ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Ingegneria e Architettura
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

RL-UniBOt: Applicazione di tecniche di Reinforcement Learning al gioco Mario Kart tramite la piattaforma Gym Retro

Tesi di laurea in
DEEP LEARNING

Relatore
(Prof.) Matteo Ferrara

Candidato
Lorenzo Simoncini

Anno Accademico 2021-2022

Sommario

In questo lavoro di tesi si propone lo studio e l'implementazione di tecniche per la realizzazione di *bot* basati su reti neurali che possano giocare correttamente ai videogiochi della serie *Mario Kart*.

Partendo da un'analisi dettagliata dei giochi e sfruttando diverse tecnologie nel campo del *Deep Learning* vengono presentate alcune soluzioni in grado di giocare a diverse modalità. Inoltre, vengono proposte diverse considerazioni sui risultati ottenuti, evidenziandone pregi e difetti allo scopo di individuare degli sviluppi futuri che potrebbero portare a dei miglioramenti delle prestazioni in gioco.

Ringraziamenti

Approfitto di questo spazio per ringraziare tutti quelli che mi hanno supportato sia nel corso della realizzazione di questo lavoro di tesi che in generale, nella vita.
Grazie ai miei amici del RIG e al Clan Oca, se non sono uscito di testa in questi anni di pandemie e altre difficoltà è molto probabilmente merito vostro.
Grazie alla mia famiglia, per supportarmi sempre nelle mie scelte e per il sostegno ricevuto nel corso di questo percorso.
Infine, grazie al professor Matteo Ferrara per la disponibilità e il supporto mostrati durante tutto questo lavoro di tesi.
A tutti voi, davvero, **Grazie**.

Indice

Elenco delle figure

Elenco delle tabelle

Elenco dei listati

INTRODUZIONE

La serie di videogiochi *Mario Kart* è una dei capisaldi del genere, e fin dall'introduzione del primo capitolo nel 1992 presenta diverse modalità e particolarità interessanti che la distinguono da altri titoli dello stesso genere. Nella maggior parte delle modalità lo scopo è condurre il proprio kart lungo un tracciato per completare un certo numero di giri, generalmente sfidando altri avversari (umani o controllati dal gioco) oppure per cercare di ottenere un buon tempo di completamento.
In letteratura l'utilizzo di tecniche di *Deep Learning* per addestrare *bot* che possano apprendere come giocare ad un videogioco è molto diffuso per titoli estremamente datati, mentre per altri relativamente più recenti ci sono ancora possibilità di approfondire questi argomenti.
L'obiettivo di questo lavoro di tesi è in primo luogo quello di analizzare tecniche già esistenti per il gioco *Super Mario Kart*, per poi applicare le conoscenze ottenute nella realizzazione di un'*Intelligenza Artificiale* in grado di apprendere come giocare correttamente a *Mario Kart: Super Circuit*.

Di seguito viene descritta la divisione in capitoli della tesi, con una breve panoramica dei loro contenuti.
Nel primo capitolo viene fatta una panoramica dello stato dell'arte del *Deep Learning*, prestando particolare attenzione alle categorie e tecniche utilizzate nel lavoro di tesi, con l'obiettivo di fornire al lettore una comprensione generale delle tematiche più importanti presenti all'interno della tesi.
Nel secondo capitolo viene condotta un'accurata analisi del dominio per comprendere al meglio le particolarità dei giochi trattati, dei sistemi in cui essi vengono eseguiti

e delle tecnologie utilizzare per la creazione dei *bot*.

Nel terzo capitolo viene analizzata l'*Integrazione* già esistente di *Super Mario Kart*, per poi descrivere quella realizzata appositamente per *Mario Kart: Super Circuit*.

Nel quarto capitolo vengono presentati i risultati degli addestramenti più rilevanti tra quelli condotti nel corso del lavoro di tesi, mostrando ciò che i *bot* hanno imparato e il modo in cui sono stati addestrati.

Infine sono riportate le conclusioni ricavate da questo lavoro di tesi, insieme a delle proposte su alcuni sviluppi futuri che potrebbero portare a miglioramenti dei modelli proposti.

CAPITOLO 1

STATO DELL'ARTE

L'obiettivo di questo lavoro di tesi è quello di addestrare una intelligenza artificiale (anche detta *AI*, da *Artificial Intelligence*) [13] che sia in grado di giocare correttamente al videogioco Mario Kart Super Circuit per Game Boy Advance. Per raggiungere tale obiettivo, verrano valutate alcune tecniche di *Reinforcement Learning* [14] (in breve RL), una categoria del *Machine Learning* che si occupa di addestrare agenti il cui scopo è massimizzare una reward cumulativa compiendo delle azioni in un ambiente. In questo capitolo verranno presentate le basi del RL e definite alcune delle tecniche più conosciute per l'addestramento di un agente.

1.1 Altre categorie del Machine Learning

Prima di presentare il RL è necessario fare un'introduzione sulle altre due categorie principali del Machine Learning, per comprenderne al meglio le differenze.

1.1.1 Apprendimento Supervisionato

Gli algoritmi di Apprendimento Supervisionato si occupano di costruire un modello matematico a partire da un set di dati (detto *Training Set*) etichettato, ovvero in cui per ogni input è presente l'output desiderato (detto *ground truth* o *label*). L'obiettivo che si vuole raggiungere è la creazione di un modello che sia in grado di predire l'output corretto anche per input che non sono presenti all'interno del training set; queste tecniche possono essere applicate a diverse tipologie di problemi, tra

cui quelli di Classificazione (in cui si assegna l'input ad una specifica classe, scelta da un set limitato) e Regressione (in cui si associa all'input un valore numerico).

1.1.2 Apprendimento Non Supervisionato

A differenza degli algoritmi visti nella sottosezione precedente, quelli di Apprendimento Non Supervisionato lavorano su un training set che contiene solo input, senza il ground truth, e si occupano di trovare una struttura intrinseca ai dati stessi. Per fare ciò, questi algoritmi cercano di trovare delle similarità nei dati che ricevono in input, così da raggrupparli efficacemente in gruppi con caratteristiche simili (detti *Cluster*); sono spesso utilizzati nell'analisi statistica dei dati.

1.2 Reinforcement Learning

Il Reinforcement Learing (anche chiamato Addestramento per Rinforzo) è una categoria del Machine Learning che si occupa di realizzare agenti autonomi che possano eseguire delle azioni in uno specifico ambiente volti a conseguire determinati obiettivi; a differenza dei due tipi di apprendimento visti precedentemente, questo si basa su decisioni sequenziali, in cui l'azione che l'agente va ad eseguire dipende dallo stato dell'ambiente e va ad impattare sullo stato futuro. La qualità delle azioni compiute è data da un valore numerico, detto reward, che viene accumulato dall'agente durante l'addestramento; il suo scopo è quindi massimizzare questo valore.
Questa tipologia di tecniche è ispirata al concetto di rinforzo [15]: secondo la psicologia comportamentista il rinforzo è un evento o uno stimolo (in questo caso lo stato dell'ambiente) che, una volta ricevuto da un organismo (in questo caso l'agente), ne va a modificare il comportamento futuro.
Un agente di RL deve quindi:

- Osservare lo **stato** dell'ambiente
- Compiere un'**azione** in base all'osservazione eseguita

- Aggiornare il proprio **comportamento** in base alla **reward** ricevuta per l'azione appena eseguita

1.2.1 Exploration vs Exploitation

Un concetto cardine alla base del RL è quello dell'*Exploration* (Esplorazione) e dell'*Exploitation* (Sfruttamento/Utilizzo), termini utilizzati per definire il tipo di decisione che viene presa dall'agente; con il primo si indica una volontà da parte dell'agente di "esplorare" nuove strade, compiendo azioni casuali per ottenere nuove informazioni, mentre con il secondo si identifica l'utilizzo da parte dell'agente delle conoscenze ottenute fino a quel momento per scegliere la decisione migliore in base allo stato dell'ambiente.
Bilanciare queste due componenti è una parte spesso fondamentale: un agente che non esplora difficilmente riuscirà ad avere abbastanza informazioni per scegliere una decisione ottima, mentre un agente che non utilizza le informazioni apprese non non farà progressi. Solitamente la probabilità di esplorare nuove opzioni (exploration) è molto elevata all'inizio dell'addestramento, quando l'agente non ha ancora appreso cosa fare, e si riduce nel tempo aumentando la probabilità di prendere decisioni in base alle conoscenze apprese (exploitation); questa probabilità viene denominata *exploration rate*, ed è espressa con un valore numerico che va da zero a uno, dove uno rappresenta il 100% di possibilità di esplorazione.

1.2.2 Tassonomia degli algoritmi di Reinforcement Learning

In questa sottosezione viene presentata una panoramica delle categorie in cui si dividono gli algoritmi di RL; in figura 1.1 è possibile vedere uno schema esemplificativo tratto dalla documentazione di OpenAI [16].

Le due macro-categorie in cui possono essere divisi gli algoritmi sono *Model-Free* e *Model-Based*: questi ultimi hanno a disposizione un modello dell'ambiente, ovvero una funzione che predice le transizioni di stato e le ricompense ottenute. Questo tipo di algoritmi è generalmente più efficace, in quanto può predire quello che succederà

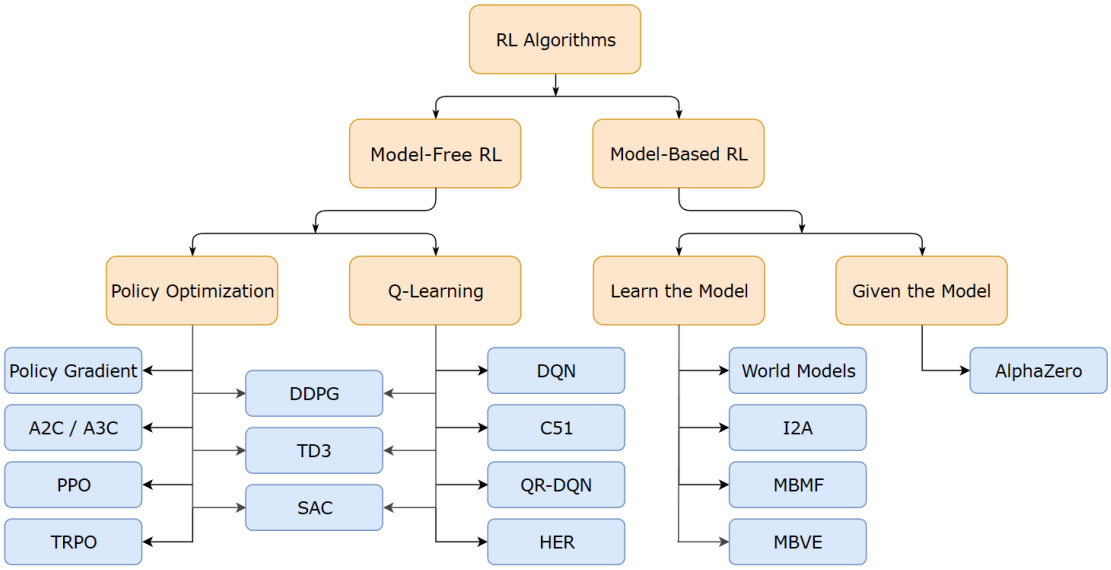

Figura 1.1: La divisione in categorie degli algoritmi di RL

in base alle sue azioni, permettendogli di pianificare le proprie mosse in anticipo; tuttavia, un modello di questo tipo non è sempre disponibile, e anche nel caso in cui se ne volesse creare uno tramite apprendimento ci sono alte probabilità di ottenere un agente che si comporti molto bene rispetto al modello creato ma che fatichi ad ottenere risultati in un ambiente reale. Per questi motivi gli algoritmi *model-free* tendono ad essere i più utilizzati e sviluppati.

Algoritmi model-based

Nel campo degli algoritmi *model-based* i modelli possono essere usati in una moltitudine di modi ortogonali, per cui non esistono gruppi di metodi ben definiti. Di seguito viene fatta una breve lista con alcuni esempi di algoritmi appartenenti a questa categoria:

- **Pianificazione Pura**: uno degli approcci più basilari, in cui non viene mai definita una politica comportamentale in maniera esplicita, utilizzando al suo posto delle tecniche di pianificazione pura (*pure planning*) quali il *model-predictive control* (in breve *MPC*) [23]. Nei *MPC*, ogni volta che l'agente

osserva l'ambiente, esso elabora un piano che descrive quali azioni compiere in una determinata finestra temporale, ottimale rispetto al modello. Dopo aver elaborato il piano l'agente esegue la prima azione descritta, per poi scartarne il resto e proseguire di nuovo con la pianificazione; questo procedimento viene eseguito ad ogni interazione con l'ambiente.

- **Iterazione Esperta**: un metodo costruito sulla base del *pure planning*, in cui si punta a imparare una rappresentazione esplicita della *policy*, $\pi_\theta(a|s)$. L'agente utilizza un algoritmo di pianificazione sul modello, generando un piano di azioni tramite il campionamento della *policy*. L'algoritmo di pianificazione va quindi a produrre un'azione migliore di quella che avrebbe prodotto la politica comportamentale da sola, divenendo quindi l'esperto che dà il nome alla tecnica. In seguito la *policy* viene aggiornata per produrre delle azioni che si avvicinino all'output dell'algoritmo di pianificazione.

- **Incorporazione della pianificazione nelle policy**: un altro approccio possibile consiste nell'integrare la procedura di pianificazione direttamente nella politica comportamentale come *subroutine*, così da rendere i piani elaborati parte delle informazioni disponibili alla *policy*. Il concetto chiave in questo caso è la possibilità per la *policy* di imparare a scegliere come e quando usare i piani a sua disposizione; questo riduce l'impatto del bias del modello, poiché la *policy* può ignorare dei piani che considera "mal pensati" e concentrarsi su quelli utili.

Algoritmi model-free

Gli algoritmi in questa famiglia si dividono in due categorie, che si distinguono in base a ciò che viene imparato dall'agente:

- **Policy Optimization**: i metodi in questa famiglia rappresentano una politica comportamentale (detta anche *policy*) in maniera esplicita come $\pi_\theta(a|s)$ e possono ottimizzare i parametri θ sia direttamente tramite discesa del gradiente sulla performance obiettivo $J(\pi_\theta)$ che indirettamente, massimizzando delle

approssimazioni locali di $J(\pi_\theta)$. Queste approssimazioni sono spesso effettuate *on-policy*, ovvero utilizzando ad ogni aggiornamento solo i dati raccolti seguendo la versione più recente della politica comportamentale.
Due esempi di metodi di *policy optimization* sono **A2C/A3C** (che agiscono direttamente) e **PPO** (che agisce indirettamente massimizzando una funzione obiettivo surrogata che fornisce una stima di quanto $J(\pi_\theta)$ cambierà in seguito all'aggiornamento).

- **Q-Learning**: questi metodi puntano ad apprendere una approssimazione $Q_\theta(s,a)$ della funzione ottima $Q^*(s,a)$, che restituisce la qualità di una certa coppia stato-azione (s,a), detto *Q-Value*; tipicamente fanno uso di una funzione obiettivo basata sull'Equazione di Bellman [17].
  L'ottimizzazione è spesso effettuata *off-policy*, che al contrario di quella *on-policy* vista in precedenza utilizza dati raccolti in qualsiasi momento durante l'addestramento. Una volta appresa la funzione Q l'agente può quindi scegliere l'azione che gli garantirà la reward futura maggiore:

$$a(s) = arg \max_a Q_\theta(s,a)$$

  Uno dei metodi più conosciuti di Q-learning è il **DQN** [20], che verrà approfondito nella prossima sottosezione.

Nell'ambito di questo lavoro di tesi verranno presi in considerazione algoritmi appartenenti a questa categoria, nello specifico **DQN**, poiché non esistono modelli già definiti per l'ambiente (i giochi della serie Mario Kart) che permettono di utilizzare algoritmi model-based.

1.2.3 Deep Q-Learning

Le *Deep Q-Networks* (in breve DQN) sono modelli che fanno uso dei principi del *Q-Learning* visti precedentemente insieme a tecniche di *Deep Learning* [18]; questa pratica è detta *Deep Q-Learning*. Nello specifico, nel *Q-Learning* l'agente

nello scegliere le azioni fa riferimento ad una matrice (detta anche *Q-Table* o *Q-Matrix*) la cui dimensione cresce esponenzialmente in base agli stati e azioni possibili nell'ambiente, rendendone difficile l'uso in problemi complessi; per ovviare a tale problema nella DQN si sostituisce la *Q-Table* con una *Deep Neural Network*, che prende in input lo stato per restituire in output il *Q-Value* delle possibili azioni. In figura 1.2 è possibile vedere una rappresentazione grafica della differenza tra *Q-Learning* e *Deep Q-Learning* (tratta da [19]).

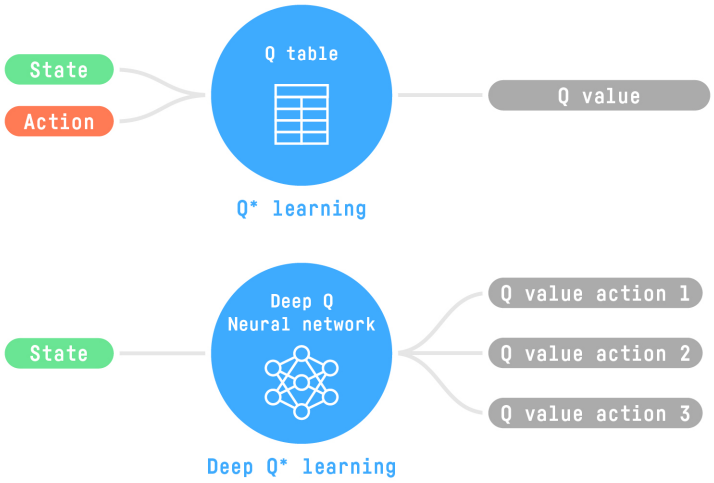

Figura 1.2: Differenza tra Q-Learning e Deep Q-Learning

Il primo modello proposto di DQN [20] fa uso di una *Convolutional Neural Network* [21] per estrarre feature d'interesse dai frame di giochi Atari [25] (immagini RGB [22] di risoluzione 128x160); data l'elevata complessità computazionale che deriva da lavorare con immagini di questo tipo, queste ultime vengono sottoposte a diversi passaggi per ricavarne una rappresentazione in scala di grigi di risoluzione 84x84 (l'altezza e la larghezza del frame vengono uniformati poiché l'implementazione utilizzata per la convoluzione 2D [5] si aspetta degli input quadrati).
Uno dei potenziali problemi nelle DQN è che l'approssimazione dei *Q-Values* tramite l'utilizzo di funzioni non lineari è piuttosto instabile; per ovviare a ciò, viene introdotto l'*experience replay*. Durante l'addestramento tutte le esperienze fatte vengono salvate in una *replay memory*; successivamente, invece di concentrarsi solo su esempi

recenti, vengono selezionati elementi casuali dalla *replay memory*, per impedire che la rete si stabilizzi in un minimo locale.

CAPITOLO 2

ANALISI DEL DOMINIO

In questo capitolo verranno presentati i giochi analizzati insieme alle relative console e le tecnologie utilizzate per lo sviluppo dei modelli e dell'ambiente di addestramento.

2.1 Super Nintendo Entertainment System

Il *Super Nintendo Entertainment System* (in breve SNES, visbile in figura 2.1) è stata la seconda console fissa prodotta dalla Nintendo, rilasciata nel 1990 in Giappone e Corea del Sud, nel 1991 in America del Nord, nel 1992 in Europa e Oceania e nel 1993 nel resto del mondo [1].

Figura 2.1: Il *Super Nintendo Entertainment System*

Il SNES dispone di un controller a dodici tasti: A, B, X, Y, L, R, START, SELECT, LEFT, RIGHT, UP e DOWN. Succedette al precedessore *Nintendo Entertainment System* (in breve NES, visibile in figura 2.2) permettendo lo sviluppo

di nuovi giochi più elaborati grazie al suo processore a 16-bit (contro gli 8 bit del NES); un esempio delle possibilità date dalla potenza maggiore della console sono i giochi di guida: se in precedenza si tendeva a realizzare visuali dall'alto o prospettive limitate (come visibile nell'immagine 2.3) con l'arrivo dello SNES essi hanno potuto vantare di una profondità inedita.

Figura 2.2: Il *Nintendo Entertainment System*

Figura 2.3: Due schermate di due giochi guida per NES, rispettivamente RadRacer a sinistra e SpyHunter a destra

In particolare, una delle svolte più importanti all'interno del genere è stata data da F-Zero (di cui è visibile una schermata di gioco in figura 2.4), che grazie all'utilizzo della funzionalità *Mode7* del SNES [2], che permette di ruotare e manipolare sprite di gioco in maniera complessa, simula un ambiente tridimensionale su cui si svolge la corsa, aumentando sia il senso di immersione che la complessità con la quale è possibile realizzare i tracciati.

Figura 2.4: Una schermata di gioco di F-Zero

In seguito, partendo dall'idea di realizzare un seguito di F-Zero che includesse anche una modalità per due giocatori, nel 1992 venne rilasciato **Super Mario Kart**, che divenne poi il capostipite di una serie che al momento della scrittura conta nove capitoli.

2.2 Super Mario Kart

Super Mario Kart è il primo gioco della serie Mario Kart, composta da giochi di guida con protagonisti i personaggi della serie Mario [3]. In questa sottosezione vengono presentati i diversi elementi del gioco e le varie modalità che lo compongono.

2.2.1 Termini di gioco

Nel corso del gioco, a prescindere dalla modalità, il giocatore prende il controllo di un personaggio alla guida di un kart e ha il compito di concludere cinque giri in uno o più percorsi; di seguito vengono definiti i termini relativi agli elementi fondamentali del gioco, molti dei quali accompagnati da un riferimento visivo in figura 2.5.

Figura 2.5: Una schermata di gioco di Super Mario Kart

- **Kart Giocatore**: Il kart controllato dal giocatore (numero 1 in figura 2.3); questo è guidato da un personaggio scelto tra otto diversi, e questa scelta influenza diversi parametri nella guida.
- **Salto**: Un'azione che può essere compiuta dal Kart giocatore; consiste nel fare saltare il kart mantenendo la sua inerzia.
- **Derapata**: Un'azione che può essere compiuta dal Kart giocatore; permette di effettuare le curve senza perdere velocità.
- **Vite**: Il numero di vite rimanenti (2); presente solo in alcune modalità.
- **Monete**: Il numero di monete raccolte (3); presente solo in alcune modalità.
- **Posizione**: La posizione attuale nella corsa (4); presente solo in alcune modalità.
- **Cubo Oggetto**: Passandoci sopra con il proprio kart è possibile raccogliere un oggetto; presente solo in alcune modalità.
- **Kart Avversari**: Gli avversari della gara corrente (6); presenti solo in alcune modalità.

- **Slot Oggetto**: Mostra l'oggetto posseduto al momento (7); presente solo in alcune modalità.
- **Timer**: Mostra quanto tempo è passato dall'inizio della corsa (8).
- **Tracciato/Percorso**: Il luogo su cui si svolge la gara; può essere di diversi tipi e a seconda di essi è composta da diversi tipi di terreno.
- **Mappa del tracciato**: La mappa del tracciato corrente, mostra la conformazione della strada e gli eventuali avversari (9).
- **Corsa/Gara**: L'attività che eseguono i kart sul tracciato, può avere obiettivi diversi in base alla modalità.
- **Giro**: Indica a quale punto della corsa si trova il giocatore; aumenta di uno ogni volta che viene percorso il tracciato nella sua interezza.

2.2.2 Regole e modalità

Super Mario Kart presenta diverse modalità di gioco in base al numero di giocatori (massimo due contemporaneamente).

Modalità ad un giocatore

Quando si seleziona l'opzione **"1P GAME"** (Giocatore singolo) sono disponibili le seguenti modalità di gioco:

- **Mariokart GP**: In questa modalità si seleziona un personaggio e si affrontano sette avversari in un campionato formato da cinque gare su percorsi diversi, con l'obiettivo di avere più punti al termine delle cinque gare; è disponibile in versione 50cc e 100cc, dove la seconda presenta una velocità maggiore. In questa modalità si possono trovare lungo la strada delle monete (che una volta raccolte vanno accumularsi contribuendo ad aumentare la velocità del kart) e dei cubi oggetto (che conferiscono al giocatore un oggetto speciale da utilizzare durante la gara); inoltre, si hanno a disposizione delle vite che

vengono consumate se si sceglie di riprovare una gara o se si conclude una corsa dalla quinta all'ottava posizione.

- **Time Trial**: In questa modalità si scelgono un personaggio e un percorso e si cerca di fare cinque giri nel minor tempo possibile; non sono presenti né monete né oggetti.

Modalità a due giocatori

Quando si seleziona l'opzione **"2P GAME"** (Due giocatori) sono disponibili tre diverse modalità di gioco:

- **Mariokart GP**: Una versione analoga della modalità omologa per un giocatore, dove però ogni giocatore controlla un kart.
- **Match Race**: Una sfida uno contro uno su un percorso a scelta, vince chi completa per primo cinque giri; in alcuni punti del tracciato possono essere presenti degli ostacoli.
- **Battle Mode**: Una battaglia che si svolge su un'arena scelta tra quattro disponibili, in cui ogni giocatore ha tre palloncini che gli orbitano attorno e vince se fa scoppiare tutti quelli dell'avversario utilizzando gli oggetti raccolti nell'arena.

Nell'ambito di questo progetto, nel caso di Super Mario kart, ci si è concentrati sulla modalità Time Trial, andando quindi ad escludere tutte le componenti non rilevanti come gli altri giocatori o gli oggetti speciali.

2.2.3 Comandi

Nella tabella 2.1 vengono mostrati i diversi pulsanti utilizzabili durante una corsa e le azioni ad essi corrispondenti.

Tasto	**Azione**
A	Utilizzo dell'oggetto
B	Acceleratore
X, SELECT	Scambio tra la visione posteriore e la visione della mappa (solo un giocatore)
Y	Freno
LEFT, RIGHT	Sterzo
L, R	Salto/Derapata se combinato con una direzione
START	Pausa

Tabella 2.1: I comandi del gioco Super Mario Kart

2.3 Game Boy Advance

Il *Game Boy Advance* (in breve GBA, visibile in figura 2.6) è stata la terza console portatile prodotta da Nintendo, dopo *Game Boy* e *Game Boy Color* (visibili in figura 2.7); è stata rilasciata nel 2001 in quasi tutte le regioni del mondo, e poteva vantare un processore a 32-bit, il doppio di cui era provvisto il SNES.

Figura 2.6: Il *Game Boy Advance*

Figura 2.7: Il *Game Boy* e il *Game Boy Color*

Rispetto al SNES ha una risoluzione di output leggermente minore (240x160 contro 256x224) ma vanta una rosa di colori visualizzabili contemporaneamente molto più alta (512 contro 256), permettendo una resa grafica complessivamente più appagante; inoltre, può utilizzare tutte le trasformazioni di sprite rese possibili dalla *Mode7* del SNES. Dispone di dieci tasti, due in meno rispetto allo SNES: A, B, L, R, START, SELECT, LEFT, RIGHT, UP e DOWN.

2.4 Mario Kart Super Circuit

Mario Kart Super Circuit (conosciuto in Giappone come Mario Kart Advance) è il terzo gioco della serie, rilasciato nel corso del 2001 in Giappone, Americhe e Europa. È il primo gioco della serie ad approdare su di una console portatile, e a differenza del predecessore Mario Kart 64 ritorna ad avere una grafica 2D, mostrandosi simile a Super Mario Kart; nonostante le similarità, la resa grafica complessiva può dirsi notevolmente migliorata, grazie ad elementi quali le texture a più alta risoluzione, il maggior numero di colori rappresentabili a schermo e gli sfondi complessi a tre livelli di parallasse [4].

Figura 2.8: Una schermata di gioco di Mario Kart Super Circuit

2.4.1 Modalità di gioco

Il gioco può essere giocato sia in modalità a giocatore singolo che in multi-giocatore (fino a quattro contemporaneamente); come per Super Mario Kart, in base al numero di giocatori sono disponibili diverse modalità.

Modalità ad un giocatore

Nella modalità a giocatore singolo sono disponibili tre diverse opzioni:

- **Mario GP**: disponibile in tre diverse cilindrate (50cc, 100cc e 150cc) che regolano la velocità massima del kart e la difficoltà degli avversari; come per la modalità quasi omonima del primo gioco della serie qui si seleziona un personaggio tra quelli disponibili e un campionato (composto da quattro gare di tre giri ciascuna), con l'obiettivo di ottenere più punti alla fine delle quattro gare disputate. All'inizio sono presenti quattro campionati, e proseguendo nel gioco se ne possono sbloccare altri sei. Lungo il tracciato è possibile trovare monete e cubi oggetto dal funzionamento analogo a quello che avevano nel gioco precedente.

- **Time Trial**: l'obiettivo di questa modalità è completare tre giri in un tracciato a scelta nel minor tempo possibile; all'inizio della gara si hanno a disposizione tre Funghi turbo da utilizzare come si preferisce nel corso della prova. I tempi migliori vengono salvati e possono poi essere scambiati con altri giocatori.

- **Quick Run**: in questa modalità si sceglie una cilindrata, un personaggio e un percorso con i quali eseguire una gara singola secondo le stesse regole del Mario GP.

Modalità a più giocatori

Sono disponibili tre modalità per più giocatori:

- **Mario GP**: come nella modalità a giocatore singolo omonima si sceglie un personaggio e una cilindrata per gareggiare in un campionato, affrontando sei avversari controllati dalla cpu e un altro giocatore.

- **Versus**: in questa modalità fino a quattro giocatori possono competere in una gara su un percorso a scelta, senza la presenza di avversari controllati dalla CPU.

- **Battle**: fino a quattro giocatori possono sfidarsi in una battaglia in un'arena apposita in cui il loro obiettivo è utilizzare gli oggetti raccolti per colpire gli avversari e fare scoppiare dei palloncini posizionati sopra le loro teste. Quando un giocatore esaurisce i tre palloncini a propria disposizione viene eliminato, e l'ultimo a rimanere in gioco viene dichiarato vincitore.

Nell'ambito di questo progetto ci si è concentrati sulle modalità a giocatore singolo Time Trial e Quick Run.

Comandi

Nella tabella 2.2 vengono mostrati i diversi pulsanti utilizzabili durante una corsa e le azioni ad essi corrispondenti.

Tasto	**Azione**
A	Acceleratore
B	Freno
B + DOWN	Retromarcia
LEFT, RIGHT	Sterzo
R	Salto/Derapata se combinato con una direzione
L	Utilizzo dell'oggetto
START	Pausa
SELECT	Clacson

Tabella 2.2: I comandi del gioco Mario Kart Super Circuit

2.4.2 Differenze principali da Super Mario Kart

Pur presentando una struttura di gioco simile al capitolo per SNES il gioco presenta alcune differenze degne di nota, soprattutto per quanto riguarda la fisica dei kart notevolmente rivisitata e la presenza della meccanica del Miniturbo; quest'ultima garantisce un piccolo scatto d'accelerazione dopo aver eseguito una derapata per un certo ammontare di tempo, incrementando l'utilità della manovra e aggiungendo un elemento in più da tenere in considerazione per migliorare i propri tempi. Inoltre, un elemento importante nella modalità Time Trial è l'aggiunta di tre Funghi turbo da utilizzare durante la gara: oltre a garantire un bonus di velocità, il loro utilizzo può permettere di attraversare senza ripercussioni terreni accidentali quali erba o terra, creando delle scorciatoie utili ad accorciare le tempistiche.
Un altro elemento introdotto in questo gioco è la partenza turbo, visibile in figura 2.9: iniziando a tenere premuto il tasto dell'acceleratore al momento giusto durante il contro alla rovescia iniziale (precisamente a metà tra l'accensione della seconda luce e l'accensione della terza) il kart partirà con una notevole accelerazione iniziale, come se avesse usato un Fungo turbo; tuttavia, accelerando prima del previsto è possibile subire un malus che rallenta la partenza facendo sbandare il kart.

Figura 2.9: La partenza turbo in Mario Kart Super Circuit

2.5 Gym Retro

Gym Retro [6] è una piattaforma dedita al reinforcement learning sui videogiochi, basata su Gym [7], una libreria open source sviluppata da OpenAI [8] che fornisce diverse API standard per gestire algoritmi ed environment dedicati al reinforcement learning.
Gym retro supporta diverse console e i relativi giochi, in particolare:

- NES - Nintendo Entertainment System
- SNES - Super Nintendo Entertainment System
- Nintendo Game Boy
- Sega Genesis
- Sega Master System
- Sega Game Gear
- Nintendo Game Boy Color
- Nintendo Game Boy Advance

- NEC TurboGrafx

Tramite Gym Retro è possibile definire parametri, funzioni di reward e altre caratteristiche specifiche per ogni videogioco; l'insieme di questi elementi è chiamato integrazione. Di seguito vengono presentati gli elementi principali coinvolti nell'apprendimento di un determinato gioco.

2.5.1 Agente

Un **Agente** è definito come un'entità che agisce (compiendo delle **Azioni**) all'interno di un **Environment** per cercare di massimizzare la **Reward** ottenuta; a tale scopo riceve a degli intervalli periodici una **Observation** dell'Environment, che utilizza per aggiornare la sua **Policy** di comportamento. In figura 2.10 è possibile vedere un grafico che mostra in maniera semplice questo ciclo [9].

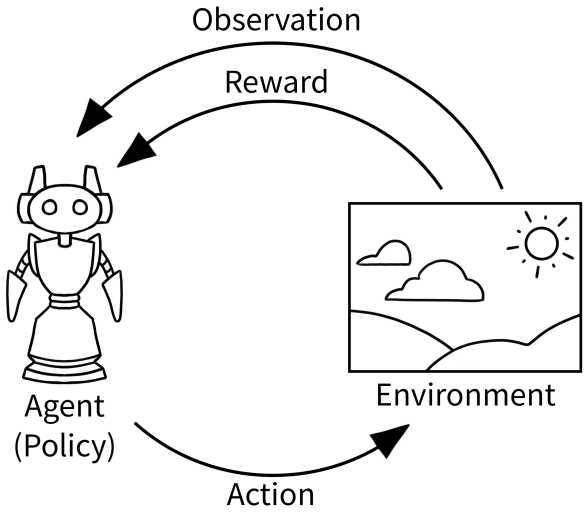

Figura 2.10: Il ciclo di apprendimento di un Agente

2.5.2 Observation

Con Observation si definisce l'insieme di informazioni che l'Agente riceve periodicamente dall'Environment; nel caso di un Agente che cerca di imparare a giocare ad un videogioco, un'Observation può essere ad esempio un frame di gioco in un determinato momento o un insieme di valori memorizzati all'interno del gioco stesso.

Gym Retro supporta due tipi di Observation: *IMAGE*, ossia le immagini di gioco, eventualmente opportunamente modificate per favorire l'individuazione di elementi rilevanti, e *RAM*, i valori salvati nella memoria di gioco.

2.5.3 Reward

La Reward è la "ricompensa" che l'Agente ottiene in base alle azioni che compie nell'Environment; lo scopo di un Agente è proprio quello di massimizzare la Reward ottenuta. In giochi semplici un esempio di Reward può essere il punteggio di gioco, che generalmente è collegato alle azioni positive che possono essere compiute all'interno dello stesso; in caso il punteggio non sia presente o l'obiettivo del gioco sia più complesso è necessario definire una Reward che tenga conto degli elementi e delle azioni che sono importanti al fine di raggiungere l'obiettivo. Nel caso di un gioco di guida la Reward può essere definita in funzione del tracciato percorso o del tempo impiegato a concludere la corsa.

2.5.4 Definizione dello stato iniziale

Gym Retro permette di definire lo stato iniziale dell'addestramento attraverso il caricamento di un *save state*, ovvero un salvataggio dello stato di gioco effettuato in un determinato momento; caricando uno specifico save state, è possibile concentrare l'addestramento nei momenti che si preferiscono, evitandone altri irrilevanti ai fini di ciò che si vuole far apprendere all'Agente. Ad esempio, nei giochi della serie Mario Kart, un buon save state potrebbe essere posizionato poco prima dell'inizio di una gara, per saltare le parti relative alla navigazione dei menù e concentrarsi solamente nell'apprendimento della corsa.

2.6 Tool di integrazione

Scaricando Gym Retro vengono fornite diverse integrazioni per alcuni giochi, la cui completezza varia da gioco a gioco; qualora il videogioco d'interesse non abbia un'integrazione o quest'ultima non sia del tutto completa OpenAI fornisce un tool

apposito con il quale è possibile realizzare un'integrazione personalizzata. Tramite esso le componenti che costituiscono l'integrazione di uno specifico gioco vengono salvate all'interno di file appositi, denominati Data (contenente informazioni sulle variabili), Scenario (contenente informazioni che definiscono lo scenario nel quale si conduce l'addestramento) e Metadata (che contiene informazioni aggiuntive); il contenuto di questi file verrà approfondito nel capitolo successivo.

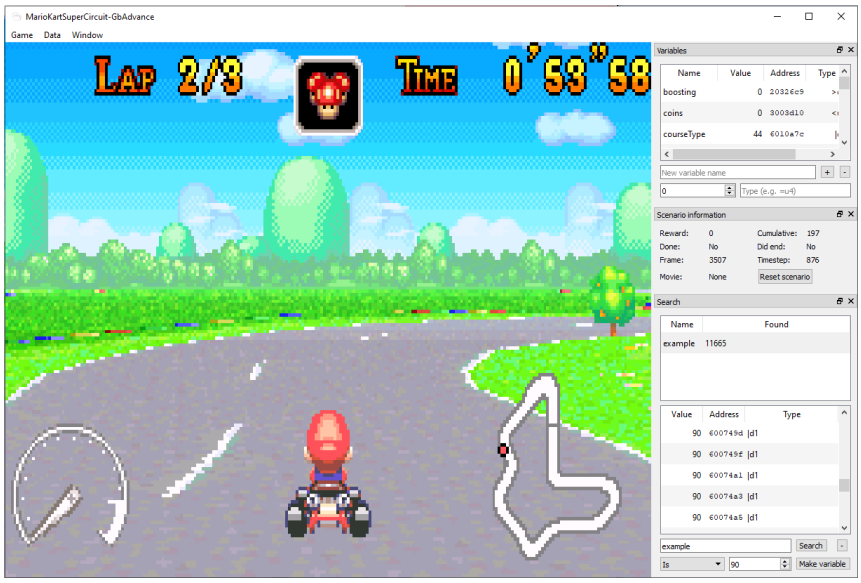

Figura 2.11: La schermata principale del tool di integrazione

In figura 2.11 è possibile vedere la schermata principale del tool di integrazione; sulla sinistra viene riprodotto il gioco caricato, mentre sulla destra sono presenti diverse informazioni e strumenti utili all'integrazione. Nello specifico:

- **Tab Variables**: mostra le diverse variabili create, in base al contenuto del file Data; in figura sono presenti quattro variabili, di cui vengono mostrati il valore, il nome, l'indirizzo e il tipo (questi ultimi tre valori possono essere modificati in qualsiasi momento dal tool). Il tipo di dato è composto dalla combinazione di un termine che specifica l'ordine dei byte, uno che definisce il formato e un ultimo termine che indica il numero di byte occupati in memoria dalla variabile (preferibilmente un multiplo di due); in tabella 2.3 è possibile

vedere i diversi termini utilizzabili per specificare l'orfdine dei byte, mentre in tabella 2.4 vengono mostrati i possibili formati.

Termine	**Ordine**
<	Little Endian
>	Big Endian
><	Middle Endian (big/little)
<>	Middle Endian (little big)
=	Nativo
>=	Middle (big/nativo)
<=	Middle (little/nativo)
|	Non specificato (utile solo per i valori a singolo byte)

Tabella 2.3: I possibili termini che indicano l'ordine dei byte

Termine	**Tipo**
i	Signed
u	Unsigned
d	Binary-coded Decimal
n	Low-nybble Binary-coded Decimal

Tabella 2.4: I termini utlizzabili per specificare il formato della variabile

- **Tab Scenario Information**: mostra varie informazioni sullo scenario e permette di resettarlo tramite la pressione di un tasto. Le informazioni principali che presenta sono:
  - Reward: La reward ottenuta al frame attuale
  - Cumulative: Il valore di reward accumulato finora
  - Done: Indica se la condizione "Done" dello scenario è stata soddisfatta
  - Frame: Il numero di frame trascorsi dall'avvio del gioco

  - Movie: Mostra se il gioco sta venendo riprodotto in maniera standard (None), se sta venendo registrato (Recording) o se sta venendo riprodotta una registrazione (Playing)

- **Tab Search**: Permette di ricercare nuove variabili e di visualizzare i risultati delle query di ricerca. La tab è divisa principalmente in tre parti:

  - La prima in alto in cui vengono visualizzate le variabili ricercate, mostrandone il nome e il numero di occorrenze trovate
  - La seconda viene popolata cliccando su una delle variabili mostrate nella parte vista in precedenza, e mostra il valore, l'indirizzo e il tipo di dato delle varie occorrenze trovate
  - L'ultima parte è quella che permette di effettuare la ricerca, scegliendo un nome per la variabile da ricercare, impostando la query di ricerca (Is, Increased By, Decreased By, Increased, Decreased, Changed, Unchanged) e l'eventuale valore di comparazione (in tabella 2.5 è possibile vedere l'effetto di queste query nel dettaglio); i tre tasti a lato permettono di avviare la ricerca, creare una nuova variabile da una delle occorrenze trovate e rimuovere dalla lista i risultati della ricerca. In figura 2.11 si può vedere il risultato della ricerca di una variabile denominata "example" utilizzando la query "Is 90"; la ricerca ha prodotto 11665 risultati, visibili nella parte centrale della tab.

Query	**Parametro**	**Effetto**
Is	Sì	Ricerca un indirizzo il cui valore è pari al parametro.
Increased By	Sì	Ricerca un indirizzo il cui valore è aumentato di un ammontare pari al parametro.
Decreased By	Sì	Ricerca un indirizzo il cui valore è diminuito di un ammontare pari al parametro.
Increased	No	Ricerca un indirizzo il cui valore è aumentato di un ammontare qualsiasi.
Decreased	No	Ricerca un indirizzo il cui valore è diminuito di un ammontare qualsiasi.
Changed	No	Ricerca un indirizzo il cui valore è cambiato dall'ultima ricerca.
Unchanged	No	Ricerca un indirizzo il cui valore è rimasto immutato dall'ultima ricerca.

Tabella 2.5: L'effetto delle query di ricerca disponibili

2.7 Stable Baselines 3

Per la realizzazione degli agenti è stato utilizzato *Stable Baselines 3* [30], un set di implementazioni affidabili di algoritmi di Reinforcement Learning realizzati in *PyTorch* [31]. Le caratteristiche principali di SB3 sono:

- Struttura unificata per tutti gli algoritmi, con codice pulito e uniformato allo stile **PEP 8** [32]
- Funzioni e classi ben documentate
- Prestazioni degli algoritmi testate per garantirne l'efficacia
- Integrazione di *TensorBoard* [33] per la valutazione e analisi degli addestramenti

Nello specifico, ci si è concentrati sull'implementazione della DQN presente all'interno di SB3 [24].

CAPITOLO 3

INTEGRAZIONE

Per iniziare a lavorare su un gioco tramite Gym Retro è necessario che sia presente la sua integrazione, ovvero una serie di file che contengono informazioni utili agli algoritmi di reinforcement learning per cercare di apprendere il comportamento desiderato. Nello specifico una integrazione è generalmente composta da:

- file **Data**: un file che contiene le posizioni nella memoria del gioco di diverse variabili utili per comporre la funzione di reward, come ad esempio le vite del personaggio o il punteggio; in alcuni casi sono reperibili in rete delle *ram map* [10] che possono contenere tutti o alcuni dei valori di interesse, mentre in altri va fatta una ricerca da zero utilizzando l'apposito tool di integrazione.

- file **Scenario**: questo file contiene diverse informazioni che definiscono lo "scenario" nel quale si conduce l'addestramento; ad esempio, qui si possono definire la dimensione dello schermo, il numero e il tipo di possibili input e il modo in cui le variabili contribuiscono a comporre la reward.

- file **Metadata**: un file che può contenere diverse informazioni utili in fase di addestramento, come ad esempio lo stato di default del gioco quando viene caricato.

- uno o più file contenenti **Script** utili per realizzare funzioni di reward più complesse; queste funzioni possono poi essere richiamate all'interno del file Scenario.

Come detto nella sezione 3.6, alcuni giochi per console più datate dispongono già di alcune integrazioni all'interno dei file forniti con Gym Retro, mentre altri relativamente più recenti ne sono sprovvisti. Nel caso dei giochi in oggetto di questo lavoro di tesi né Super Mario Kart né Mario Kart Super Circuit dispongono di un'integrazione ufficiale; tuttavia, il gioco per SNES dispone di un'integrazione realizzata da terzi reperibile su Github [11]. Nella prima parte di questo lavoro di tesi è stata analizzata questa integrazione per comprendere quale tipo di variabili fosse possibile creare per questo tipo di giochi, per poi realizzare un'integrazione personalizzata per Mario Kart Super Circuit.

3.1 Analisi dell'integrazione di Super Mario Kart

I punti principali dell'integrazione di Super Mario Kart sono le variabili create, contenute nel file Data, e le funzioni contenute nel file *script.lua*.

3.1.1 Variabili

All'interno del file Data sono state definite trenta variabili diverse, e di seguito verranno elencate quelle che sono ritenute le più importanti ai fini dell'addestramento:

- **coins**: il numero di monete raccolte.
- **course**: il tracciato nel quale si sta giocando.
- **currMilliSec, currMin, currSec**: indicano rispettivamente i millisecondi, minuti e secondi nelle modalità in cui è presente un timer; possono essere combinati per ottenere il tempo trascorso dall'inizio della gara.
- **current_checkpoint**: il checkpoint nel quale si trova al momento il kart controllato dal giocatore.
- **getFrame**: il numero di frame susseguitisi dall'inizio della gara.

- **getGameMode**: indica la modalità di gioco, utile per eseguire controlli diversi in base al tipo di obiettivo.
- **isTurnedAround**: indica se il kart controllato dal giocatore è rivolto nel senso di marcia contrario.
- **item_box**: mostra quale oggetto è stato raccolto dal giocatore; nella modalità Time Trial il suo valore rimane sempre a zero.
- **kart1_speed**: la velocità attuale del kart controllato dal giocatore.
- **lap**: indica il giro attuale in cui si trova il giocatore; ha come valore iniziale 127.
- **rank**: la posizione del kart controllato dal giocatore in una gara, ha valore zero per la prima posizione e sette per l'ottava (l'ultima).
- **surface**: il terreno su cui si trova il giocatore.
- **totalCheckpoints**: il numero totale di checkpoint di cui è composto il tracciato sul quale si sta svolgendo la corsa.

Tutte queste variabili possono essere utili per avere informazioni sullo stato del kart e sulla sua posizione nel tracciato durante la gara; inoltre, la presenza delle variabili **current_checkpoint** e **totalCheckpoints** mostra come ogni tracciato del gioco sia diviso in sezioni, denominate checkpoint. Il numero di checkpoint in cui è diviso un percorso può variare a seconda del tracciato, ed è indicato dalla variabile **totalCheckpoints**; in figura 3.1 è possibile vedere il modo in cui il tracciato Mario Circuit è diviso in checkpoint.

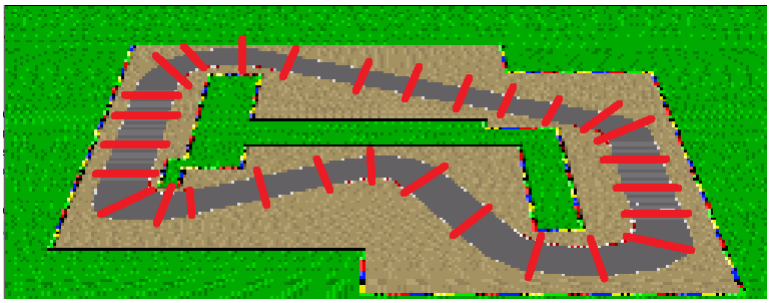

Figura 3.1: La divisione in checkpoint del percorso Mario Circuit

3.1.2 Script

Nell'integrazione di Super Mario Kart è presente un file di script le cui funzioni vengono utilizzate per calcolare la Reward ottenuta durante l'addestramento e per stabilire la fine di uno scenario. Nello specifico, lo scenario viene concluso quando il kart completa tutti i giri della gara (portando il valore della variabile lap a 133) o quando si scontra contro un muro o una parete della pista più di quattro volte nell'arco di 500 step di gioco (ogni step equivale ad un frame, quindi 500 step corrispondono circa ad otto secondi, dato che il gioco ha un frame rate di 60FPS [26]); questi controlli vengono effettuati solo nel caso della modalità **Time Trial**, l'unica presa in considerazione in questa integrazione.
La Reward viene invece calcolata in base al numero di checkpoint attraversati dal kart, conferendo un ammontare positivo quando viene raggiunta una nuova sezione della pista e conferendo un ammontare negativo quando si torna al checkpoint precedente; inoltre, viene conferita una penalità quando il giocatore urta contro un muro, cade fuori dalla pista o cade in acque profonde (nelle piste che permettono queste azioni). Questa Reward presenta dei limiti, in quanto non tiene conto in alcun modo del tempo impiegato dal kart per oltrepassare i checkpoint, e inoltre ignora la tipologia di terreno su cui è posizionato il giocatore, non considerando quindi l'impatto che esso ha sulla velocità del kart.

3.2 Integrazione di Mario Kart: Super Circuit

In questa sezione viene presentata l'integrazione per Mario Kart: Super Circuit, realizzata da zero in questo lavoro di tesi poiché non presente tra quelle disponibili in Gym Retro. Al termine di questo capitolo verrà valutata la possibilità di mettere a disposizione della comunità l'integrazione realizzata tramite Github.

3.2.1 Variabili

Il primo passo nel realizzare l'integrazione è la creazione delle variabili, andando a considerare quali possono essere quelle importanti per l'addestramento (sia nel caso specifico di questo progetto che per addestramenti più generali) e i valori che esse possono assumere.
Il primo passo di questo procedimento è stato selezionare alcune delle variabili presenti nell'integrazione di Super Mario Kart e verificare se fossero presenti dei corrispettivi simili anche per questo gioco; un esempio è la variabile **lap**, che indica il giro attuale della corsa. Nel gioco per SNES questa variabile ha come valore iniziale 127, e aumenta di uno ad ogni superamento della linea del traguardo (fino ad un massimo di 133, dato che le gare sono composte da cinque giri); per questo motivo per trovare una variabile equivalente è stato ricercato un indirizzo di memoria che contenesse un valore che aumentasse ad ogni giro completato. Nello specifico, sono stati compiuti i seguenti passaggi all'interno del tool di integrazione (visto nella sezione 3.6):

- Caricamento del gioco con un save state posizionato all'inizio di una prova a tempo

- Ricerca di una variabile di nome "lap", con query "Unchanged"; questo è il primo passo da fare per ogni variabile e ci mostra tutti gli indirizzi e i rispettivi valori contenuti (figura 3.2)

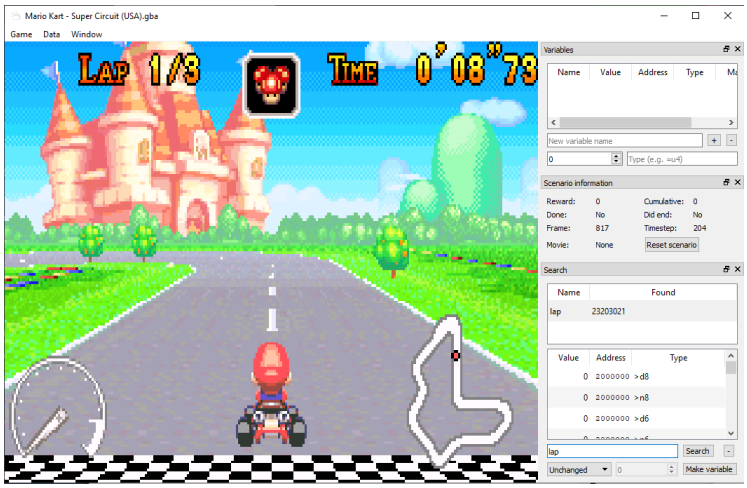

Figura 3.2: Ricerca della variabile "lap" con query "Unchanged"

- Ricerca con la query "Increased" dopo aver finito il primo giro; in questo caso il risultato atteso è che la variabile sia aumentata di uno o più in seguito all'evento (figura 3.3)

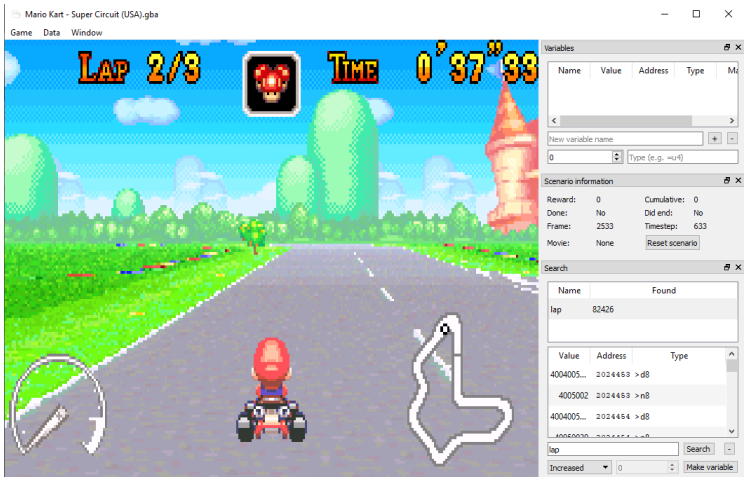

Figura 3.3: Ricerca della variabile "lap" con query "Increased" durante il secondo giro

- Ripetizione del passaggio due durante il terzo giro per sfoltire il numero di risultati (figura 3.4)

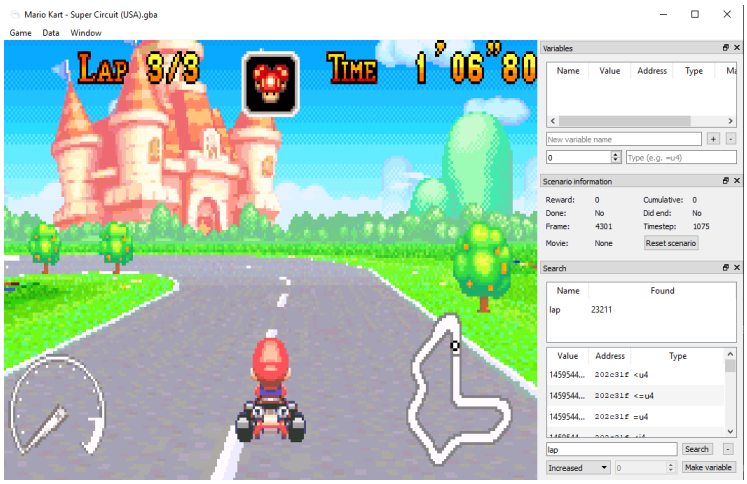

Figura 3.4: Ricerca della variabile "lap" con query "Increased" durante il terzo giro

- Diverse ricerche con query "Unchanged" in diverse condizioni (ad esempio avanzando nel tracciato, cambiando il tipo di terreno e invertendo il senso di marcia) per ridurre il numero di risultati (figura 3.5)

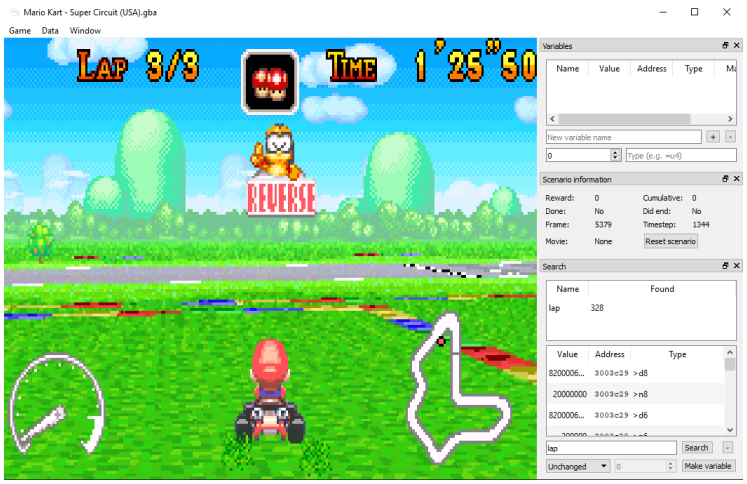

Figura 3.5: Riduzione del numero dei risultati tramite ricerche con query "Unchanged" in diverse condizioni

- Reset dello stato del gioco e ricerca con query "Decreased" e successivamente una ripetizione dei passaggi precedenti per ridurre ulteriormente il numero di risultati (figura 3.6)

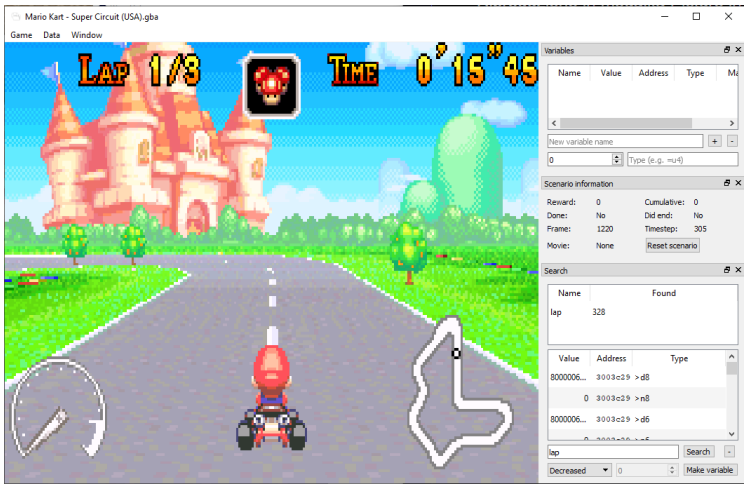

Figura 3.6: Ricerca della variabile "lap" con query "Decreased" dopo il reset dello stato di gioco

- Infine sono stati controllati gli indirizzi trovati per valutarne il contenuto e come esso cambiasse in relazione agli eventi in gioco, per trovare poi l'indirizzo corrispondente alla variabile desiderata (evidenziato in figura 3.7)

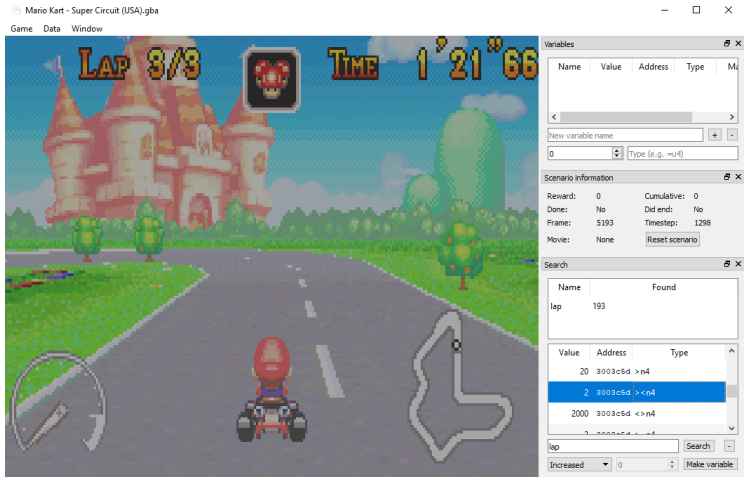

Figura 3.7: L'indirizzo corrispondente alla variabile desiderata

Durante questo procedimento è necessario prestare particolare attenzione ad indirizzi diversi che contengono valori che cambiano in maniera simile in relazione agli eventi; ad esempio, nel caso di **lap**, è stato scelto l'indirizzo 0x3003c5d, il cui valore parte da zero e sale di uno ad ogni superamento della linea del traguardo, ma durante

la ricerca ci si è imbattuti nell'indirizzo 0x203ec58, il cui valore parte da 1 e aumenta ogni volta che viene superato il primo checkpoint del giro attuale. Per verificare la correttezza dell'indirizzo scelto in alcuni casi è possibile utilizzare l'emulatore VisualBoyAdvance [12]; utilizzando la funzione di ricerca dei cheat si possono ottenere i valori contenuti nei vari indirizzi di memoria e definire dei "cheat" che vanno a modificarli. Se il cambiamento del valore comporta una modifica a schermo è possibile verificare di aver scelto l'indirizzo giusto creando un cheat che va ad aumentare o diminuire il valore in oggetto e controllando la zona dello schermo interessata; nell'esempio della variabile "lap", andando a modificare il valore all'indirizzo 0x3003c5d da 80000000 a 82000000 mentre ci si trova nel primo giro il contatore nell'angolo in alto a sinistra dello schermo mostrerà 3/3 (come visibile in figura 3.8)

Figura 3.8: Il contatore dei giri prima e dopo la modifica tramite emulatore

Un altro esempio è la ricerca della variabile **isGoingBackwards**:

- Come prima cosa, esattamente come nel caso precedente, viene caricato il gioco con il save state corretto
- Successivamente si ricercano tutti gli indirizzi disponibili con la query "Unchanged"
- Nel terzo passaggio si effettua una ricerca con query "Changed" dopo aver cambiato senso di marcia (come si può notare dall'immagine 3.9)

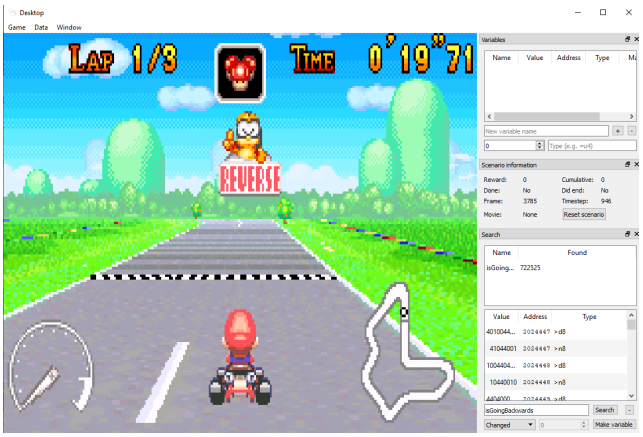

Figura 3.9: Un passaggio del processo di ricerca della variabile isGoingBackwards

- Si procede poi a ripetere passaggi precedenti cambiando le condizioni per sfoltire i risultati, controllando a mano gli indirizzi rimanenti.

Una componente fondamentale per l'addestramento è l'individuazione dei checkpoint, che permettono con relativa facilità di comprendere la progressione dell'agente nel corso della gara. Dopo aver eseguito la ricerca in maniera simile a quanto fatto per le due variabili mostrate in precedenza, sono state individuate due variabili relative a questi elementi: **cp** e **cp_change**; la prima mostra il checkpoint in cui si trova il giocatore, la seconda mostra il momento effettivo in cui il kart supera una sezione per entrare in quella successiva. A differenza di Super Mario Kart, in questo gioco i checkpoint nella singola pista tendono ad essere molti meno, e il valore della variabile relativa scala di quattro ad ogni sezione superata, non più di uno.

In figura 3.10 è possibile vedere la divisione in sezioni della prima pista del gioco, Peach Circuit; queste sono solamente otto, considerevolmente meno di quelle in cui era diviso il primo tracciato del gioco per SNES (come visto in precedenza in figura 3.1).

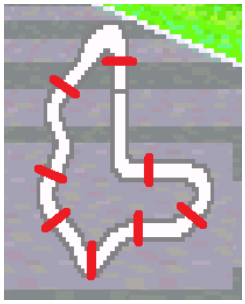

Figura 3.10: La divisione in checkpoint del tracciato Peach Circuit

Un elemento da tenere a mente durante la ricerca delle variabili è che alcuni indirizzi possono avere comportamenti differenti a seconda del tracciato; quando si esegue questa operazione quindi è necessario provare le query su più tipi di circuito per assicurarsi che il valore all'indirizzo scelto cambi allo stesso modo indipendentemente dalla pista.
Nel listato 3.1 è possibile vedere le variabili create (con il loro nome, indirizzo e tipo, inserite poi all'interno del file *Data.json*.

```
{
  "info": {
    "boosting": {
      "address": 33760969,
      "type": ">d8"
    },
    "coins": {
      "address": 50347280,
      "type": "<u2"
    },
    "courseType": {
      "address": 100731516,
```

```
      "type": "|d1"
    },
    "cp": {
      "address": 50340812,
      "type": "|u1"
    },
    "cp_change": {
      "address": 50340816,
      "type": "|u1"
    },
    "hit": {
      "address": 50345328,
      "type": "|u1"
    },
    "isGoingBackwards": {
      "address": 50347069,
      "type": ">u1"
    },
    "itemSlot": {
      "address": 100666978,
      "type": "|u1"
    },
    "lap": {
      "address": 50347101,
      "type": "><n4"
    },
    "place": {
      "address": 50340788,
      "type": "=u4"
    },
    "race_state": {
      "address": 117440525,
      "type": "|u1"
    },
    "speed": {
      "address": 50347094,
```

```
49        "type": "|u2"
50      },
51      "terrainType": {
52        "address": 50346988,
53        "type": "<u2"
54      },
55      "time": {
56        "address": 50355328,
57        "type": "|u2"
58      },
59      "track": {
60        "address": 33573734,
61        "type": "|u1"
62      },
63      "turbo": {
64        "address": 33761433,
65        "type": ">n8"
66      }
67    }
68  }
```

Listing 3.1: Variabili create per Mario Kart: Super Circuit

Nel dettaglio, le variabili sono:

- **boosting**: Indica se il kart del giocatore è in stato di Turbo, generato dall'uso di un Fungo turbo o da una partenza con il tempismo perfetto; assume il valore 80 se lo è, 0 altrimenti
- **coins**: Il numero di monete raccolte durante una gara
- **courseType**: Indica il tipo di tracciato, che può essere associato ad uno o più percorsi. In tabella 3.1 vengono mostrati i diversi valori che può assumere.
- **cp**: Il checkpoint in cui si trova il giocatore; parte da zero e aumenta il valore di 4 ad ogni checkpoint superato (il valore massimo che può assumere cambia a seconda del percorso).

Valore	**Tipo tracciato**
44	Peach Circuit
4	Beach, Cheap Cheap Island
32	Riverside park
11	Bowser Castle
98	Mario Circuit
0	Boo Lake, Sky Garden
56	Cheese land
34	Luigi Circuit
30	Sunset Wilds
20	Snow Land
18	Ribbon Road
22	Yoshi Desert

Tabella 3.1: I valori possibili della variabile courseType

- **cpchange**: Cambia valore nel momento in cui si supera un checkpoint

- **hit**: Indica se il kart ha urtato un ostacolo, assume valore 1 in caso positivo, 0 altrimenti

- **isGoingBackwards**: Assume valore 16 o 24 se il kart sta guidando nella direzione sbagliata, 0 od 8 altrimenti

- **itemSlot**: Mostra quale oggetto è contenuto nello slot oggetti; in tabella 3.2 vengono mostrati i diversi oggetti possibili e il valore che assume la variabile di conseguenza

- **lap**: Indica il numero del giro in cui si trova attualmente il giocatore; parte da zero e viene incrementato di uno ad ogni superamento della linea di traguardo.

Valore	**Oggetto**
255	Slot Vuoto
169	Fungo singolo
153	Fungo Doppio
90	Fungo Triplo
203	Superstella
220	Fulmine
247	Buccia di banana
66	Boo
18	Guscio Verde/Rosso
17	Gucio Verde/Rosso Triplo
204	Guscio Blu

Tabella 3.2: I valori possibili della variabile itemSlot

- **place**: La posizione del giocatore; in una prova a tempo ha sempre valore 0, mentre in una gara varia da 0 (corrispondente al primo posto) a 7 (corrispondente all'ottavo)

- **racestate**: Lo stato della corsa, può assumere valore 113 o 115 a seconda se ci si trova rispettivamente in una gara o una prova a tempo in corso, valore 117 per una prova a tempo finita e 4 al termine di una gara con altri avversari; inoltre, assume valore 81 nel caso durante una gara il giocatore si trovi in ultima posizione con tutti gli avversari che hanno concluso la corsa.

- **speed**: La velocità del kart del giocare; parte da zero per arrivare ad un massimo di 2016

- **terrainType**: il tipo di terreno su cui si trova il giocatore; può assumere lo stesso valore per diversi tipi di terreno a seconda del tipo di tracciato. Inoltre, mostra quando il kart esegue le azioni di Salto, Freno o Derapata. In tabella 3.3 vengono mostrati i possibili valori che può assumere e il tipo di terreno

corrispondente; con "standard" si intende un tipo di terreno che non comporta malus negativi sulla guida e che costituisce la parte principale del tracciato, generalmente al centro (con l'eccezione del terreno "Ghiaccio standard", che seppur essendo effettivamente quello di cui è composta la maggior parte della pista rende il kart più difficile da controllare, facendolo scivolare).

Valore	**Tipo Terreno**	**Effetto**
0	Acqua bassa, Fango, Neve alta	Rallentamento del kart
2	Strada standard, Rete metallica, Strada standard alternativa (Sunset Wilds), Strada standard scura (Cheese Land)	Nessun effetto
3	Spiaggia standard, Deserto standard, Strada standard chiara (Cheese Land)	Nessun effetto
3	Terrreno accidentato (Bowser Castle 1/3, Luigi Circuit, Sky Garden, Sunset Wilds, Ribbon Road)	Rallentamento del kart
4	Erba alta, Terreno accidentato (Cheese Land)	Rallentamento del kart
4	Strada standard marrone (Bowser Castle), Strada standard (Sunset Wilds)	Nessun effetto
5	Ghiaccio standard (Snow Land)	Attrito ridotto
6	Terreno accidentato (Yoshi Desert)	Rallentamento del kart
7	Salto, Freno, Derapata	Effetto dell'azione corrispondente

Tabella 3.3: I diversi tipi di terreno di cui possono essere composti i tracciati

- **time**: Il tempo trascorso dall'inizio della corsa/prova a tempo; le due cifre meno significative indicano i centesimi di secondo, mentre le più significative

i secondi trascorsi (ad esempio per un tempo di un minuto, due secondi e ottantotto centesimi avrà valore 6288)

- **track**: il tracciato attuale, utile a capire quale sia il valore massimo che può assumere la variabile cp (legata al numero di checkpoint), visto che in questo gioco non è presente un indirizzo in cui viene salvato questo dato. In tabella 3.4 è possibile vedere tutti i percorsi disponibili all'inizio del gioco insieme al valore massimo che può assumere la variabile cp e il relativo valore della variabile Track.

Valore Track	**Nome Tracciato**	**Massimo valore cp**
35	Peach Circuit	28
11	Shy Guy Beach	12
23	Riverside Park	48
73	Bowser Castle 1	28
44	Mario Circuit	16
63	Boo Lake	40
44	Cheese Land	36
27	Bowser Castle 2	28
37	Luigi Circuit	36
26	Sky Garden	44
70	Cheep-Cheep Island	40
69	Sunset Wilds	36
4	Snow Land	40
36	Ribbon Road	52
12	Yoshi Desert	40
51	Bowser Castle 3	48

Tabella 3.4: Il valore massimo che può assumere la variabile cp in ogni percorso

- **turbo**: Indica se il kart è in stato di Miniturbo (generato da una derapata,

come descritto in sezione 2.4.2), e in caso positivo assume un valore compreso tra 1 e 5

3.2.2 Script

Nella realizzazione del file contenente gli script si è presa come base il file già presente nell'integrazione di Super Mario Kart; in questa sezione verranno presentati i metodi principali e la costruzione della Reward.

Scenario

Lo scenario viene definito concluso quando si verifica una delle seguenti condizioni:

- il kart controllato dal giocatore termina con successo la gara o la prova a tempo (controllando il valore della variabile **race_state**)
- il kart si scontra per più di sette volte contro un ostacolo nel corso di ottocento step
- il timer del gioco arriva a tre minuti senza che si verifichi una delle due condizioni riportate nei punti precedenti

Nel listato 3.2 si può vedere parte del metodo *getWallPenalty()* che si occupa anche di eseguire il controllo sul numero di urti eseguiti dal kart entro un certo periodo di tempo; di particolare interesse sono il controllo sulla diversità del valore della variabile **hit** allo step precedente (per evitare di considerare più volte lo stesso urto) e il controllo sulla variabile **boosting** a riga 2: quest'ultimo è necessario poiché lo stato di accelerazione dovuto ad una partenza col tempismo perfetto o all'utilizzo di un Fungo turbo va a modificare lo stesso indirizzo di memoria in cui è salvata la variabile **hit**, creando il rischio di avere dei falsi positivi nel rilevamento degli urti.

```
if data.hit ~= oldHit then
  if data.hit == 1 and data.boosting == 0 and data.speed < 600 then
    wallHits = wallHits + 1
    if wallHits >= 7 then
      earlyStop = true
    end
    reward = -1
  end
end
if wallTimer > 800 then
  wallTimer = 0
  wallHits = 0
end
```

Listing 3.2: Parte del metodo *getWallPenalty()*

Reward

La Reward ottenuta dall'agente ad ogni step viene definita come la combinazione di diverse componenti, alcune positive (corrispondenti ad azioni che contribuiscono al raggiungimento dell'obiettivo di gioco) e altre negative (che corrispondono ad azioni che ostacolano o rallentano il raggiungimento dell'obiettivo). Ogni componente è rappresentata da un valore booleano (true se la componente si è verificata o false altrimenti) e da un peso, che va a stabilire se la componente è positiva o negativa (segno) e l'importanza della componente stessa nella Reward totale (valore assoluto); definite quindi per un certo stato s le componenti come c_i e i pesi come w_i la Reward complessiva r è definibile come:

$$r = \sum_{i=0}^{N} c_i(s) * w_i$$

Di seguito sono presentate le diverse componenti che compongono la Reward e le azioni che esse rappresentano:

- **Componente cp**: la componente più importante per stabilire il progresso dell'agente, a cui è associato un peso molto alto che varia a seconda del numero

di checkpoint di cui è costituito un tracciato. Questa componente presenta un valore positivo solamente al superamento di un nuovo checkpoint, per evitare che l'agente attraversi continuamente lo stesso punto; a tale scopo, ogni sezione è identificata da un valore che tiene conto sia del checkpoint stesso che del giro in cui si trova il giocatore (come visibile nel listato 3.3). La variabile **lapsize** viene inizializzata all'inizio della sessione di allenamento in base al valore massimo assumibile dalla variabile **cp** nel tracciato in oggetto.

```
1 if data.lastCheckpoint == checkpoint + (lap -1)*lapsize then
2     return false
3 end
4 if checkpoint + (lap)*lapsize > data.lastCheckpoint then
5     data.lastCheckpoint = checkpoint + (lap)*lapsize
6     return true
7 else
8     return false
9 end
```

Listing 3.3: Estratto del metodo *checkpointPassed()* in cui viene controllato il superamento di un checkpoint

- **Componente wall**: una componente negativa che viene applicata ogni volta che il kart urta contro un muro, un ostacolo o un avversario. Come visto in precedenza, troppi urti consecutivi possono portare alla fine prematura di uno scenario.

- **Componente terrain**: una componente negativa che si applica quando il kart percorre un terreno accidentato. Per fare ciò viene controllato il tipo di terreno e il tipo di tracciato tramite le variabili di riferimento (**terrainType** e **courseType**); inoltre, viene controllato se il kart del giocatore è in stato di Turbo (relativo all'uso di un Fungo Turbo), in quanto essere in questo stato previene il rallentamento che comporta normalmente il passaggio su di un terreno accidentato. Nel listato 3.4 è possibile vedere il metodo che si occupa di fare questi controlli, *isSafeTerrain()*.

```
function isSafeTerrain()
  if data.boosting == 80 and data.speed > 700 then
    return true
  end
  if data.terrainType == 2 or data.terrainType == 7 then
    return true
  elseif (data.courseType == 4 or data.courseType == 22 or
        data.courseType == 56) and data.terrainType == 3 then
    return true
  elseif (data.courseType == 30 or data.courseType == 11)
        and data.terrainType == 4 then
    return true
  elseif data.courseType == 20 and data.terrainType == 5 then
    return true
  end
  return false
end
```

Listing 3.4: Il metodo *isSafeTerrain()*

- **Componente backwards**: componente negativa che viene applicata quando il giocatore guida nel senso opposto di marcia; viene applicata rica dieci volte al secondo.

- **Componente time**: componente negativa applicata ogni cinquanta step di gioco. Lo scopo di questa componente è favorire un tempo di completamento della gara minore, in quanto grazie ad essa a parità di checkpoint superati uno scenario con tempo di completamento minore avrà ricevuto meno penalità, e avrà quindi una Reward complessiva maggiore. Nel listato 3.5 è possibile vedere il metodo *getTimePenalty()*, che si occupa di restituire in output questa componente.

```
function getTimePenalty()
  if data.time ~= 0 and (data.race_state == 115
    or data.race_state == 113 or data.race_state == 81) then
    if data.time % 50 == 0 then
      return 1
    end
  end
  return 0
end
```

Listing 3.5: Il metodo *getTimePenalty()*

- **Componente speed**: componente negativa che ha il compito di sfavorire un comportamento statico o un'accelerazione incostante da parte dell'agente; se il giocatore si muove ad una velocità inferiore alla soglia fissata viene applicata una penalità che va ad impattare sulla Reward complessiva finale. Nel listato 3.6 è possibile vedere il metodo *getSpeedPenalty()*, che si occupa di fare questo controllo.

```
function getSpeedPenalty()
    if data.time ~= 0 and data.time % 50 == 0 then
        if data.speed > 1080 then
            return 0
        else
            return 1
        end
    end
    return 0
end
```

Listing 3.6: Il metodo *getSpeedPenalty()*

- **Componente start**: una componente negativa introdotta per disincentivare l'agente ad utilizzare un Fungo Turbo per compensare il malus negativo dato da una partenza errata. Come visibile nel listato 3.7, se il kart utilizza uno dei

Funghi turbo che ha in dotazione all'inizio della corsa nei primi otto secondi riceve una penalità, pertanto per massimizzare la Reward ottenuta il comportamento migliore che può apprendere è quello di non sbagliare la partenza, così da non ricevere alcun malus alla velocità e non creare la necessità di utilizzare l'oggetto.

È possibile estendere questa penalità all'utilizzo di altri oggetti in contesti diversi, ma nell'ambito di questo lavoro di tesi ci si è limitati a gestire questa specifica situazione.

```
function getStartPenalty()
  if not mushroomUsed then
    if data.time < 900 and data.itemSlot ~= 90 then
      mushroomUsed = true
      return 1
    end
  end
  return 0
end
```

Listing 3.7: Il metodo *getStartPenalty()*

3.2.3 Caricamento dell'integrazione

Allo scopo di facilitare sviluppi futuri e nuovi addestramenti si è deciso di caricare l'integrazione realizzata su GitHub, al seguente link: `https://github.com/LorenzoSimoncini2/MarioKartSuperCircuit-GbAdvance`.

L'integrazione caricata contiene i file Data e Scenario, alcuni save state e i principali file di script realizzati nel corso del lavoro di tesi.

CAPITOLO 4

MODELLI E ADDESTRAMENTI

In questo capitolo verranno presentati gli addestramenti effettuati e i risultati ottenuti, oltre alle operazioni preliminari effettuate sull'ambiente di addestramento. La rete utilizzata è l'implementazione della DQN presente all'interno di *Stable Baselines 3* [24], utilizzando come CPU una *AMD EPYC* [28] di seconda generazione (16 core ad una frequenza di 2.8 GHz) e come GPU una *NVIDIA QUADRO RTX 5000* [29] (16 GB di memoria GDDR6).

4.1 Preparazione dell'ambiente

Prima di procedere con l'addestramento, è necessario definire un *Gym Retro Environment*, l'ambiente con il quale interagirà l'agente durante l'apprendimento. Per fare ciò è sufficiente richiamare il metodo *retro.make()*, passandogli in input il nome del gioco (come è definito nell'integrazione di Gym Retro realizzata all'interno di questo lavoro di tesi, quindi "MarioKartSuperCircuit-GbAdvance") ed eventuali altri parametri quali lo scenario, il save state da caricare e la volontà o meno di salvare una registrazione delle azioni dell'agente; quest'ultima viene salvata come la sequenza di tasti premuti dall'agente, riducendo così notevolmente la sua occupazione in memoria.
Dopo aver creato l'ambiente esso può essere modificato tramite l'applicazione di alcuni metodi definiti *wrapper*, che possono ad esempio modificare l'osservazione che

viene restituita in output oppure il tipo di input che riceve il gioco. Di seguito verranno presentati in ordine i wrapper utilizzati:

- **MKSCDiscretizer**: questo wrapper si occupa di cambiare il tipo degli input che vengono passati al gioco, trasformandoli da *MultiBinary* (una lista contenente ogni possibile azione, in cui una qualsiasi combinazione di esse può essere utilizzata ad ogni step di gioco) a *Discrete* (una lista delle azioni possibili, di cui una sola utilizzabile per ogni step di gioco). Questa è un'operazione necessaria poiché l'implementazione della DQN presente all'interno di *Stable Baselines 3* supporta solo operazioni del secondo tipo. Per fare ciò, si scelgono a priori le combinazioni di pulsanti che verranno rese possibili all'interno del gioco, per poi creare un vettore di possibili azioni sotto forma di un array di booleani, in cui ogni elemento rappresenta la pressione (True) o meno (False) del pulsante corrispondente. Il metodo presenta un parametro (*bostAndWait*) per scegliere se permettere o meno le azioni di attesa e utilizzo dell'oggetto; queste due azioni sono disattivabili per consentire un addestramento facilitato che non comporta conseguenze troppo gravi durante la corsa. Nel listato 4.1 è possibile vedere la classe wrapper *MKSCDiscretizer*.

```
class MKSCDiscretizer(gym.ActionWrapper):
  def __init__(self, env, boostAndWait = False):
      super(MKSCDiscretizer, self).__init__(env)
      buttons = env.unwrapped.buttons
      if boostAndWait:
        actions = [["A"], ["LEFT", "A"], ["RIGHT", "A"], ["LEFT",
    "A", "R"], ["RIGHT", "A", "R"], ["A", "R"],["A", "L"],[]]
      else:
        actions = [["A"], ["LEFT", "A"], ["RIGHT", "A"], ["LEFT",
    "A", "R"], ["RIGHT", "A", "R"], ["A", "R"]]
      self._actions = []
      for action in actions:
          arr = np.array([False] * 12)
          for button in action:
              arr[buttons.index(button)] = True
```

```
14              self._actions.append(arr)
15          self.action_space = gym.spaces.Discrete(len(self._actions))
16      def action(self, a):
17          return self._actions[a].copy()
18
```

Listing 4.1: Il metodo MKSCDiscretizer

Le combinazioni di tasti e le corrispondenti azioni permesse sono:

- **(A)**: Accelerazione senza sterzo
- **(LEFT + A), (RIGHT + A)**: Accelerazione con sterzo a destra o sinistra
- **(LEFT + A + R), (RIGHT + A + R)**: Derapata verso destra o verso sinistra
- **(A + R)**: Accelerazione con salto
- **(A+L)**: Accelerazione e uso dell'oggetto
- **(Nessuna pressione)**: Azione di attesa, nessuna conseguenza nel gioco

Ad eccezione dell'ultima azione (che può comunque essere esclusa dalle azioni possibili con il parametro *boostAndWait* visto in precedenza) le altre condividono il fatto di tenere sempre premuto l'acceleratore; questo perché con uno stile di gioco ideale si predilige una costante accelerazione, per mantenere il più possibile la velocità massima. Per fare ciò è importante la manovra della derapata, che permette di affrontare curve complesse senza perdere velocità. Per lo stesso motivo non è stato inserito il tasto relativo alla frenata: imparando a giocare in modo corretto l'agente non dovrebbe avere il bisogno né di frenare né di andare in retromarcia. L'azione di attesa è stata inserita per poter permettere all'agente di effettuare una buona partenza (o nel migliore dei casi una partenza Turbo); mantenendo un'accelerazione costante, il kart sbaglierebbe sempre la partenza e subirebbe un malus che comporta un notevole rallentamento nei primi secondi.

- **WarpFrame**: uno dei wrapper forniti all'interno di Gym Retro, si trova all'interno del package *atari_wrappers*; si occupa di trasformare l'osservazione dell'ambiente da un'immagine del frame di gioco RGB ad una in scala di grigi e ne riduce la dimensione. Lo scaling predefinito trasforma l'immagine in input ad una dimensione 84x84 (pensata per i giochi Atari [25]), mentre nell'ambito di questo progetto si è scelta una dimensione 100x100; questa decisione è stata presa poiché ad una dimensione 84x84 la mappa del percorso presente in basso a destra dello schermo risulta di difficile comprensione, causando una perdita di informazioni possibilmente utili (in figura 4.1 è possibile vedere un confronto tra la mappa di un tracciato nelle due differenti dimensioni).

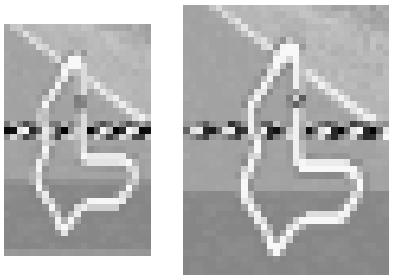

Figura 4.1: La mappa di un tracciato in un frame a risoluzione 84x84 (a sinistra) e 100x100 (a destra)

- **MaxAndSkipEnv**: un altro dei wrapper presenti nel package *atari_wrappers*, restituisce all'agente un'osservazione modificata dell'ambiente ogni quattro step di gioco; l'osservazione passata all'agente è il risultato dell'operazione *max* tra gli ultimi due frame trascorsi (in figura 4.2 si può vedere un esempio grafico di questa operazione su un gioco Atari, tratta da [27]; i frame di gioco contrassegnati da una "X" vengono saltati completamente, su quelli con bordo giallo viene applicata l'operazione *max*).

  Questo wrapper è utilizzato per velocizzare l'apprendimento dell'agente e per evitare che l'agente modifichi la sua azione troppo spesso. In figura 4.3 sono visibili i diversi risultati dei passaggi di preprocessing dell'immagine.

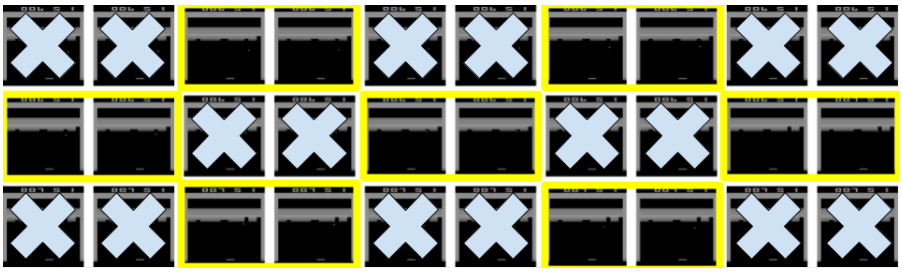

Figura 4.2: Una sequenza di frame di un gioco Atari

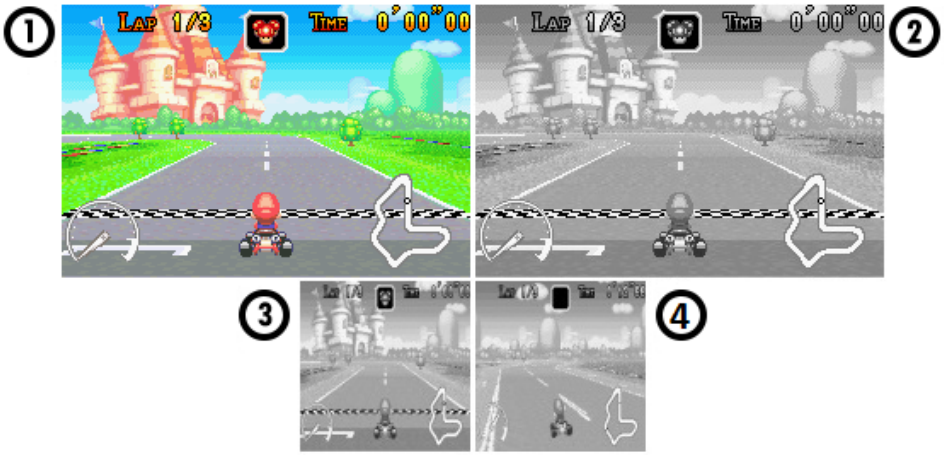

Figura 4.3: I risultati dei diversi passaggi di preprocessing dell'immagine: frame originale RGB (1), conversione da RGB a scala di grigi (2), scaling a dimensione 100x100 (3) e operazione max di due frame consecutivi (4)

4.2 Time Trial

In questa sezione vengono presentati i risultati ottenuti nella modalità Time Trial, utilizzando come personaggio Mario nel tracciato Peach Circuit.
L'obiettivo di questa modalità è concludere tre giri del percorso nel minor tempo possibile, pertanto la scelta dei pesi delle componenti della funzione Reward punta a massimizzare questo traguardo.

4.2.1 Addestramenti con comandi limitati

Per prima cosa sono stati condotti degli addestramenti con comandi limitati, ovvero escludendo l'azione di attesa e l'uso di oggetti; questo è stato fatto per

puntare ad ottenere come risultato un agente che fosse in grado di concludere la prova a tempo mantenendo un'accelerazione continua, ignorando per il momento la questione della partenza errata e dell'utilizzo dei Funghi turbo durante la prova. In tabella 4.1 è possibile vedere i pesi utilizzati.

Componente reward	**Valore**
wall_c	-10
cp_c	400
terrain_c	-15
time_c	-0.5
speed_c	-0.5
backwards_c	-2
start_c	0

Tabella 4.1: I pesi utilizzati negli addestramenti con comandi limitati

Come è possibile vedere nella tabella 4.1, si è scelto di dare un peso molto elevato alla componente **cp**, relativa al superamento dei checkpoint, allo scopo di incentivare per prima cosa il completamento della prova; il valore 400 è specifico per questo percorso e cambia in modo inversamente proporzionale rispetto al numero di sezioni (definito come *lapsize*) in cui è diviso un tracciato tramite la seguente formula (realizzata a partire dal valore fisso di 400 testato sul percorso Peach Circuit):

$$cp\_c(lapsize) = \lfloor 11600/lapsize \rfloor$$

Un'altra componente degna di nota è quella **backwards**, posta a -2 per disincentivare fortemente la guida contromano; considerando che questa componente viene applicata circa dieci volte al secondo (cinque volte in più rispetto a **time_c** e **terrain_c**) anche un valore minore rappresenta una penalità considerevole. La differenza nella frequenza di applicazione è data dal modo in cui è gestito il riconoscimento della guida contromano all'interno del gioco (finché il kart non si ferma essa non viene rilevata, rendendo necessario controllarla più di frequente): negli sviluppi

futuri le frequenze di applicazione delle componenti possono essere uniformate, per rendere più chiaro il valore dei pesi).

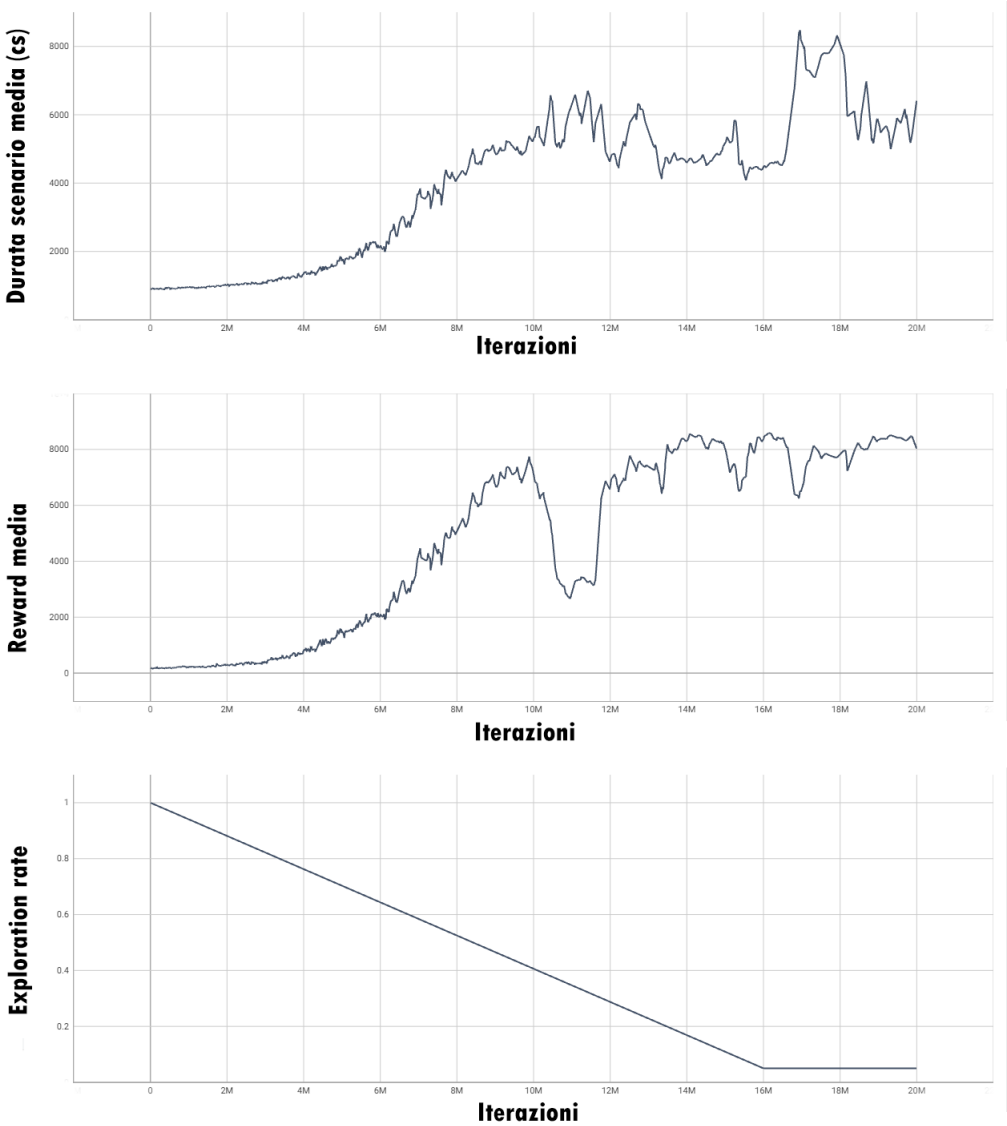

Figura 4.4: I grafici relativi all'addestramento migliore con comandi limitati

Il migliore addestramento con queste impostazioni conta venti milioni di iterazioni, svolte in circa 56 ore; in figura 4.4 è possibile vedere tre grafici che mostrano rispettivamente l'andamento della durata media di uno scenario negli ultimi quattro episodi (espresso in centesimi di secondo), la Reward media e l'andamento

dell'*exploration rate* nel corso delle iterazioni. Dai grafici si può notare come nella parte iniziale la durata e la Reward ottenuta aumentano quasi di pari passo, poiché deve trascorrere il tempo necessario per concludere la prova; successivamente, il rapporto fra i due valori cambia e non si nota più una correlazione diretta.
Nel corso dell'addestramento i pesi dell'agente vengono salvati sia periodicamente (ogni milione di iterazioni) sia quando viene ottenuta una nuova Reward migliore; le considerazioni fatte su questo ed altri agenti sono sempre relative alla versione migliore dell'agente. Durante questo addestramento l'agente ha imparato con successo come guidare nel tracciato, portando a termine la prova a tempo con una media di circa un minuto e cinque secondi misurata su venti tentativi; un giocatore umano di livello medio (non particolarmente esperto ma nemmeno un principiante, in grado di utilizzare correttamente la meccanica della derapata) nelle stesse condizioni dell'agente (accelerazione costante e senza l'utilizzo di oggetti) è in grado di completare il tracciato in circa un minuto e un secondo, con una differenza di tempo quindi trascurabile. In Tabella 4.2 si possono vedere i vari tempi realizzati dall'agente insieme al numero di volte in cui essi sono stati conseguiti.

Tempo di completamento	#
1.03	3
1.04	3
1.05	2
1.06	8
1.07	4

Tabella 4.2: Tempi realizzati dall'agente su venti tentativi

Tra i comportamenti degni di nota dell'agente si riscontrano un buon utilizzo della derapata (ma senza l'attivazione del Miniturbo), un utilizzo intensivo del salto (che gli permette di evitare il malus dato dai terreni accidentati e non comporta conseguenze negative su un terreno "standard") e la scoperta di una possibile scor-

ciatoia, che gli permette di tagliare una curva percorrendo uno specifico tratto d'erba saltando (in figura 4.5 è possibile vedere l'agente che tenta di tagliare la curva).

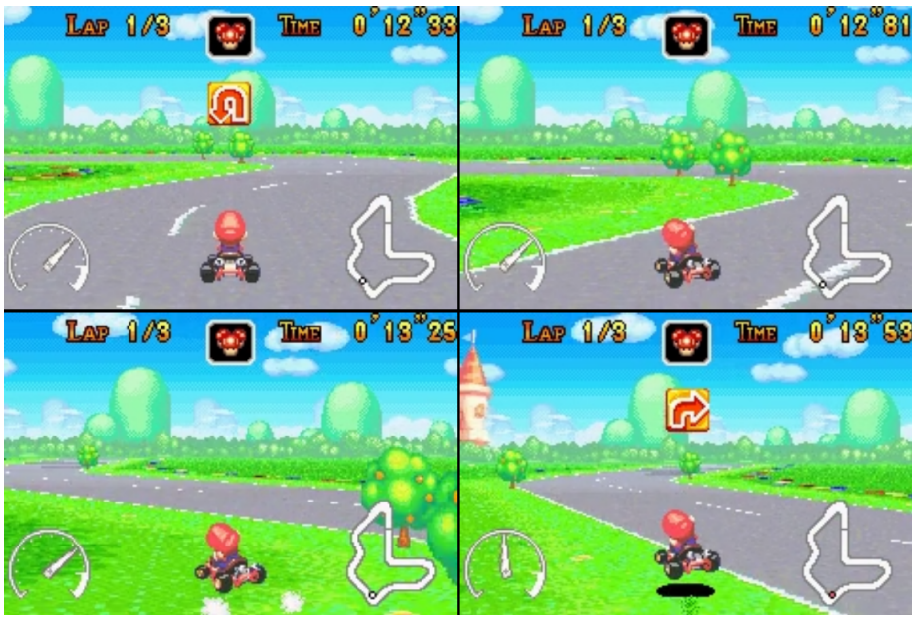

Figura 4.5: L'agente in salto mentre prende una scorciatoia

4.2.2 Addestramenti con comandi completi

In seguito sono stati condotti diversi addestramenti con comandi completi, includendo quindi anche l'azione di attesa e l'uso di oggetti. L'obiettivo di questi addestramenti è realizzare un agente in grado di guidare correttamente, effettuare una partenza corretta e utilizzare i tre Funghi turbo in maniera logica; in un primo momento i pesi utilizzati nella funzione Reward sono stati mantenuti invariati.
Il primo addestramento è stato condotto per venti milioni di iterazioni, come nel caso precedente; in figura 4.6 è possibile vedere i relativi grafici.

Anche in questo caso l'agente riesce a concludere la prova a tempo, ma i tempi di completamento risultano peggiori, mantenendosi intorno a un minuto e ventidue secondi. La causa di questo è principalmente la tendenza dell'agente a non mantenere premuto l'acceleratore (dalle registrazioni si può vedere come alterni le azioni che comprendono l'uso dell'acceleratore ad azioni di attesa), mantenendo quindi una

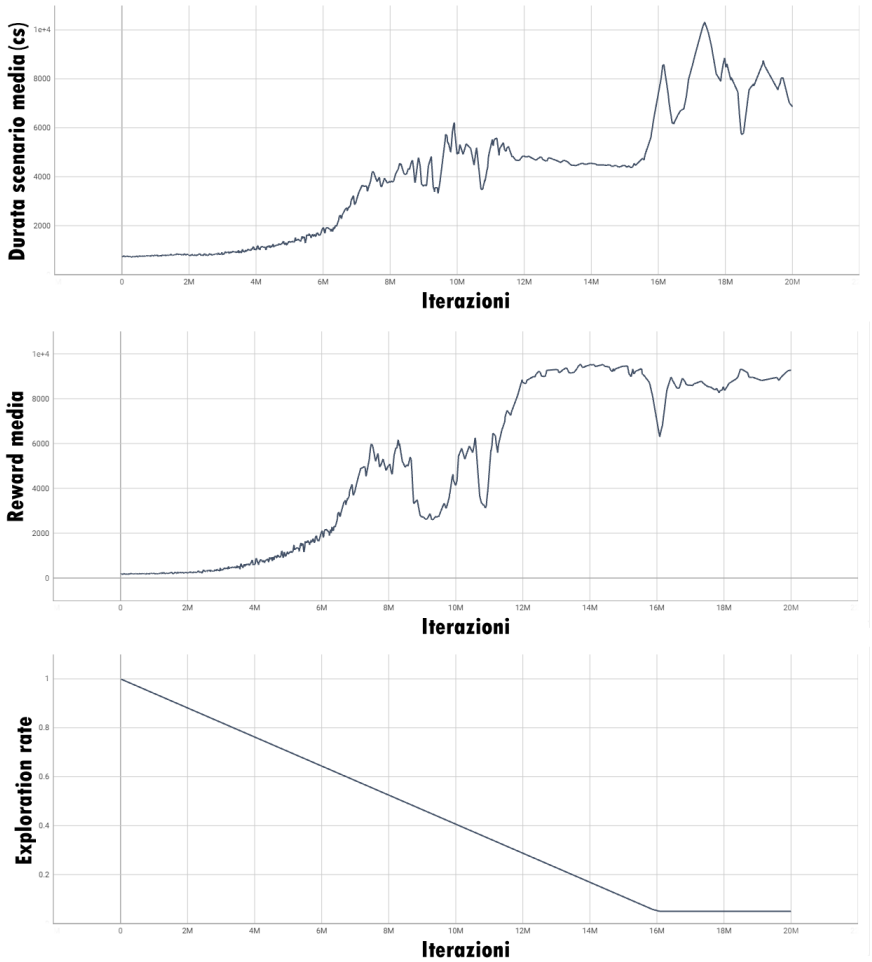

Figura 4.6: I grafici relativi al primo addestramento con comandi completi

velocità media più bassa. Inoltre, l'azione di attesa non ha portato particolari benefici al momento della partenza, poiché l'agente tende a sbagliare la partenza per poi compensare il malus con l'utilizzo di un Fungo turbo. Per compensare queste problematiche sono state pensate due soluzioni:

- incrementare la velocità da mantenere per azzerare l'impatto della componente negativa speed nella funzione Reward (come visibile in sezione 3.2.2), portandola da 700 a 1080 (la velocità massima raggiungibile in un terreno "standard" senza utilizzare alcun tipo di turbo è circa 1090); in questo modo, il kart sarà incentivato a mantenere una velocità più elevata per massimizzare la Reward ottenuta

- aggiungere una forte penalità alla funzione Reward nel caso in cui si utilizzi un Fungo turbo entro i primi quattro secondi dall'inizio della gara (detta *start penalty*)

Il primo addestramento eseguito con queste modifiche è stato condotto per più di 14 milioni di iterazioni, circa equivalenti a 44 ore. L'agente addestrato in questo modo mostra dei miglioramenti, con dei tempi di completamento della prova a tempo che vanno da un minuto e dodici secondi a un minuto e diciotto secondi e una velocità media più elevata; tuttavia, la *start penalty* non previene la partenza sbagliata, che ora ha un impatto più elevato dato che l'agente non utilizza l'oggetto per compensare il malus ricevuto.
Per cercare di migliorare la situazione è stato condotto un terzo addestramento con i seguenti cambiamenti:

- è stato raddoppiato il tempo in cui viene applicata la penalità legata all'utilizzo di un Fungo turbo subito dopo la partenza (che passa da quattro a otto secondi)
- sono stati modificati alcuni pesi della funzione Reward (visibili in tabella 4.3); in particolare, **time_c** passa da -0.5 a -5 per incrementare l'effetto che il tempo di completamento della prova ha sulla Reward totale. In tabella viene mostrato anche il peso relativo alla start penalty, rimasto invariato rispetto all'addestramento precedente.

Componente reward	**Valore**
wall_c	-20
cp_c	400
terrain_c	-20
time_c	-5
speed_c	-5
backwards_c	-2
start_c	-1000

Tabella 4.3: I pesi utilizzati nel terzo addestramento con comandi completi

Al termine di quest'ultimo addestramento (dalla durata di circa 59 ore) l'agente ha nuovamente appreso in modo corretto come terminare la corsa, con tempo medio di completamento (misurato su venti tentativi, visibili in tabella 4.4) di un minuto e undici secondi, mostrando quindi un leggero miglioramento rispetto all'addestramento precedente; nonostante il miglioramento delle tempistiche, in alcuni casi (due su venti) l'agente non è riuscito a completare la prova a tempo, a causa di troppi urti consecutivi contro i muri.

Tempo di completamento	#
1.05	1
1.07	1
1.09	3
1.10	4
1.11	2
1.12	1
1.13	4
1.14	2
Non completata	2

Tabella 4.4: Tempi realizzati dall'agente in venti tentativi

Figura 4.7: L'agente mentre subisce il malus dato dalla partenza sbagliata

Tuttavia, la problematica legata alla partenza rimane praticamente inalterata, visto che l'agente è incapace di moderare l'uso dell'acceleratore per non subire il malus iniziale (come visibile in figura 4.7).

4.2.3 Test e problematiche rilevate

In questa sottosezione verranno fatte alcune considerazioni sull'agente migliore (quello addestrato con comandi limitati) e ne verrà testata la capacità di generalizzare su altri percorsi.
Come visto nella sottosezione 4.2.1, l'agente riesce a concludere con successo la prova a tempo, e considerando i limiti a cui deve sottostare a causa dei comandi limitati (senza quindi avere la capacità di smettere di accelerare o di utilizzare i Funghi turbo), lo fa con dei buoni tempi. Tuttavia, l'agente in questione presenta comunque delle problematiche che possono essere migliorate con addestramenti futuri:

- **Guida fuori strada**: nonostante l'agente tenda a guidare per la maggior parte del tempo sul tracciato "standard" (che non influisce negativamente sulla velocità del kart), a volte, specialmente dopo o durante delle curve, capita che esso finisca fuori strada per qualche secondo, riducendo notevolmente la sua velocità (come visibile in figura 4.8).

Figura 4.8: L'agente fuori strada al termine di una curva

- **Mancata attivazione del Miniturbo**: come visto in sezione 2.4.2, la meccanica del Minuturbo conferisce un breve scatto di accelerazione al kart dopo

che esso ha eseguito una derapata per un certo periodo di tempo. Nonostante l'agente abbia appreso come derapare per affrontare al meglio le curve più ostiche, non riesce comunque ad attivare questa meccanica; generalmente questo accade poiché per attivare il Miniturbo il pad direzionale della console non deve mai trovarsi in posizione neutra mentre si derapa (un esempio di sequenza di comandi valida è ad esempio "**LEFT+A+R, RIGHT+A+R, LEFT+A+R**" mentre una che non attiva il turbo è "**LEFT+A+R, A+R, LEFT+A+R**"). In altri casi può capitare che l'agente interrompa brevemente un'azione di derapata per eseguire un'azione di salto (come visibile in figura 4.9), non raggiungendo il tempo necessario per l'attivazione del Miniturbo.

Figura 4.9: L'agente annulla brevemente una derapata con un'azione di salto

Per testare la capacità di generalizzazione dell'agente esso è stato testato su un tracciato simile a quello in cui è stato addestrato (Peach Circuit): Mario Circuit, di cui è visibile una schermata in figura 4.10; anche quest'ultimo presenta una strada asfaltata circondata da un terreno principalmente erboso, caratteristiche che lo accomunano al primo percorso.

Nonostante le similarità, l'agente riesce con successo a completare solamente una parte della prova, arrivando nel migliore dei casi a circa metà del secondo giro prima di fallire a causa di troppi urti contro le pareti (come visibile in figura 4.11). Per migliorare la generalizzazione una possibile soluzione può essere quella di addestrare l'agente su più percorsi, al fine di permettergli di apprendere come gestire al meglio le situazioni indipendentemente dal tipo di tracciato.

Figura 4.10: La prima parte del tracciato Mario Circuit

Figura 4.11: L'agente finisce fuori strada e si scontra contro la parete esterna del tracciato Mario Circuit

4.3 Quick Run

In questa sezione vengono presentati i risultati ottenuti nella modalità Quick Run, utilizzando come personaggio Mario nel tracciato Peach Circuit alla cilindrata 150cc. L'obiettivo della modalità è concludere la gara nella posizione migliore possibile, pertanto la scelta dei pesi della funzione Reward punta a massimizzare questo risultato. Considerando inoltre l'importanza dell'uso degli oggetti in questa modalità e l'assenza di Funghi turbo al momento della partenza gli addestramenti sono stati eseguiti con i comandi completi.
In tabella 4.5 vengono mostrati i pesi utilizzati per gli addestramenti; l'unica differenza con i pesi visti precedentemente per la modalità Time Trial è l'inclusione di un modificatore alla componente relativa ai checkpoint superati: grazie ad essa

il kart guadagnerà più o meno punti a seconda della sua posizione al momento del superamento della sezione, in un range che va dal 30% in meno al 40% in più.

Componente reward	**Valore**
wall_c	-10
cp_c	400 * (1.4 - posizione/10)
terrain_c	-15
time_c	-0.5
speed_c	-0.5
backwards_c	-2
start_c	0

Tabella 4.5: I pesi nell'addestramento sulla modalità Quick Run

Questa scelta è stata presa per spronare l'agente a conquistare una buona posizione fin dall'inizio, senza concentrarsi solamente sul risultato finale; questo anche in funzione del fatto che separarsi presto dagli avversari in coda al gruppo garantisce generalmente una minore probabilità di essere colpiti da oggetti più pericolosi (dato che vengono conferiti principalmente in base alla posizione).

L'addestramento è stato condotto per circa trenta milioni di iterazioni, per un totale di circa 80 ore; in figura 4.12 e 4.13 vengono mostrati i relativi grafici.

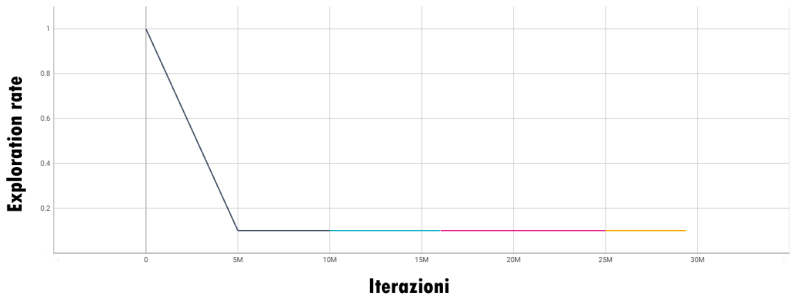

Figura 4.12: Grafico relativo all'addestramento nella modalità Quick Run (andamento dell'exploration rate)

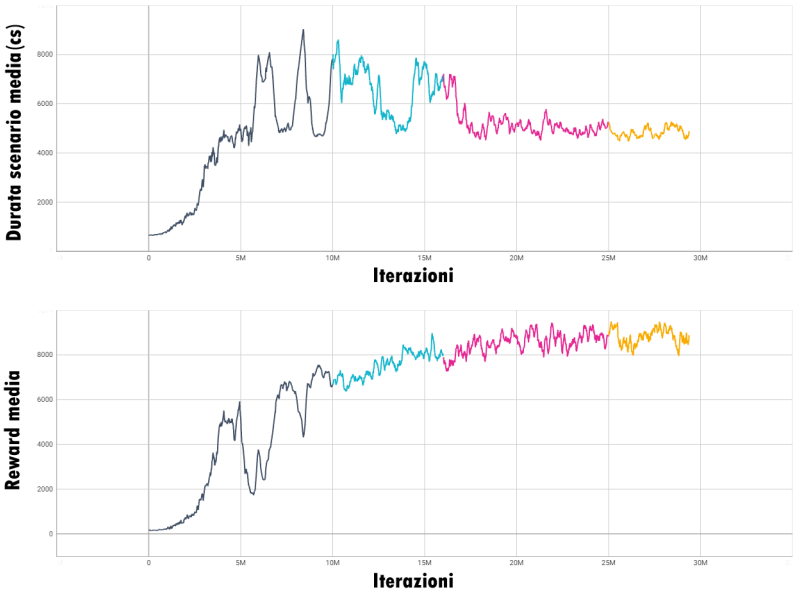

Figura 4.13: Grafici relativi all'addestramento nella modalità Quick Run (media della durata dello scenario e della reward)

Nella prima parte dei grafici si può vedere un netto miglioramento della Reward, corrispondente alla conclusione della gara con conseguente superamento di tutti i checkpoint; i successivi miglioramenti meno ripidi mostrano un graduale miglioramento della posizione media raggiunta.

Posizione	#
1	0
2	1
3	2
4	3
5	4
6	2
7	6
8	2

Tabella 4.6: Posizione finale dell'agente in venti gare

Al termine dell'addestramento l'agente è in grado di terminare la gara in media tra la quinta e la sesta posizione, un buon risultato considerando l'alto numero di componenti casuali che influiscono sul risultato di una corsa; tuttavia, proprio a causa dell'elevata casualità, in diverse gare l'agente viene colpito da diversi strumenti lanciati dagli avversari e non riesce a trovare dei buoni oggetti, rimanendo nelle ultime due posizioni. In tabella 4.6 si possono vedere le varie posizioni conquistate dall'agente in venti gare.
Inoltre, non avendo modo di compensare una partenza sbagliata, ha prima imparato come partire in maniera normale e poi come effettuare una partenza turbo, così da guadagnare una buona posizione all'inizio della corsa (come visibile in figura 4.14).

Figura 4.14: L'agente in prima posizione dopo aver effettuato una partenza turbo

4.3.1 Test e problematiche rilevate

Come nel caso della modalità Time Trial, in questa sottosezione verranno affrontate alcune problematiche dell'agente addestrato e verrà valutata la sua capacità di generalizzazione.
Come visto in precedenza, l'agente riesce sempre a concludere la gara, con una posizione che può variare significativamente in base agli oggetti utilizzati dai suoi av-

versari e dallo stesso agente; di seguito vengono elencate alcune delle problematiche emerse dai test effettuati:

- **Guida fuori strada**: l'agente fa fatica a rimanere sempre sul tracciato "standard", finendo occasionalmente fuori strada; questo lo porta a perdere posizioni e farsi superare dai suo avversari, come visibile in figura 4.15

Figura 4.15: L'agente finisce fuori strada poco dopo l'inizio della gara

- **Utilizzo degli oggetti**: per massimizzare la probabilità di finire la gara in una delle prime posizioni è importante utilizzare appieno gli oggetti raccolti sul tracciato. In alcuni casi l'agente riesce in questo compito, lanciando subito strumenti che hanno un effetto immediato (come i gusci rossi, che inseguono e colpiscono l'avversario davanti a chi li lancia) e tenendo per un secondo momento oggetti che possono avere un impatto maggiore in futuro, come il Fungo turbo o la Superstella. Nel caso di quest'ultima, in figura 4.16 è possibile vedere come l'agente non la usa nel momento in cui la ottiene, ma aspetta di trovarsi di fronte ad un rettilineo, probabilmente per massimizzare il bonus di velocità conferito dall'oggetto e per sfruttare l'invincibilità ottenuta.

Figura 4.16: L'agente attende qualche secondo prima di utilizzare una Superstella

Tuttavia, ci sono casi di oggetti la cui utilità principale è quella di essere mantenuti dietro il kart, tramite la pressione prolungata del tasto "**L**"; ne è un esempio la Buccia di banana (visibile in figura 4.17) che se mantenuta dietro il kart in questo modo protegge il giocatore da un oggetto in arrivo.

Figura 4.17: Una buccia di banana mantenuta dietro il kart del giocatore

In questi casi l'agente non comprende questo utilizzo degli oggetti, e li lancia subito alle sue spalle per lasciarli come ostacolo sul tracciato privandosi quindi di una possibile protezione.

- **Guida non ottimale**: come detto in precedenza, nelle gare in cui il kart viene colpito spesso da oggetti avversari e rimane in ultima posizione difficilmente

riuscirà a recuperare senza l'utilizzo degli strumenti migliori del gioco. Questo perché le sue abilità di guida non sono ottimali, e rispetto agli agenti addestrati nella modalità Time Trial la sua capacità di eseguire tecniche più complesse come la derapata è limitata.

Come per la modalità Time Trial, sono state condotte alcune gare sul percorso Mario Circuit, per testare il comportamento dell'agente su tracciati mai visti; come nel caso precedente, nei tentativi migliori l'agente completa una parte di percorso, ma non riesce ad arrivare al traguardo. Inoltre, una particolarità che si può evidenziare è che nonostante il tempismo per effettuare la partenza turbo sia lo stesso, in questo caso l'agente non riesce ad effettuarla correttamente (come visibile in figura 4.18). Anche in questo caso si potrebbe addestrare l'agente su più percorsi, per fargli apprendere come giocare correttamente a prescindere dalle caratteristiche del tracciato.

Figura 4.18: L'agente sbaglia la partenza nel percorso Mario Circuit

CONCLUSIONI

Grazie alla piattaforma *Gym Retro* [6] è possibile applicare tecniche di *Reinforcement Learning* [14] ad una moltitudine di videogiochi, sia quelli più basilari (dei quali esistono numerosi esempi in letteratura) che quelli più complessi. Nel corso di questo lavoro di tesi è stata analizzata un'integrazione già esistente per *Super Mario Kart*, per poi realizzarne una completamente nuova per il gioco *Mario Kart: Super Circuit*, rilasciato per *Game Boy Advance*.

Utilizzando l'integrazione realizzata sono stati addestrati diversi agenti su due differenti modalità di gioco, *Time Trial* e *Quick Run*, mostrando le potenzialità di tecniche di RL (nello specifico utilizzando delle *Deep Q-Network* [20]) nell'apprendimento corretto dei comportamenti di gioco. I risultati ottenuti sono stati analizzati per comprendere al meglio le particolarità di ogni approccio, evidenziandone al meglio i limiti e i traguardi raggiunti, oltre all'importanza della definizione di una corretta funzione di Reward che vada a premiare i giusti comportamenti.

Questo lavoro di tesi si presta ad ulteriori sviluppi futuri, sia per risolvere problematiche già evidenziate in precedenza, come la mancanza di generalizzazione tra i diversi circuiti (provando ad addestrare l'agente su più percorsi), che per sperimentare addestramenti in diverse condizioni, analizzando per esempio il peso che può avere la scelta di un determinato personaggio nel modo in cui l'agente apprende come giocare. Inoltre, per migliorare la comprensione del movimento da parte dell'agente si potrebbero effettuare degli addestramenti passando in input alla rete uno *stack* di più frame consecutivi.

Allo scopo di facilitare sviluppi futuri anche da parte di terzi, l'integrazione realizzata nel corso di questo lavoro di tesi è stata resa disponibile su *GitHub*, permet-

tendo così a chi si approccia al problema di concentrarsi solamente sulle tecniche di addestramento.

Bibliografia

[1] Brian Byrne, Brian (2017). History of the Super Nintendo (SNES): Ultimate Guide to the SNES Games & Hardware. Console Gamer Magazine. p. 4. ISBN 978-1549899560.

[2] Fabio Rossi, Modo 7, in Dizionario dei videogame, collana Domino n° 19, Milano, Vallardi, novembre 1993, p. 240, ISBN 88-11-90422-6.

[3] https://it.wikipedia.org/wiki/Mario_(serie_di_videogiochi)#Serie_principale

[4] Morgan, Thomas (September 4, 2017). "Tech Evolution: 25 years of Super Mario Kart". Eurogamer. Retrieved January 12, 2023.

[5] Sascha Lange and Martin Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In Neural Networks (IJCNN), The 2010 International Joint Conference on, pages 1–8. IEEE, 2010

[6] Sito ufficiale di Gym Retro. `https://openai.com/blog/gym-retro/`

[7] Repository Github di Gym. `https://github.com/openai/gym`

[8] Sito ufficiale di OpenAI. `https://openai.com`

[9] Documentazione di Gym. `https://gymnasium.farama.org/content/basic_usage/`

[10] La RAM map di Super Mario Bros. per NES. `https://datacrystal.romhacking.net/wiki/Super_Mario_Bros.:RAM_map`

[11] Repository github contenente l'integrazione di Super Mario Kart. https://github.com/esteveste/gym-SuperMarioKart-Snes

[12] Sito ufficiale di Visual Boy Advance. https://visualboyadvance.org

[13] Definizione di Intelligenza Artificiale. https://en.wikipedia.org/wiki/Artificial_intelligence

[14] Definizione di Reinforcement Learning. https://en.wikipedia.org/wiki/Reinforcement_learning

[15] Definizione di Rinforzo. https://it.wikipedia.org/wiki/Rinforzo

[16] Tassonomia degli algoritmi di RL. https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#citations-below

[17] Definizione delle equazioni di Bellman. https://spinningup.openai.com/en/latest/spinningup/rl_intro.html#bellman-equations

[18] Definizione di Deep Learning. https://it.wikipedia.org/wiki/Apprendimento_profondo

[19] Grafico che mostra le differenze fra Q-Learning e Deep Q-Learning. https://knowledge.dataiku.com/latest/_images/illustration-71.jpg

[20] Paper del primo modello di DQN. https://arxiv.org/pdf/1312.5602v1.pdf

[21] Definizione di Convolutional Neural Network. https://en.wikipedia.org/wiki/Convolutional_neural_network

[22] Definizione di RGB. https://en.wikipedia.org/wiki/RGB_color_model

[23] Definizione di MPC. https://en.wikipedia.org/wiki/Model_predictive_control

[24] Documentazione di Stable Baselines 3 relativa alla DQN. https://stable-baselines3.readthedocs.io/en/master/modules/dqn.html

[25] Pagina wikipedia della console Atari 2600. https://it.wikipedia.org/wiki/Atari_2600

[26] Definizione di frame rate. https://en.wikipedia.org/wiki/Frame_rate

[27] Operazioni di frame-skipping e preprocessing di un gioco Atari. https://danieltakeshi.github.io/2016/11/25/frame-skipping-and-preprocessing-for-deep-q-networks-on-atari-2600-games/

[28] La CPU utilizzata. https://www.amd.com/en/products/cpu/amd-epyc-7282

[29] La GPU utilizzata. https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/quadro-product-literature/quadro-rtx-5000-data-sheet-us-nvidia-704120-r4-web.pdf

[30] Documentazione ufficiale di Stable Baselines 3. https://stable-baselines3.readthedocs.io/en/master/

[31] Sito ufficiale di PyTorch. https://pytorch.org

[32] Guida allo stile di codice PEP 8. https://peps.python.org/pep-0008/

[33] Documentazione di TensorBoard. https://www.tensorflow.org/tensorboard?hl=it