdev(bandwidth))
print("Varianza:", statistics.variance(bandwidth))
```

Listato 4.2: Script Python per il calcolo del bit-rate richiesto

Gioco	Tipo	Media	Dev. std	Varianza
Castelvania	piattaforma	0.867	0.537	0.289
Metal Slug 3	piattaforma	0.688	0.360	0.129
Out Run	guida	0.835	0.443	0.196
Puzzle Booble 2	puzzle	0.731	0.320	0.102
Street Fighter III	picchiaduro	0.848	0.400	0.160
Street Slam	sport	1.075	0.555	0.308

Tabella 4.6: Bit-rate richiesto per vari tipi di giochi (in Mbps)

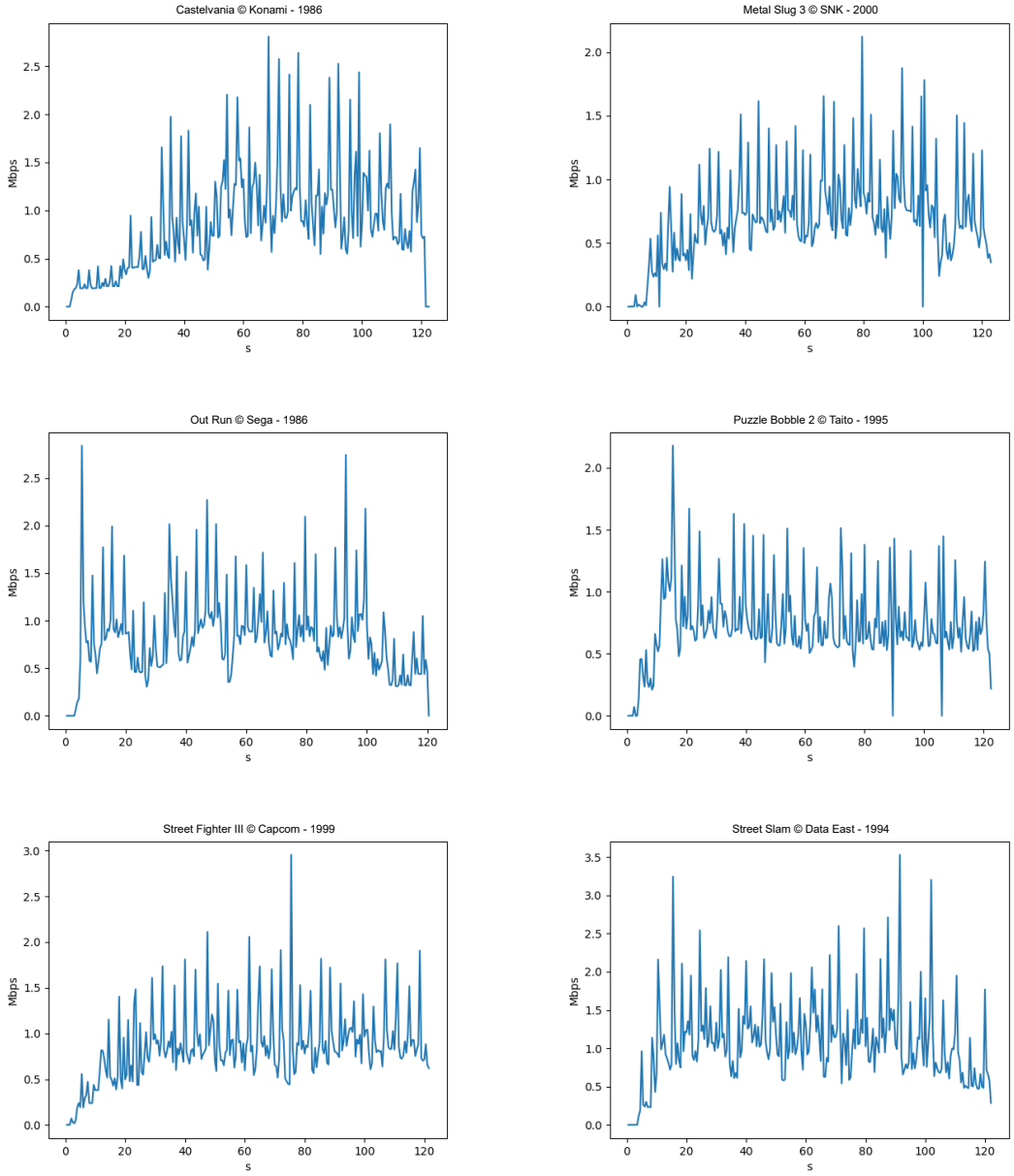

Figura 4.4: Bit-rate richiesto per vari tipi di giochi

4.4 Miglioramenti dell'esperienza di gioco

Le attuali tecnologie informatiche hanno reso possibile il cloud gaming e in seguito le nuove tecnologie andranno a migliorare l'esperienza di gioco. I nuovi formati per la compressione digitale aumenteranno la qualità audiovisiva e ridurranno il bit-rate richiesto. Alcuni esempi sono *H.266* del *Moving Picture Experts Group*, *AV1* del *Alliance for Open Media*, *Opus* e *Theora* del *Xiph.Org Foundation*. Per ridurre la latenza di rete nel paradigma del cloud computing si

ricorre all'utilizzo di sistemi distribuiti sul territorio, bilanciando il carico dei sistemi e istradando il traffico di rete dell'utente per vicinanza geografica [Sun 2019].

Nella progettazione del MAME CGP si è tenuto conto di queste problematiche lasciando aperte strade per futuri sviluppi. Come detto nel paragrafo 3.2, la classe che si occupa della codifica utilizza *FFmpeg* ed è quindi possibile sia selezionare un altro formato di compressione (a patto che sia supportato dal browser) sia cambiare durante l'esecuzione il livello di compressione per diminuire la latenza. Altre idee per futuri sviluppi verranno trattati nel capitolo seguente.

Direzioni future di ricerca e conclusioni

In questa tesi è stata proposta una piattaforma di cloud gaming per videogiochi arcade, le cui caratteristiche principali sono state la portabilità e la possibilità di utilizzarla senza dover installare software aggiuntivi. Queste caratteristiche si sono tradotte in una compilazione multipiattaforma lato server e un'interfaccia web lato client.

Il lavoro comprende un'analisi delle varie piattaforme di cloud gaming del passato, di quelle attualmente sul mercato e di alcune piattaforme open source; questa analisi è stata la base di partenza per la progettazione della piattaforma. Il lavoro è continuato con la modifica del codice sorgente del software MAME (licenza GNU-GPL) tramite l'implementazione in C++ delle cinque fasi aggiuntive del cloud gaming: cattura, codifica, decodifica, trasmissione e gestione dell'input utente. Come formato audiovisivo è stato utilizzato lo standard *MPEG-1*. La valutazione dei tre difetti intrinseci del cloud gaming e delle prestazioni ottenute concludono il lavoro.

I tre difetti intrinseci del cloud gaming sono stati valutati con l'utilizzo di metodi oggettivi. La qualità video ha ottenuto un valore di PSNR di $27,703$ dB, che ha valori tipici tra 20 dB e 30 dB per la compressione video per lo streaming. Mentre l'indice SSIM è risultato del $89,9\%$. La qualità audio ha ottenuto un grado PEAQ di 2 che nella scala ODG corrisponde a fastidioso. La latenza è stata calcolata cronometrando il round trip delay medio che è risultato di 182 ms; il valore risulta leggermente superiore alla soglia di ritardo per i giochi FPS (100 ms) ma inferiore a quella degli RPG (500 ms). Il bit-rate richiesto per la risoluzione di 480p oscilla, in base al tipo di gioco, tra $0,5$ e $3,5$ Mbps con una media di 1 Mbps, rendendo il sistema perfettamente fruibile nelle moderne reti LAN e WAN.

I test preliminari sono stati effettuati con un numero contenuto di utenti dai quali sono stati ricevuti feedback positivi sia riguardo le performance sia riguardo l'interesse che hanno ancora oggi i videogiochi arcade. Una valutazione più completa è rimandata ad un lavoro futuro.

Il progetto apre le porte a molte idee e miglioramenti che possono portarlo a essere effettivamente competitivo. Molto utile per la fidelizzazione dell'utente sarebbe la creazione di un sistema di account tramite cui il giocatore può salvare e caricare lo stato del gioco, pubblicare i punteggi nella leaderboard, invitare altri giocatori a unirsi alla partita supportando così il multiplayer da devices differenti. Per ridurre la dimensione dei pacchetti si potrebbero usare due codec open source che in futuro saranno supportati nativamente dalla maggior parte dei browser: *AOMedia Video 1 (AV1)*, progettato per trasmissioni video su Internet, e *Opus*, un codec audio lossy utilizzato per la comunicazione in tempo reale; entrambi inseribili nel contenitore *WebM*. Per diminuire l'overhead di comunicazione si potrebbe sostituire il protocollo *WebSocket* con *RTP* utilizzabile tramite la tecnologia *WebRTC*. Nella sua versione attuale il progetto è in ascolto su una porta specifica e può essere distribuito su più server utilizzando un software per il bilanciamento del carico (ad esempio *Nginx*), oppure ampliando la classe che si occupa di orchestrare la

comunicazione con il client, sfruttando lo stesso protocollo utilizzato per la ricezione dell'input utente si possono scambiare informazioni con altre istanze del MAME CGP in esecuzione su altri server per un efficiente bilanciamento del carico. Un'altra strategia di ottimizzazione, sebbene complessa, consiste nel dividere le varie fasi del cloud gaming in componenti e distribuirle dinamicamente su più server in base alle risorse richieste e disponibili.

A valle di tutto quello di cui si è parlato finora si ritiene che gli obiettivi iniziali della tesi siano stati raggiunti e che il MAME CGP sia in grado di fungere da piattaforma di cloud gaming. Infine, il fatto che gli sviluppi possibili siano molteplici induce a pensare che il sistema abbia ampi margini evolutivi e che questo lavoro possa stimolare altre persone ad una maggiore innovazione sui sistemi di cloud gaming e sulle tecnologie di streaming in generale.

Appendice A

Manuale utente

In questo appendice viene descritta la procedura di configurazione del programma.

A.1 Installazione

Il codice sorgente e i compilati per Fedora e Windows sono ospitati su GitHub al seguente indirizzo: `github.com/mikymaione/MAMEStreamingPlatform`.

A.2 Configurazione

Nella cartella `./roms/` vanno inserite tutte le ROM dei giochi che si vuole rendere disponibili.
Nella cartella `./Streaming/HTML/roms/` vanno inserite le copertine dei giochi disponibili in formato PNG, con dimensione 222×315 px.
Nel file `./Streaming/HTML/roms/list.txt` vanno inserite le informazioni del gioco, nel seguente formato `rom_name;description;other_info` ad esempio:
`sfiii3nr1;Street Fighter III: 3rd Strike;Capcom - 1999`.

A.3 Esecuzione

Per avviare il MAME CGP bisogna eseguire il comando:
`./mame64 -streamingserver -video accel -sound sdl -resolution 640x480@30`
Per Linux e Windows sono forniti nel progetto i due scripts `run.sh` e `run.bat`.

Elenco delle tabelle

Elenco delle figure

Elenco dei listati

Sitografia

Acks, William, Sam LaCroix, France Costrel e Melissa Wood (ago. 2020). *High Score - Ep. 1 - Boom & Bust*.

ALSA team (giu. 2021). *Advanced Linux Sound Architecture (ALSA) project homepage*. alsa-project.org/wiki/Main_Page. Accessed: 2021-06-04.

Amazon (apr. 2021). *Amazon Luna*. amazon.com/luna. Accessed: 2021-04-10.

AMD (gen. 2021). *GCN Architecture*. amd.com/en/technologies/gcn. Accessed: 2021-06-03.

Apple (giu. 2021). *Core Audio*. developer.apple.com/documentation/coreaudio. Accessed: 2021-06-04.

Bankhurst, Adam (ott. 2018). *Microsoft Announces Global Game Streaming Service, Project xCloud, Beta Next Year*. ign.com/articles/2018/10/08/microsoft-announces-global-game-streaming-service-project-xcloud-beta-next-year. Accessed: 2021-05-12.

Beijing Cloud Union (ago. 2018). *Cloud Union*. cloudunion.cn. Accessed: 2021-05-07.

Del Bimbo, Alberto (mag. 2018). *Progettazione e produzione multimediale: Video*. e-l.unifi.it/pluginfile.php/611400/mod_resource/content/1/VIDEO.pdf. Accessed: 2021–06-15.

Dickson, Chris (set. 2016). *The Technology Behind A Low Latency Cloud Gaming Service*. blog.parsec.app/description-of-parsec-technology-b2738dcc3842. Accessed: 2021-06-04.

Electronic Arts (ott. 2018). *Project Atlas*. ea.com/news/announcing-project-atlas. Accessed: 2021-04-09.

FFmpeg Team (mar. 2021). *FFmpeg Documentation*. FFmpeg.org/doxygen/trunk. Accessed: 2021-03-30.

Gaikai (giu. 2012). *Gaikai Open Platform*. gaikai.com/open-platform. Accessed: 2021-05-12.

Google (mar. 2019a). *Google Partners with AMD for Custom Stadia GPU*. stadia.dev/intl/fr_ca/blog/google-partners-with-amd-for-custom-stadia-gpu. Accessed: 2021-05-12.

— (apr. 2021). *Google Stadia*. stadia.google.com. Accessed: 2021-04-10.

— (mar. 2019b). *Welcome to Stadia*. stadia.dev/intl/it_it/blog/welcome-to-stadia. Accessed: 2021-05-12.

Hollister, Sean (dic. 2010). *Gaikai enters closed beta, we get an exclusive first look*. engadget.com/2010-12-02-gaikai-enters-closed-beta-we-get-an-exclusive-first-look.html. Accessed: 2021-05-07.

Intel (mar. 2020). *Intel Quick Sync Video*. intel.com/content/www/us/en/architecture-and-technology/quick-sync-video/quick-sync-video-general.html. Accessed: 2021-06-03.

JP Mangalindan (ott. 2020). *Cloud gaming's history of false starts and promising reboots*. polygon.com/features/2020/10/15/21499273/cloud-gaming-history-onlive-stadia-google. Accessed: 2021-04-09.

Leswing, Kif (set. 2020). *Apple issues new rules for App Store that will impact streaming game services from Google and Microsoft*. cnbc.com/2020/09/11/apple-app-store-new-rules-will-affect-google-stadia-microsoft-xcloud.html. Accessed: 2021-04-20.

MAME Team (ago. 2018). *FAQ:Performance*. wiki.mamedev.org/index.php/FAQ:Performance. Accessed: 2021-05-25.
— (apr. 2021). *The Official Site of the MAME Development Team*. mamedev.org. Accessed: 2021-05-03.
Mauro, David (apr. 2021). *Keypress*. github.com/dmauro/Keypress. Accessed: 2021-04-21.
Microsoft (giu. 2021a). *About WASAPI*. docs.microsoft.com/en-us/windows/win32/coreaudio/wasapi. Accessed: 2021-06-04.
— (giu. 2021b). *Desktop Duplication API*. docs.microsoft.com/en-us/windows/win32/direct3ddxgi/desktop-dup-api. Accessed: 2021-06-04.
— (apr. 2021c). *Xbox Game Pass cloud gaming*. xbox.com/en-US/xbox-game-pass/cloud-gaming. Accessed: 2021-04-09.
Montoro, Alvaro (apr. 2021). *gameController.js*. github.com/alvaromontoro/gamecontroller.js. Accessed: 2021-04-21.
Mozilla (mar. 2021a). *Audio and Video Delivery*. developer.mozilla.org/en-US/docs/Web/Guide/Audio_and_video_delivery. Accessed: 2021-04-16.
— (feb. 2021b). *Canvas API*. developer.mozilla.org/en-US/docs/Web/API/Canvas_API. Accessed: 2021-04-21.
— (giu. 2021c). *Web audio codec guide*. developer.mozilla.org/en-US/docs/Web/Media/Formats/Audio_codecs. Accessed: 2021-06-06.
— (giu. 2021d). *Web video codec guide*. developer.mozilla.org/en-US/docs/Web/Media/Formats/Video_codecs. Accessed: 2021-06-06.
— (mar. 2021e). *WebGL*. developer.mozilla.org/en-US/docs/Web/API/WebGL_API. Accessed: 2021-04-21.
— (feb. 2021f). *WebSocket - Web APIs*. developer.mozilla.org/en-US/docs/Web/API/WebSocket. Accessed: 2021-04-16.
— (giu. 2021g). *Writing WebSocket servers*. developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_servers. Accessed: 2021-06-09.
Newzoo (apr. 2020). *Global Cloud Gaming Report*. newzoo.com/global-cloud-gaming-report. Accessed: 2021-04-09.
Nvidia (apr. 2021). *GeForce Now*. nvidia.com/en-us/geforce-now. Accessed: 2021-04-11.
— (mag. 2020). *NVIDIA Video Codec SDK*. developer.nvidia.com/nvidia-video-codec-sdk. Accessed: 2021-06-03.
Orland, Kyle (set. 2020). *Amazon Luna servers will run Windows games directly on Nvidia T4 GPUs*. arstechnica.com/gaming/2020/09/amazon-luna-supports-existing-windows-games-on-turing-level-gpus. Accessed: 2021-05-13.
Paris, Stefano (mag. 2011). *Reti internet multimediali*. cs.unibg.it/paris/rim/download/06-Mpeg.pdf. Accessed: 2021–06-22.
Pitozzi, Andrea (dic. 2019). *I 4 fallimenti più clamorosi del decennio*. wired.it/economia/business/2019/12/27/fallimenti-decennio. Accessed: 2021-05-16.
Playkey (apr. 2021). *Playkey*. playkey.io. Accessed: 2021-04-19.
Qualcomm (lug. 2020). *Hexagon DSP SDK*. developer.qualcomm.com/hexagon-processor. Accessed: 2021-06-03.
RemoteMyApp (apr. 2021). *Vortex cloud gaming*. vortex.gg. Accessed: 2021-04-19.
SDL Community (dic. 2020). *SDL Wiki*. wiki.libsdl.org. Accessed: 2021-03-29.
Sony (apr. 2021). *PlayStation Now*. playstation.com/en-us/ps-now. Accessed: 2021-04-10.
Szablewski, Dominic (apr. 2021). *JSMpeg - MPEG-1 Video & MP2 Audio Decoder in JavaScript*. github.com/phoboslab/jsmpeg. Accessed: 2021-04-05.
The Moving Picture Experts Group (apr. 2003). *MPEG-1 Video*. mpeg.chiariglione.org/standards/mpeg-1/video. Accessed: 2021-06-21.

Warren, Tom (set. 2020a). *Amazon's Luna game streaming service is powered by Windows and Nvidia GPUs*. theverge.com/2020/9/25/21455610/amazon-luna-game-streaming-windows-nvidia-gpu-servers. Accessed: 2021-05-12.

— (giu. 2020b). *Microsoft to upgrade its xCloud servers to Xbox Series X hardware in 2021*. theverge.com/2020/6/18/21295326/microsoft-project-xcloud-xbox-series-x-servers-hardware-2021. Accessed: 2021-05-12.

Whitwam, Ryan (gen. 2014). *Sony's PlayStation Now uses custom-designed hardware with eight PS3s on a single motherboard*. extremetech.com/gaming/175005-sonys-playstation-now-uses-custom-designed-hardware-with-eight-ps3s-on-a-single-motherboard. Accessed: 2021-05-12.

Wiggins, David P. (apr. 2012). *XVFB*. x.org/releases/X11R7.7/doc/man/man1/Xvfb.1.xhtml. Accessed: 2021-06-04.

Bibliografia

Bielievtsov, Stanislav, Igor Ruban, Kyrylo Smelyakov e Dmytro Sumtsov (dic. 2018). «Network technology for transmission of visual information». In: pp. 104–120.

Câmpeanu, Dinu e Andrei Câmpeanu (giu. 2021). «PEAQ–An Objective Method To Assess The Perceptual Quality of Audio Compressed Files». In:

Chen, Kuan-Ta, Yu-Chun Chang, Hwai-Jung Hsu, De-Yu Chen, Chun-Ying Huang e Cheng-Hsin Hsu (2014). «On the Quality of Service of Cloud Gaming Systems». eng. In: *IEEE transactions on multimedia* 16.2, pp. 480–495. ISSN: 1520-9210.

D'Angelo, Gabriele, Stefano Ferretti e Moreno Marzolla (mag. 2015). «Cloud for Gaming». In: pp. 1–6. DOI: 10.1007/978-3-319-08234-9_39-1.

Domenico, Andrea, Gianluca Perna, Martino Trevisan, Luca Vassio e Danilo Giordano (dic. 2020). *A network analysis on cloud gaming: Stadia, GeForce Now and PSNow*.

Forsyth, David A., Jean Ponce, Soumen Mukherjee e Arup Kumar. Bhattacharjee (2015). *Computer vision: a modern approach*. Pearson Education Limited.

Grigorik, Ilya (2013). *High Performance Browser Networking*. O'Reilly Media, Inc. ISBN: 1449344763.

Horé, Alain e Djemel Ziou (2013). «Is there a relationship between peak-signal-to-noise ratio and structural similarity index measure?» In: *IET image processing* 7.1, pp. 12–24. ISSN: 1751-9659.

Huang, Chun-Ying, Kuan-Ta Chen, De-Yu Chen, Hwai-Jung Hsu e Cheng-Hsin Hsu (gen. 2014). «GamingAnywhere: The first open source cloud gaming system». In: *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)* 10. DOI: 10.1145/2537855.

Jurgem, Ronald K. (1997). *Digital consumer electronics handbook*. McGraw-Hill.

Karachristos, Theofilos, Dimitrios Apostolatos e D. Metafas (set. 2008). «A real-time streaming games-on-demand system». In: pp. 51–56. DOI: 10.1145/1413634.1413648.

Lim, Hyesook, Vincenzo Piuri e Earl E. Swartzlander (2000). «A serial-parallel architecture for two-dimensional discrete cosine and inverse discrete cosine transforms». eng. In: *IEEE transactions on computers* 49.12, pp. 1297–1309. ISSN: 0018-9340.

Maggiorini, Dario, Laura Anna Ripamonti, Giacomo Quadrio, A. Bujari e Daniele Ronzani (giu. 2016). «Network analysis of the Sony Remote Play system». In: pp. 10–13. DOI: 10.1109/ISCC.2016.7543706.

Manzano, Marc, Manuel Urueña, Mirko Suznjevic, Eusebi Calle, José Hernández e Maja Matijasevic (mar. 2014). «Dissecting the protocol and network traffic of the OnLive cloud gaming platform». In: *Multimedia Systems* 20, pp. 1–20. DOI: 10.1007/s00530-014-0370-4.

Mileff, Peter e Judit Dudra (mag. 2012). «Efficient 2D software rendering». In: *Production Systems and Information Engineering* 6, pp. 99–110.

Pazera, Ernest (2003). *Focus on SDL*. Premier Press.

Pocta, Peter e John G Beerends (2015). «Subjective and Objective Assessment of Perceived Audio Quality of Current Digital Audio Broadcasting Systems and Web-Casting Applications». eng. In: *IEEE transactions on broadcasting* 61.3, pp. 407–415. ISSN: 0018-9316.

Popovic, Milica, Dejan Drajic, Philipp Svoboda, N. Nikaein, Srdjan Krco e Markus Laner (gen. 2016). «Latency analysis for M2M and online gaming traffic in an HSPA network». In: 31, pp. 259–277.

Rupp, Markus (2008). *Video and multimedia transmissions over cellular networks: analysis, modelling, and optimization in live 3G mobile networks*. Wiley. Cap. 5, 11. ISBN: 1-282-29168-8.

Salmoria, Nicola (2002). «Il progetto MAME: reverse engineering e macchine da gioco». Tesi di laurea mag. Università degli studi di Siena.

Shang, Xiwu, Guozhong Wang, Haiwu Zhao, Jie Liang, Chengjia Wu e Chang Lin (lug. 2017). «A new combined PSNR for objective video quality assessment». In: pp. 811–816. DOI: 10.1109/ICME.2017.8019494.

Shea, Ryan, Jiangchuan Liu, Edith Ngai e Yong Cui (lug. 2013). «Cloud Gaming: Architecture and Performance». In: *Network, IEEE* 27, pp. 16–21. DOI: 10.1109/MNET.2013.6574660.

Spanias, Andreas (2008). *Audio signal processing and coding*. eng. 1st edition. Wiley-Interscience. ISBN: 1-280-82180-9.

Sun, Hanlin (2019). «Research on Latency Problems and Solutions in Cloud Game». In: *Journal of Physics: Conference Series* 1314, p. 012211. DOI: 10.1088/1742-6596/1314/1/012211.

Suznjevic, Mirko e Maja Homen (feb. 2020). «Use of Cloud Gaming in Education». In: ISBN: 978-1-83880-009-3. DOI: 10.5772/intechopen.91341.

Thomos, N, N.V Boulgouris e M.G Strintzis (2006). «Optimized transmission of JPEG2000 streams over wireless channels». eng. In: *IEEE transactions on image processing* 15.1, pp. 54–67. ISSN: 1057-7149.

Ulovec, K e M Smutny (2018). «Perceived Audio Quality Analysis in Digital Audio Broadcasting Plus System Based on PEAQ». eng. In: *Radioengineering* 27.1, pp. 342–352. ISSN: 1210-2512.

Ringraziamenti

Ringrazio tutti coloro che hanno fatto parte del percorso di laurea magistrale:

- mio padre Giovanni, mia madre Maria e mio fratello Alessandro che mi hanno sempre supportato;
- Laura Antonella (alias Dr. Dino) che con il suo amore ha chiuso il buco che avevo nel petto;
- Laura & Giulia, Simona, Eleonora, Martina, Greta e tutte le altre ragazze del dipartimento di farmacia. Grazie per aver reso divertenti le giornate di studio. C'era sempre il sole in biblioteca;
- i miei giocatori del BawiTeam che tra lacrime, infortuni, risate e gioie sono stati una squadra meravigliosa;
- i miei compagni di corso: Carrarini, Dettori, Iervolino, Lombardi, Bonapace, Paduano, Vannucci, Moriani, Laganà, Zhab'yak per gli esami e i fantastici progetti preparati insieme;
- i professori del dipartimento di informatica, che dai loro insegnamenti ho cercato di trarre il massimo per poterli poi concretamente utilizzare;
- Mario, Fede, Nadia, Giorgio, Giovanni e Mariapina per esserci sempre stati (da oltre 20 anni!);
- Alessandro, Carmen, Claudio, Sba, Marika, Grazia, Emiliana e Kikka, anche se non ci siamo visti spesso siete stati vicini;
- i miei parenti di Treviglio che mi hanno aiutato e ospitato. Grazie per il supporto;
- tutti gli altri, anche se non menzionati, siete nel mio cuore.

Dedico questa tesi a Dr. Dino augurandole grandi successi accademici.

Milano, ottobre 2021

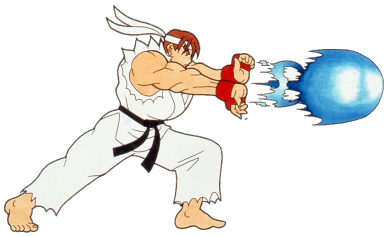

Figura A.1: Street Fighter Alpha artwork. © Capcom