Progetto di Ignegneria informatica

Sviluppo di bot Telegram o Discord in Python
D&D 5e Telegram

Giulio Montuori - 955248
Proff di riferimento: Giovanni Agosta

17 luglio 2023

Indice

1 Specifica

Il progetto consiste nell'implementazione di un supporto digitale per assistere vari `party`, composto da un numero variabile di giocatori, nella loro campagna D&D 5e. Questo supporto viene implementato tramite un bot ospitato sulla piattaforma Telegram, famosa applicazione di messaggistica istantanea.

1.1 Strumenti utilizzati

Per lo sviluppo di questo bot è stato usato il linguaggio di programmazione `Python` con l'aggiunta di `python-telegram-bot`[1] una libreria che agisce da wrapper alle API ufficiali di Telegram costruendo classi e oggetti che permettono una gestione delle richeste `HTTP` e la lettura delle loro risposte ad un livello più alto.

La libreria permette anche di sviluppare il bot fruttando la concorrenza del programma, gestendo quindi più chats e più richeste contemporaneamente. Questo è possibile dalla versione `20.0` di `python-telegram-bot` perché è stata costruita sopra il modulo `asyncio` di `Python`.

Il database permanente è stato implementato tramite una serie di file `.json` divisi per categorie: `parties`, `invites`, `characters`. Questi file vengono aperti all'interno del programma e trasformati in dizionari tramite la libreria `json`. I file `.json` sono ottimi per gestire un database di piccole/medie dimensioni e sono facili da usare all'interno del codice.

1.2 Struttura e Scelte implementative

Il bot[2] è diviso in 2 parti principali: `src` e `JSON`.

1.2.1 SRC

Il codice sorgente, situato nella directory `src`, è diviso in due parti principali: `DataManager` e `dnd`.

- `DataManager`: questo file contiene 4 classi create per gestire i file `.json`, una tra quelle classi, `DataManager`, è astratta e contiene 3 metodi, le altre sono sue sottoclassi. I primi 2 metodi sono implementati e tramite dei lock, usati per garantire corretta sincronizzazione dei dati, salvano i dati aggiornati nel file `.json` o restituiscono l'ultimo `id` utilizzato.
- `dnd`: questo file contiene il `main`, la definizione e l'implementazione di ogni comando. Principalmente è diviso in 4 parti:
  - `characters`: gestito tramite un menù con un `ConversationHandler`[3] e si preoccupa della creazione del personaggio e della sua modifica.
  - `dices`: gestito tramite un menù creato con un `ConversationHandler`[3] e si preoccupa del lancio di uno o più dadi.
  - `parties` e `invites`: gestiti tramite un menù `ConversationHandler`[3] e si preoccupa sia della creazione del party, creando anche inviti, sia della partecipazione ad esso.
  - `currency`: gestito da una serie di comandi per datare hai giocatori maggiore flessibilità nella modifica delle monete e quindi richiedere input più intuitivamente dall'utente.

1.2.2 JSON

All'interno di `JSON` vengono gestiti 3 database che permettono di salvare permanentemente i 3 aspetti principali del gioco: i `parties`, i `personaggi` e gli `inviti`.

- `parties`: questo file contiene tutti i dati di tutti i `party` che gestisce il bot. Il file è strutturato come una lista di oggetti di tipo dizionario, ogni oggetto rappresenta un `party` e i suoi `membri`. In Figura 1 possiamo vedere un esempio di un oggetto `party` nella lista, abbiamo 2 chiavi una rappresenta l'`id` e l'altra una lista di membri con tutti i dati a loro associati
- `invites`: questo file contiene tutti i dati di tutti gli `inviti`. Il file è strutturato come una lista di oggetti di tipo dizionario, ogni oggetto rappresenta un singolo `invito`. In Figura 2 possiamo vedere un esempio di un oggetto `invito` nella lista, abbiamo 4 chiavi che rappresentano rispettivamente l'`id`, la data di scadenza dell'invito, l'`id` del `party` a cui fa riferimento l'invito e infine l'`l'username` dell'invitato. L'`username` non è sempre presente poiché non tutti gli utenti Telegram ne sono provvisti.
- `charachters`: questo file contiene tutti i dati di tutti i `personaggi`. Il file è strutturato come una lista di oggetti di tipo dizionario ogni oggetto contiene una sola chiave `chat_id` che rappresenta l'`id` del giocatore e come valore una lista di `3` oggetti di tipo dizionario e ognuno rappresenta un personaggio. Questo è il file `.json` più complesso e verboso e non verrà mostrato qui, un esempio può essere trovato nella cartella `JSON` nella repository di github[2]

```
{
  "id": 1000,
  "members": [
    {
      "chat_id": 100000000,
      "name": null,
      "character": null,
      "master": true,
      "currency": {
        "copper": 0,
        "silver": 0,
        "electrum": 0,
        "gold": 0,
        "platinum": 0
      }
    },
    {...},
  ]
},
```

Figura 1: Esempio di un oggetto party all'interno del JSON

```
{
  "id": 8500,
  "expiration": false,
  "party_id": 1000,
  "username": 100000000
}
```

Figura 2: Esempio di un oggetto invito all'interno del JSON

2 Funzioni di Dnd_5e

In questa sezione verrà descritta nel dettaglio ogni comando e funzione del bot di telegram [4] ad alto livello vedendo come il bot[4] produce output e richiede input, entrando un po' più nel basso livello, vedendo come e cosa utilizza per produrre output e richiedere input, quando necessario.

Il bot è completamente configurato con le opzioni disponibili da `@BotFather`, è quindi provvisto di descrizioni, about e immagine profilo. L'immagine profilo è stata creata con l'aiuto di una intelligenza artificiale `DALL-E 2` che ha proposto una base con cui l'immagine finale è realizzata, è stato usato il seguente prompt per creare l'immagine di base:

> a comic style profile picture that represent the Dungeons & Dragons 5 edition game

I bot è composto da `8` comandi:

- `/help`: mostra il messaggio di aiuto dove sono scritti tutti i comandi e una breve descrizione.
- `/start`: mostra il messaggio iniziale dove è presente una breve descrizione del bot e una brevissima spiegazione dei 3 menù principali.
- `/party`: mostra il menù per gestire i party e gli inviti.
- `/character`: mostra il menù per gestire i propri personaggi, e dove il Master può modificare e visualizzare i personaggi dei membri del party.
- `/roll`: mostra il menù per gestire il lancio del dado.
- `/show_currency`: mostra in dettaglio quanto Copper, Silver, Electrum, Gold e Platinum ogni membro del party ha nel suo portafoglio.
- `/pay <currency_type> <ammount>`: permette ai giocatori di pagare direttamente dal proprio portafoglio.
- `/add_currency <character> <currency_type> <ammount>` permette al Master di aggiungere currency nel portafoglio di ogni giocatore.

2.1 /help e /start

I comandi `/help` e `/start` utilizzano un `CommandHandler` per chiamare la funzione che si occupa di inviare all'utente il messaggio di aiuto (Figura 3) o di inizio (Figura 3).

Figura 3: Messaggio di help e di start

2.2 /party

Il comando `/party` è gestito con un `ConversationHandler`, ed è il più complesso tra gli altri comandi, e può essere anche rappresentato come una macchina stati finiti (Figura 4). Questo comando ha la funzione di gestire completamente il `party` e il sistema di `inviti`, successivamente segue una spiegazione di ogni stato e come ci si arriva ad esso.

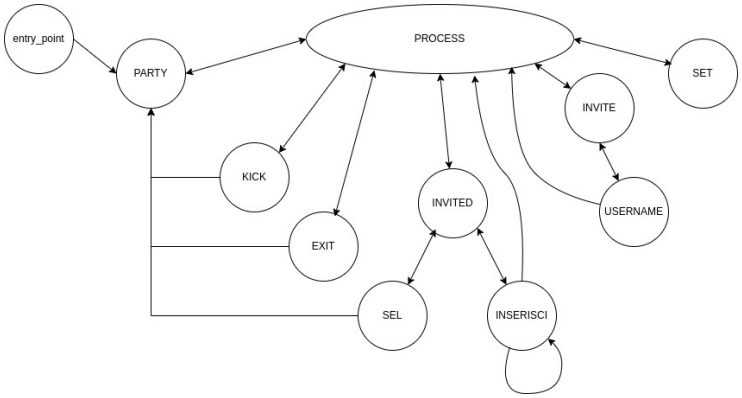

Figura 4: FSM di /party

- `entry_point`: questo stato è quello associato al comando iniziale `/party` ed è l'unico `CommandHandler` tra tutti gli stati. Qui viene chiesto all'utente una ulteriore conferma per procedere con il menù.
- `PARTY`: l'utente è in questo stato se ha confermato la visione del menù dei party nello stato `entry_point` o se è tornato indietro da uno stato successivo con il pulsante `BACK` ed è gestito da un `CallBackQueryHandler`. Questo stato, se l'utente è già un membro di un party, mostra una lista dei

componenti del party con il personaggio da loro selezionato, e una serie di pulsanti, `InlineKeyboardButton`, con le azioni disponibili all'utente in base al suo ruolo all'interno del `party`, ad esempio il master può espellere i membri ma non può impostare un personaggio per la campagna. Se, invece, l'utente non è presente in un party, viene mostrato un messaggio informativo e dei pulsanti, `InlineKeyboardButton`, con l'opzione di creare un `party` o di entrare in uno già creato.

- `PROCESS`: l'utente è in questo stato se ha premuto uno qualsiasi dei pulsanti dello stato `PARTY` o se decide di tornare indietro utilizzando il pulsante `BACK` ed è gestito da un `CallBackQueryHandler`. Come suggerisce il nome, questo stato processa tutti i pulsanti premuti nello stato precedente e prepara tutto il necessario per visualizzare la richiesta e i pulsanti necessari per la fase successiva. `PROCESS` gestisce, quindi, tante mini funzioni quanti tutti i possibili pulsanti premibili da `PARTY`.

  Esempio Se nello stato precedente viene premuto il pulsante `JOIN` vengono mostrati gli inviti in sospeso e pulsanti: uno per visualizzare la schermata per selezionare l'invito, se presenti, uno per inserire un codice di invito, uno per tornate in dietro e altro per annullare.

- `KICK`: Questo stato è raggiungibile solo dal `Master` del `party` ed è gestito da un `CallBackQueryHandeler`. In questo stato viene espulso il membro del `party` e propone i pulsanti per tornare allo stato `PARTY`, `BACK` o annullare tutto `ANNULLA`. È importante notare che è già stato selezionato il giocatore da espellere, poiché si occupa `PROCESS` di creare il messaggio e i pulsanti per la scelta del giocatore.

- `EXIT`: l'utente è in questo stato se in `PROCESS` ha confermato di voler uscire dal `party`, è gestito da un `CallBackQuearyHandler`. In questo stato l'utente esce dal party e se è anche il `Master` il `party` viene eliminato e propone i pulsanti per tornare allo stato `PARTY`, `BACK` o annullare tutto `ANNULLA`.

- `SET`: l'utente è in questo stato se in `Process` è stato selezionato il personaggio da impostare come predefinito per il `party`, è gestito da un `CallBackQuearyHandler`. È importante notare che è già stato selezionato il personaggio da voler impostare, poiché si occupa `PROCESS` di creare il messaggio e i pulsanti per la scelta del personaggio.

- `INVITE`: l'utente è in questo stato se in `PROCESS` è stato selezionato il pulsante `CODE` o `USERNAME`, è gestito da un `CallBackQuearyHandler`. Se si arriva con `CODE` viene generato un codice univoco utile per utenti Telegram sprovvisti di `username`, e anche i pulsanti per tornare a `PROCESS` con `BACK` o annullare tutto con `ANNULLA`. Se si arriva con `USERNAME` vengono generati gli stessi pulsanti del caso `CODE` e chiede all'utente un input di tipo `username` per creare l'invito nel database.

- `USERNAME`: l'utente è in questo stato se in `INVITE` è stato selezionato il pulsante `USERNAME`, è gestito da un `MessageHandler`. In questo stato viene analizzato l'input dell'utente per validarne la correttezza, se scorretto

lo richiede altrimenti crea l'oggetto invito nel databese e comunque in entrambi i casi vengono generati 2 pulsanti, uno per tornare nello stato di `PROCESS` con `BACK` e altro per annullare con `ANNULLA`.

- `INVITED`: l'utente è in questo stato se in `PROCESS` è stato selezionato il pulsante `ACCETTA`, è gestito da un `CallBackQueryHandler`. In questo stato vengono generati, oltre ai pulsanti `BACK` e `ANNULLA`, tanti pulsanti quanti gli inviti ricevuti dall'utente (non è possibile accedere a questo stato se l'utente non ha inviti in sospeso) in modo da selezionare quale accettare.

- `SEL`: l'utente è in questa stato se in `INVITED` è stato selezionato un invito, è gestito da un `CallBackQueryHandler`. In questo stato l'utente viene aggiunto al `party` e il `Master` viene notificato.

È inoltre importante considerare che i `ConversationHandler` sono provvisti di uno stato speciale `fallback` che si può accedere da qualsiasi stato molto simile ad un `resert` di una `FSM`, in questo caso, come anche nei prossimi `ConversationHandler`, lo stato di `fallback` è rappresentato dal pulsante, `InlineKeyboardButton`, `"ANNULLA"` che termina la conversazione.

2.3 /character

Il comando `/character` è gestito da `ConversatinHandler` meno complesso di `/party`, che può essere anche rappresentato come una macchina stati finiti (Figura 5). Questo comando ha la funzione di gestire per ogni utente la creazione la modifica di ogni statistica del proprio personaggio, e al `Master` da la la possibilità di modificare ogni personaggio dei membri del `party`, successivamente segue una spiegazione di ogni stato e come ci si arriva ad esso.

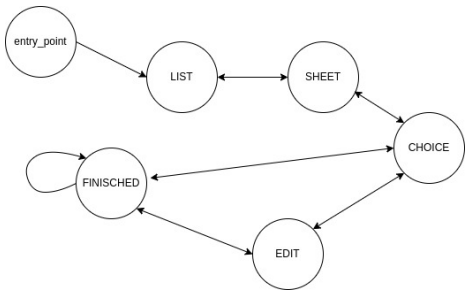

Figura 5: FSM di /character

- `entry_point`: questo stato è quello associato al comando iniziale `/character` ed è l'unico `CommandHandler` tra tutti gli stati. Qui viene chiesto all'utente una ulteriore conferma per procedere con il menù.

- `LIST`: l'utente arriva in questo stato quando conferma la visione del menù nello stato `entry_point` e anche se decide di cancellare un personaggio con i pulsanti `DEL`, è gestito da un `CallBackQueryHandler`. In questo stato è presente un messaggio di introduzione con 3 pulsanti che rappresentano gli

slot dell'utente, e 3 pulsanti per eliminare uno di questi slot e un pulsante per annullare. Se invece l'utente è ha premuto il pulsante `SUPERVISIONE` in `entry_point`, e solo il `Master` ha la possibilità di vedere quel pulsante, appaiono in pulsanti la lista dei personaggi dei membri del `party`.

- `SHEET`: l'utente è in questo stato se ha selezionato uno slot in `LIST`, è gestito da un `CallBackQueryHandler`. In questo stato vengono creati tanti pulsanti quanti le varie categoria che ogni personaggio possiede, in aggiunta al pulsante `BACK` per tornare allo stato `LIST` e `ANNULLA` per chiudere la conversazione.

- `CHOICE`: l'utente è in questo stato se ha selezionato una categoria specifica da visualizzare in `SHEET`, è gestito da un `CallBackQueryHandler`. In questo stato viene mostrato in un messaggio tutte gli attributi di quella categoria e vengono creati i pulsanti `EDIT` e `BACK`. Se la categoria selezionata è `NAME` viene creato un messaggio in cui richiede di inserire il nome con i pulsanti `BACK` e `ANNULLA`.

- |`EDIT`: l'utente è in questo stato se ha selezionato il pulsante `EDIT`, è gestito con un `CallBackQueryHandler`. In questo stato viene mostrato come formattare il messaggio per permettere di modificare con successo l'attributo, dando anche la possibilità di lanciare un dado.

- `FINISCED`: l'utente raggiunge questo stato mandando un messaggio che rispetta la `RegEx` ed è gestito da un `MessageHandler`. In questo stato il messaggio invitato viene processato e se corretto modifica l'attributo ed è pronto per ricevere un altro messaggio o la pressione dei 2 pulsanti `BACK` e `ANNULLA`.

2.4 /roll

Il comando `/roll` è gestito con un `ConversationHandler` ed è il più semplice dei `ConversationHandler`, e può essere anche rappresentato come una macchina stati finiti (Figura 6). Il comando ha la funzione di gestire i vari dati e le varie modalità di lancio, succesivamente segue una spiegazione di ogni stato e come ci si arriva ad esso.

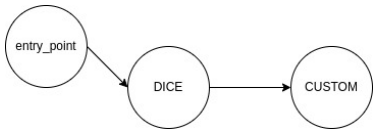

Figura 6: FSM di /roll

- `entry_point`: questo stato è quello associato al comando iniziale `/roll` o `/roll p` ed è l'unico `CommandHandler` tra tutti gli stati. Qui viene generato il menù con tutti i possibili dati da lanciare più i pulsanti `ANNULLA` per chiudere la conversazione e `CUSTOM` per lanciare più dadi in combinazione. L'`entry_point` tramite il comando `/roll p` è esclusivo al `Master` e permette di effettuare un lancio privato, questo perché con il comando `/roll` il risultato viene comunicato a tutti i membri del `party`.

- `DICE`: l'utente arriva in questo stato quando preme un qualsiasi pulsante nello stato `entry_point`, è gestito da un `CallBackQueryHandler`. In questo stato se il pulsante `CUSTOM` è premuto viene preparato il messaggio con la spiegazione del suo funzionamento, altrimenti il dado selezionato viene lanciato e il risultato comunicato in base al tipo di `entry_point`.

- `CUSTOM`: l'utente arriva in questo stato se nello stato `DICE` è stato preparato il messaggio per il pulsante `CUSTOM` in `entry_point`, gestito da un `MessageHandler`, che accetta i messaggi che rispettano le convezioni specificate nel messaggio e vengono controllate da una `RegEx`. In questo stato vengono lanciati i dadi in base alla scelta dell'utente e vengono comunicati in base al tipo di `entry_point`.

2.5 /show_currency

Questo comando è gestito semplicemente da un `CommandHandler` come `/help` e `/start` e crea un messaggio che mostra il portafoglio di tutti i membri del `party`.

2.6 /pay

Questo comando è gestito da un `ComandHandler`, viene usato con 2 parametri `/pay <currency_type> <ammount>` e permette al giocatore di pagare dal proprio portafoglio. Il comando da design non prevede di pagare un altro giocatore, questo per dare più flessibilità al `party` per gestire i pagamenti, dando il compito di aggiungere moneta solo al `Master`.

2.7 add_currency

Questo comando è gestito da un `ComandHandler`, viene usato con 3 parametri `/add_currency <character> <currency_type> <ammount>` e permette solamente al `Master` di aggiungere una quantità `ammount` alla moneta `current_type` al giocatore con il personaggio di nome `character`.

3 Possibili migliorie

Il bot comprende varie funzioni per aiutare il `party` a gestire la loro campagna a distanza, in seguito verranno elencate un serie di funzioni avanzate utili per migliorare e allargare l'esperienza che il bot propone.

- `Livelli`: per ora i livelli sono gesti come ogni altra caratteristica e il salire di livello costringe il giocatore di modificare le caratteristiche coinvolte manualmente, si potrebbe implementare una funziona che gestisce automaticamente la modifica di quelle specifiche caratteristiche.

- `Mappa`: si potrebbe aggiungere un collegamento o creare una mappa di gioco in modo da gestire tutti movimenti del `party` in tempo reale su un mondo visibile e interattivo.

- `Combattimenti`: si potrebbe implementare un sistema che gestisce all'interno del bot i turni e le azioni con un combattimento automatico.

- `Completamente menù`: Per ora solo alcune parti del bot sono gestite con menù e `ConversatioHandler` si potrebbe unificare tutto in una sola interfaccia che gestisce ogni funzione del bot.

4 Conclusioni

Il progetto D&D 5e Telegram Bot ha rappresentato un'opportunità significativa per imparare a programmare in Python. Attraverso lo sviluppo di questo bot, ho potuto esplorare le potenzialità offerte dall'interazione tra l'ambiente di programmazione Python e le API di Telegram.

Il bot, progettato per facilitare il gioco di Dungeons and Dragons 5th edition, ha dimostrato come la tecnologia possa essere utilizzata portare un gioco da tavolo prevalentemente giocato in persona a distanza. Le funzionalità implementate, tra cui la gestione del party, la creazione del personaggio e il lancio dei dadi, forniscono un valido supporto ai giocatori, permettendo loro di concentrarsi maggiormente sull'aspetto narrativo e strategico del gioco.

Riferimenti bibliografici

[1] Repository ufficiale su github (link)

[2] Repository contenente il progetto sviluppato (link)

[3] Documentazione ufficale di python-telegram-bot (qui)

[4] Bot Telegram con username @Dnd_5d_BR_bot o al link Dnd_5e