Progetto di Ingegneria Informatica

Sviluppo di bot Telegram in Python

Flavio Di Lorenzo - Numero matricola: 961351

Anno Accademico 2023/2024
Tutor: Giovanni Agosta

Indice:

1 Descrizione del progetto

1.1 Obiettivi

Il progetto ha come obiettivo la realizzazione di un bot Telegram in grado di assistere una campagna di un gioco di ruolo open source, con funzioni di supporto utili ai giocatori. È dunque un bot reattivo, che, fino a quando il server rimane attivo, è in attesa di input da parte degli utenti giocanti.

Il gioco in questione è "Basic Roleplaying" di cui sono state implementate tutte le funzionalità, a partire dalla creazione del personaggio, fino alla gestione dei turni di combattimento.

Tramite il bot creato un qualsiasi giocatore può connettersi e simulare partite con gli amici (in questo caso un membro del gruppo verrà designato col ruolo di Game Master) o da solo (svolgendo sia il ruolo di player che di GM allo stesso tempo). Il bot si occuperà di salvare le informazioni relative ai personaggi durante le partite e a calcolare le probabilità dei vari scenari così come i danni inflitti durante i combattimenti in modo da rendere il gioco fluido per tutti gli utenti novizi e non.

1.2 Tecnologia utilizzata

Per la realizzazione del bot è stato scelto il linguaggio Python, e, in particolare, il modulo "python-telegram-bot" , che fornisce una traduzione in Python di tutti i metodi dell'API messa a disposizione da Telegram. Il modulo fornisce anche alcune classi estese in grado di occuparsi di alcuni aspetti tecnici specifici della realizzazione di un bot ad alto livello: ad esempio il riconoscimento di comandi o la gestione di una conversazione, tramite simulazione di una macchina a stati.

1.3 Scelte implementative

È stato necessario fare delle scelte per quanto riguarda il salvataggio dei dati necessari per il gioco. Nel caso della memoria a lungo termine, il materiale viene salvato in formato ".json" all'interno della cartella "BasicRolePlay_TelegramBot/Json" (interna alla repository del progetto) a cui il codice ha permesso di scrittura e lettura. Per quanto riguarda quella a breve termine, invece, ho fatto uso di strutture temporanee disponibili in Python per il salvataggio momentaneo dei dati.

1.3.1 Memoria a lungo termine

Per permettere il corretto funzionamento di una campagna di gioco il bot necessita di salvare tutte le informazioni relative ai fari personaggi: nome, abilità, statistiche, armi, punti vita, etc. per ovviare al problema ad ogni utente viene creato un file ".json" presente nella cartella "/users". Al momento della creazione del primo personaggio da parte dell'utente, il programma aggiunge in automatico il file alla cartella "/users".

Ogni persona ha la possibilità di creare più personaggi. Infatti, ogni volta che un nuovo "character" viene creato, questo viene aggiunto al file ".json" privato del giocatore, che potrà poi scegliere liberamente quale adoperare nella campagna di gioco

1.3.2 Memoria a breve termine

In alcuni casi, invece di ricorrere immediatamente al salvataggio tramite file ".json" ho ritenuto opportuno salvare temporaneamente i dati. Per fare ciò, ho usato principalmente due strutture: liste e dizionari.

Le prime sono state usate nei casi dove il giocatore è obbligato a fare scelte multiple con lo stesso messaggio, ad esempio la selezione di più abilità contemporaneamente. In questo caso, gli elementi inseriti dall'utente sono salvati in una lista che verrà ispezionata per garantire la correttezza e la validità dell'input. Solo dopo questo processo di verifica i dati potranno essere salvati definitivamente passando dalla memoria a breve a quella a lungo termine tramite trascrizione su un file ".json".

I dizionari, invece, sono stati usati in fasi dove era necessario mantenere o dare una certa struttura alle informazioni inserite dall'utente. Un esempio è la fase di creazione del personaggio dove l'utente inserisce a mano a mano tutte le caratteristiche per comporre il suo "alter-ego" di gioco. In questa circostanza è infatti richiesta una struttura ben precisa dell'informazioni, dato che poi altre funzioni dovranno accedere ai dati del personaggio creato. Per fare ciò, la struttura viene prima estratta da un file ".json" e assegnata ad un dizionario che verrà mano a mano riempito. Solo una volta terminato il processo, il dizionario verrà salvato nel file del giocatore. In questo modo, nel caso in cui l'utente decidesse di annullare la creazione del personaggio, verrebbe semplicemente eliminato il dizionario senza accedere inutilmente alla cartella dei ".json".
L'utilizzo di queste strutture dati per gestire la memoria a breve termine si è rivelato fondamentale per evitare inutili accessi a file, ottimizzando così la velocità del codice.

1.3.3 Info gioco

Durante il gioco è permesso al player di usare abilità, armi, di comprare oggetti quali armature, scudi e così via. Per garantire la coerenza di gioco sono stati creati file ".json" contenenti tutti gli oggetti presenti "in game" così come tutte le specifiche delle varie abilità, armi, professioni, etc. In particolare, nella directory "Json" troviamo:

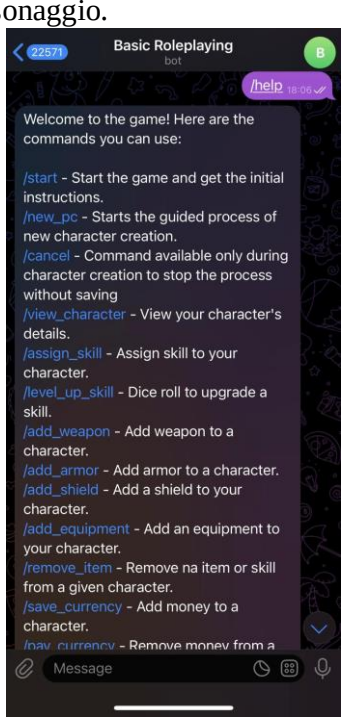
- users – Contiene i file ".json" relativi ai singoli utenti
- weapons – Al suo interno troviamo quattro file ogniuno contenente tutte le specifiche necessarie al funzionamento del gioco di ogni arma utilizzabile:
    - firearm - contiene la lista di tutte le armi da fuoco presenti
    - heavy - è un elenco di armamenti pesanti da usare in combattimento
    - melee - è dedicato alle armi a corto raggio usate in scontri ravvicinati per attaccare, oppure per parare un colpo avversario
    - ranged - è l'elenco delle armi non da fuoco a lungo raggio disponibili "in game"
- armors – Al suo interno troviamo la lista di armature. Di ogniuna sono specificati i punti ferita da aggiungere alla vita del personaggio che la indossa ed i relativi potenziamenti o depotenziamenti che dona.
- character_sheet – È la guida di come la scheda contenente tutte le info di un personaggio dovrebbe sembrare.
- genre_choice – Lista di generi selezionabili per un personaggio. Può essere facilmente espandibile se ritenuto necessario
- professions – Oltre a mostrare tutte le possibili professioni, specifica per ogniuna le relative abilità primarie e secondarie che ne derivano.
- races – contiene un dizionario per ogni razza (al momento solo "human") che ne specifica i tratti distintivi
- resistance_table – Tabella usata per decidere l'esito di un **"/resitance_roll"**. In orizzontale abbiamo i livelli della statistica attiva mentre in verticale quelli della passiva. Ritorna la percentuale di successo di quella attiva.
- shields - Elenco di scudi presenti "in game" con relative info per il combattimento
- skills – Al suo interno si trova la lista di abilità assegnabili. Per ogniuna è salvato il livello di partenza dell'abilità e, in alcuni casi, un elenco delle abilità secondarie più specifiche derivate da quella base
- special_success – Tabella usata per gestire il successo speciale

2 Funzioni di BRP

2.1. Panoramica

I comandi disponibili in “Basic Roleplaying” sono i seguenti:

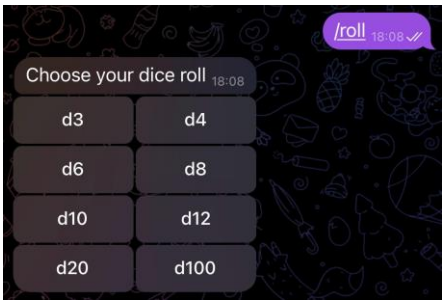
- **/start** - Avvia il gioco e ricevi le istruzioni iniziali.
- **/new_pc** - Avvia il processo guidato di creazione di un nuovo personaggio.
- **/cancel** - Comando disponibile solo durante la creazione del personaggio per interrompere il processo senza salvare.
- **/view_character** - Visualizza i dettagli del tuo personaggio.
- **/assign_skill** - Assegna una competenza al tuo personaggio.
- **/level_up_skill** - Tiro di dado per migliorare una competenza.
- **/add_weapon** - Aggiungi un’arma a un personaggio.
- **/add_armor** - Aggiungi un’armatura a un personaggio.
- **/add_shield** - Aggiungi uno scudo al tuo personaggio.
- **/add_equipment** - Aggiungi un equipaggiamento al tuo personaggio.
- **/remove_item** - Rimuovi un oggetto o una competenza da un determinato personaggio.
- **/save_currency** - Aggiungi denaro a un personaggio.
- **/pay_currency** - Rimuovi denaro da un personaggio.
- **/ability_roll** - Tiro di dado per usare un’abilità del personaggio.
- **/ability_vs_ability** - Decidi il vincitore tra due abilità usate contemporaneamente.
- **/resistance_roll** - Tiro di dado per usare una statistica contro una resistenza.
- **/stat_roll** - Tiro di dado per usare una statistica.
- **/attack** - Calcola il danno e il risultato di un attacco dato la distanza del bersaglio e l’arma utilizzata.
- **/evade** - Restituisce il risultato di un tentativo di schivare l’attacco dato il danno e il tipo di successo.
- **/shield** - Restituisce il risultato di un tentativo di parare l’attacco dato il danno e il tipo di successo.
- **/remove_hp** - Usato per infliggere danno nel caso in cui il difensore non abbia né l’abilità “Schivare” né uno scudo.
- **/heal** - Rigenera i punti vita del personaggio del numero indicato.
- **/roll** - Tiro di dado generico con opzione per più dadi.
- **/help** - Mostra tutti i comandi possibili.



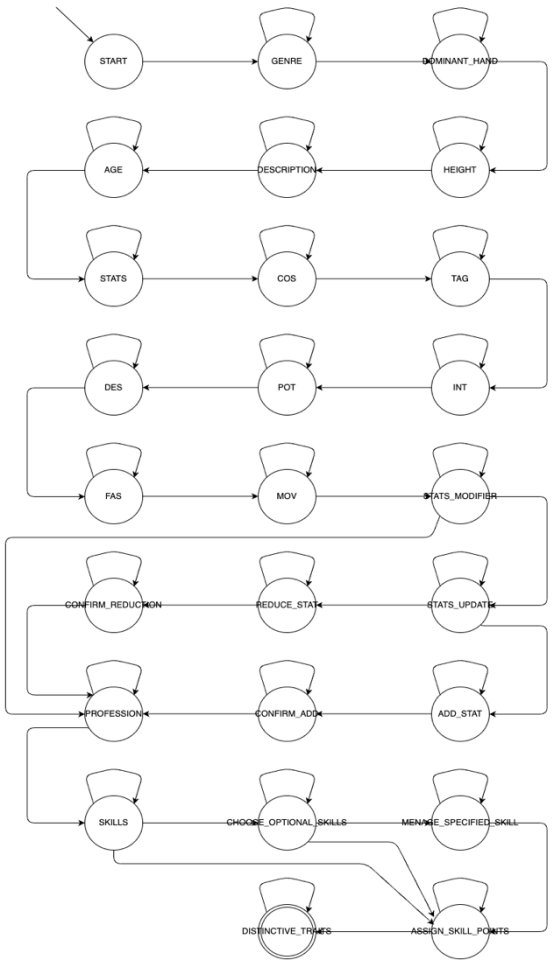
Per ogni funzione è previsto un controllo iniziale del formato utilizzato nella chiamata. In caso questo non sia riconosciuto come valido viene mostrato all’utente il formato giusto da utilizzare per chiamare correttamente il comando

I comandi presenti in game richiedono un ordine di utilizzo.
Tramite “/start” il bot viene attivato e solo successivamente sarà possibile iniziare la chat di gioco.
Il secondo comando da utilizzare è invece **“/new_pc”** per la creazione del personaggio. Infatti, nessuna delle funzioni progettate è utilizzabile se l’utente non ha un personaggio su cui utilizzarle. Solo in questa fase sarà disponibile **“/cancel”** per annullare la procedura iniziata con **“/new_pc”**.
Successivamente l’utente sarà in grado di utilizzare anche tutti gli altri comandi, specificando il personaggio con cui vanno usati.
Fa eccezione il comando **“/help”** che mostra l’elenco delle funzionalità disponibili. Questo, infatti, non ha bisogno di un personaggio esistente per essere eseguito e sarà dunque attivo subito dopo l’avvio del bot tramite **“/start”**.

2.2 Lancio dei dadi



La funzione di lancio dei dadi è disponibile sia in versione testuale che grafica, tramite l'utilizzo di una tastiera virtuale "inline".
La versione testuale è in grado di accettare il lancio di multipli gruppi di dadi, oltre a un bonus o malus. Il comando viene invocato tramite la sintassi "**/roll** <input>", ad esempio: un comando accettato è "**/roll** 3d8, 1d6, 2d4 + 5". Per riconoscere la validità della richiesta dell'utente viene utilizzata un'espressione regolare "pattern" con cui va confrontata la stringa ricevuta in input. In seguito, una fase di parsing dell'input si occupa di riconoscere i diversi gruppi di lanci da effettuare. Il risultato è presentato all'utente facendo vedere lanci parziali e il totale.
La versione grafica è invocata semplicemente dal comando **"/roll"**. L'utente riceve in risposta una tastiera virtuale persistente, che permette di scegliere tra varie tipologie di dadi utili al fine del gioco:



2.3 Creazione del personaggio

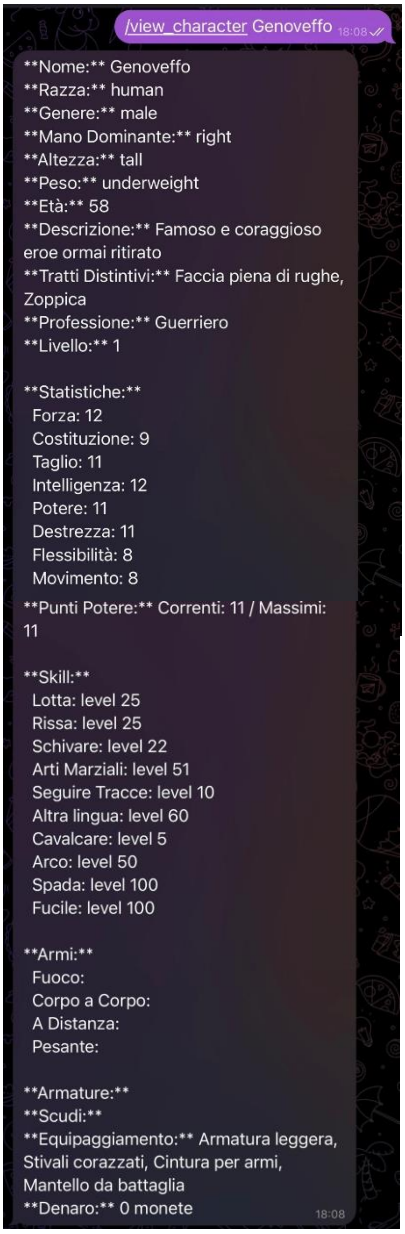
Per la creazione del personaggio, l'utente è guidato attraverso il processo da una serie di messaggi lineari preimpostati in grado di rispondere a modo ai vari input.
Per realizzare questa sequenza lineare, è stato scelto di utilizzare un ConversationHandler. Questa opzione messa a disposizione dal modello "python-telegram-bot" è in grado di simulare una macchina a stati dove ad ogni scelta dell'utente corrisponde uno stato della conversazione.
Tra i vari stati, li dove sono richieste delle scelte vincolate, vengono mostrate all'utente le possibili opzioni tramite tastiera virtuale apposita, per facilitarne il compito. Le scelte fatte dall'utente durante il processo vengono salavate a mano a mano nel dizionario temporaneo "tmp_user_data" che rappresenta la scheda del personaggio. In questo modo, qualora l'utente decidesse di annullare la creazione del nuovo personaggio tramite l'apposito comando **"/cancel"**, il dizionario verrebbe ripulito prima ancora di essere salvato poi nel file del giocatore.
Il dizionario è salvato nel file ".json" dell'utente a creazione finalizzata. Il file ".json" è univoco per ogni giocatore, e contiene tutti i personaggi che ha creato.
Il comando **"/new_pc"** dà il via alla macchina a stati per la gestione della creazione nuovo personaggio

2.4 Visione della scheda personaggio

Il comando viene invocato con la sintassi "**/view_character** <character name>". Le informazioni della scheda del personaggio corrispondente vengono recuperate dal file ".json" dell'utente in questione e mostrate con un messaggio di risposta ordinato per garantirne la leggibilità.

2.5 Gestione finanze

Le finanze di un personaggio sono gestibili tramite i due appositi comandi **"/save_currency"** e **"/pay_currency"**.
Il primo si occupa di aggiungere un certo importo monetario alle casse del giocatore specificando a fianco al comando il nome del personaggio a cui va aggiunto il denaro e la quantità da aggiungere preceduta da una virgola (**/save_currency** <character name>, <quantity>).
Il secondo, invece, gestisce la rimozione di monete di un personaggio specificato seguendo sempre l'apposito formato per il riconoscimento del comando (**/pay_currency** <character name>, <quantity>).



2.6 Gestione "items"

Per "item" che un personaggio può possedere si intendono: armi, scudi, armature, abilità ed equipaggiamenti vari.

Tramite il comando **"/add_armor"** si può aggiungere ad un personaggio esistente una delle armature presenti in gioco. Queste sono elencate in un file ".json" che comprende anche i relativi potenziamenti o depotenziamenti che danno se indossate.
Usando **"/add_shield"** l'utente può invece dotare un "character" di uno scudo. Anche in questo caso il numero di scudi da cui scegliere è limitato e tutte le opzioni sono presenti in un file ".json" dotato di specifiche dei relativi oggetti.
Il comando **"/add_equipment"** è invece più generico e permette di aggiungere un qualsiasi item. Solitamente viene usato per assegnare del vestiario o comunque elementi poco rilevanti al fine dei combattimenti.
Infine, **"/add_weapon"** permette di aggiungere un'arma preimpostata all'inventario del personaggio specificato. Le armi sono divise in quattro categorie: ravvicinate, da fuoco, a distanza e pesanti. Ogni arma ha i propri punti ferita (nel caso venga usata per parare), il proprio danno, l'abilità necessaria per usarla e la distanza massima che può coprire. Tutte queste informazioni sono reperibili per ogni arma all'interno del relativo file ".json" presente nella cartella "BasicRolePlay_TelegramBot/Json/weapons".
Nel caso in cui per qualsiasi motivo si voglia rimuovere un item o una skill da un personaggio, il comando **"/remove_item"** svolge questo compito. Alcuni casi banali di utilizzo possono essere ad esempio un arma che viene rubata, la perdita di una skill dopo anni di non utilizzo, e così via. La logica in ingresso aspettata è del tipo "<character name>, <type of item>: <item to remove>". Specificando quindi il tipo dell'item da rimuovere (ad esempio "weapon", "skill", etc.) la funzione analizza l'inventario del "character" specificato per trovare l'item selezionato e procede alla rimozione salvando gli aggiornamenti nel file ".json"

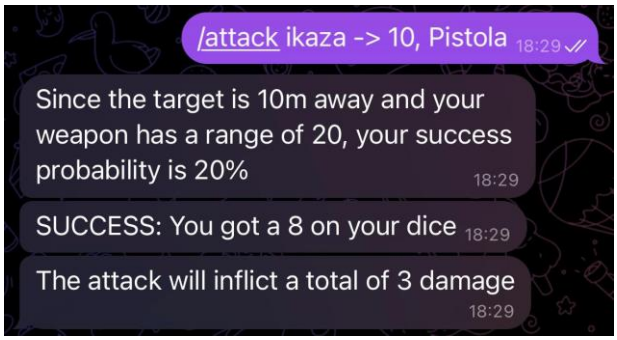
2.7 Gestione combattimento

Nei turni di combattimento l'utente può utilizzare i seguenti comandi:

- **/attack**
- **/evade**
- **/shield**
- **/heal**
- **/remove_hp**

2.7.1 Attack

Calcola l'esito dell'attacco (successo speciale, successo o fallimento) e il relativo danno inflitto. La funzione richiede che l'input abbia il seguente formato: "<attacker name> -> <enemy distance>, <weapon used>". Può dunque essere utilizzata per un attacco contro un NPC; infatti, basta conoscere la distanza dell'avversario (in questo caso il GM può fornire arbitrariamente la distanza del bersaglio) per eseguire l'attacco



2.7.2 Evade

Verifica l'esito del tentativo di schivare un attacco avversario. Richiede che il personaggio schivante possieda l'abilità "Schivare".

Dopo aver calcolato l'esito del comando (successo speciale, successo o fallimento) la funzione si occupa di aggiornare, qualora fosse necessario, gli "hp" del personaggio che subisce l'attacco

2.7.3 Shield

Si occupa di gestire il tentativo di parata dell'attacco avversario. Richiede che il difensore possieda o l'abilità "Scudo" oppure un'arma a corto raggio per parare.

Come per il comando "/evade", viene prima calcolato il tipo di successo o il fallimento e in seguito diminuiti i punti vita del personaggio se necessario

2.7.4 Heal

L'input richiesto dal comando è: "<character name>, <hit points to add>".

La funzione assegna al personaggio in questione i punti salute indicati, assicurandosi che non superino i suoi "hp" massimi

2.7.5 Remove_hp

Nel caso in cui il difensore non possa né parare né schivare l'attacco, sarà costretto a subire tutti i danni. In questo caso viene utilizzato questo comando che permette di sottrarre gli "hp" indicati al personaggio in questione

2.8 Utilizzo abilità e statistiche

2.8.1 Abilità

Il livello delle abilità è espresso con un valore numerico compreso tra 1 e 100. Questo rappresenta la probabilità di successo nell'utilizzo della skill in questione.

Per verificare l'esito di utilizzo di un'abilità viene lanciato un dado a 100 facce: se il risultato è inferiore o uguale al livello si ottiene un successo; in caso contrario si parla di fallimento. Il comando che si occupa del calcolo è **"/ability_roll"**.

Nel caso in cui l'obiettivo sia decidere quale tra due abilità che si scontrano abbia avuto successo si utilizza invece il comando **"/ability_vs_ability"**, la cui sintassi richiesta è: "<character name>: <skill used>, <enemy skill level>".

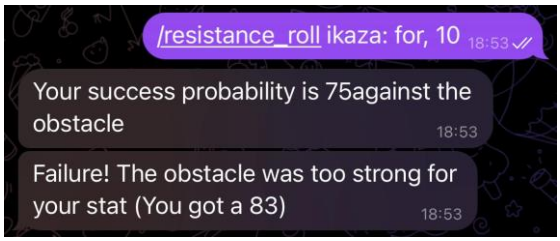
Tramite il solito processo di lancio del dado viene calcolato il tipo di successo o fallimento delle varie abilità e messo a confronto per decidere quale tra le due primeggia (un successo speciale vincerà su un successo "normale", mentre il successo batte il fallimento).

Dopo che un'abilità è stata usata con successo per la prima volta, questa può essere "livellata" col comando **"/level_up_skill"**. Il suo funzionamento è il seguente: viene lanciato un dado "d100" e, se il risultato è maggiore

del livello attuale della skill, questo diventa il nuovo livello. In questo modo col passare del tempo aumentare la propria abilità diventerà sempre più difficile.

2.8.2 Statistiche

Le statistiche sono valori che, in condizioni standard, vanno da 1 a 18 per un essere umano. Questo numero viene convertito in punteggio percentuale qualora fosse necessario verificare l'esito di utilizzo della statistica. La funzione che se ne occupa è **"/stat_roll"**, che richiede come input un messaggio del tipo: "<character name>, <stat to use>". Dopo aver controllato la validità del messaggio, verificando che il personaggio esista ed appartenga all'utente e che la statistica inserita esista, si procede tramite il solito lancio "d100" a decretare il successo speciale, successo oppure fallimento.



Come per le abilità, ci si può trovare davanti ad una situazione in cui la statistica di un personaggio si deve confrontare con un'altra, che sia di un oggetto inanimato oppure un altro player. In questo caso si usa **"/resistance_roll"** per verificare se la statistica "attiva" (quella del nostro personaggio) è riuscita o no a sopraffare quella passiva

3 Conclusione

Il progetto di sviluppo di un bot Telegram per l'assistenza nelle campagne di gioco di ruolo ha rappresentato un'esperienza formativa e sfidante. Dall'ideazione alla realizzazione finale, ogni fase ha contribuito ad arricchire le competenze tecniche e organizzative necessarie per portare a termine un progetto complesso. Grazie a questo lavoro, è stato possibile applicare in maniera concreta i concetti teorici appresi durante il percorso di studi, con un focus particolare sull'interazione tra Python e l'API di Telegram.

3.1 Difficoltà Incontrate

Durante lo sviluppo, sono emerse numerose sfide, molte delle quali legate all'implementazione delle funzionalità richieste dal gioco di ruolo. Un primo ostacolo significativo è stato il bisogno di garantire la corretta gestione delle informazioni di gioco, come le statistiche dei personaggi e i vari oggetti utilizzabili, tramite file ".json". La complessità e la varietà dei dati da gestire hanno richiesto una struttura robusta e ben definita per evitare inconsistenze e garantire una buona esperienza d'uso.

Un'altra difficoltà rilevante è stata la gestione della conversazione con l'utente attraverso il ConversationHandler. Creare un flusso di dialogo che fosse allo stesso tempo intuitivo e funzionale si è rivelato impegnativo, soprattutto nel dover mantenere una coerenza logica tra i vari stati della conversazione. Infine, la necessità di testare e depurare il codice per garantire che ogni comando funzionasse correttamente ha richiesto un'attenzione particolare, portando spesso a dover rivedere e ottimizzare il codice.

3.2 Lezioni Apprese

L'esperienza ha permesso di apprendere e consolidare diverse competenze. Innanzitutto, la gestione dei dati tramite file ".json" in Python si è dimostrata cruciale per il successo del progetto, insegnando l'importanza di una struttura dati flessibile e ben organizzata. Inoltre, il lavoro ha rafforzato le capacità di debugging e "problem solving", essenziali per identificare e risolvere i problemi che inevitabilmente emergono in un progetto di questa portata.

Dal punto di vista dell'interazione utente, è emerso quanto sia importante progettare un'interfaccia conversazionale che sia non solo efficace ma anche intuitiva, facilitando l'utente senza sovraccaricarlo di informazioni. Questa consapevolezza sarà certamente utile in futuri progetti di sviluppo software, dove l'usabilità dell'interfaccia può fare la differenza tra un'applicazione di successo e una poco utilizzata.

3.3 Possibili migliorie

Sebbene il progetto sia completo e rispetti le specifiche descritte nel file di istruzioni per il gioco, potrebbe essere migliorato sotto alcuni aspetti

Ad esempio, al momento l'unica razza supportata è quella umana. Il file contenente le razze potrebbe essere ampliato permettendo la creazione di scenari ancora più ampi. Il processo non dovrebbe essere troppo dispendioso in quanto è stato pensato per essere modulabile e permettere l'aggiunta di nuove regole. Questo è stato possibile grazie all'uso dei file ".json" facilmente aggiornabili e da un codice che si adatta agli stessi

Un'altra idea è quella di aggiungere una lista di scenari preimpostati sotto forma di ".json" per permettere anche all'utente più inesperto di gustarsi l'esperienza "role-play" senza limitazioni. Questi potrebbero essere più o meno corti, con diverse difficoltà, per accontentare una maggiore fetta di pubblico. Si verrebbe così incontro a quei giocatori che non hanno voglia di pensare ad una campagna di gioco da zero, ma preferiscono qualcosa di già ben pensato e senza inconsistenze.

3.4 Considerazioni Finali

In conclusione, il progetto ha rappresentato un'importante opportunità di crescita, sia dal punto di vista tecnico che personale. Le difficoltà incontrate sono state affrontate e superate con determinazione, portando a un prodotto finale che risponde agli obiettivi iniziali e che potrebbe essere ulteriormente sviluppato in futuro. Le competenze acquisite, specialmente in ambito di sviluppo bot, gestione dati e interfaccia utente, saranno fondamentali per affrontare con successo progetti sempre più complessi.