Programma Python: documentazione riguardante il programma sull'algoritmo di risoluzione del Cubo di Rubik

Michele Sallaku & Gabriele Bier

Dipartimento di Sistemi intelligenti, Università degli Studi di Brescia, via Piazza del Mercato, 15, 25121 Brescia BS, Italia

Keywords: Cubo di Rubik, Algoritmo intelligente, Pygame, Esecuzione e risoluzione

e-mail: m.sallaku@studenti.unibs.it , g.bier@studenti.unibs.it

ABSTRACT: Il Cubo di Rubik è un popolare rompicapo tridimensionale inventato nel 1974 dall'architetto ungherese Ernő Rubik. È composto da un cubo di dimensioni 3x3x3, suddiviso in 6 facce colorate. Ogni faccia è formata da una griglia di 9 quadrati più piccoli, per un totale di 54 quadrati. Le sei facce sono colorate con sei colori diversi e l'obiettivo del puzzle è di ripristinare l'ordine originale dei colori sulle facce del cubo, che viene mischiato ruotando le sue sezioni. Ci sono diverse configurazioni possibili per il Cubo di Rubik, ma solo una soluzione corretta. Per risolvere il cubo, è necessario fare una serie di mosse strategiche e studiate per spostare i pezzi colorati nelle posizioni corrette. Il Cubo di Rubik richiede logica, strategia e pazienza per essere risolto. Negli anni sono state sviluppate diverse metodologie e algoritmi per risolvere il cubo in maniera efficiente e veloce. Il programma ideato e scritto ha permesso di simulare tramite un'interfaccia resa disponibile dalla libreria Pygame un Cubo di Rubik al fine di mischiarlo e risolverlo per scopi didattici: il tutto seguendo un algoritmo di risoluzione intelligente che ha reso possibile l'applicazione e il controllo del cubo lungo la sua risoluzione.

Applicazioni e Librerie utilizzate

Software Visual Studio Code per la stesura del programma

Libreria Pygame per lo sviluppo del "Gioco"

Libreria dataclasses per la memorizzazione di valori che sarebbero stati utili ad identificare il singolo movimento

Libreria typing al fine di rendere più facile la lettura e gestione delle variabili e delle istanze all'interno del codice

Libreria intertools per la gestione delle possibili permutazioni degli sticker posizionati sui quadrati del cubo

Libreria time per gestire lo sleep del movimento

Libreria copy per generare "copie profonde" dell'oggetto cubo e degli oggetti interni ad esso per la loro risoluzione

Libreria Openpyxl per l'apertura di file excel per le statistiche

Libreria Matplotlib per la gestione del grafico delle statistiche

Introduzione

Un Cubo di Rubik è essenzialmente una matrice tridimensionale che presenta 6 facce distinte ognuna dall'altra da 6 colori differenti (BIANCO, VERDE, ARANCIO, BLU, ROSSO, GIALLO); la rappresentazione dello stesso potrebbe essere eseguita utilizzando un software complesso che permette di gestire in modo più pesante ma diretto ogni singolo cubetto dei 26 che compongono il Cubo ma, data la difficoltà nella gestione e realizzazione di un'interfaccia 3D, il gruppo ha pensato di rappresentare il cubo disponendo ogni matrice colorata in un piano di lavoro bidimensionale.

Documentazione del codice

Modulo colore.py

Il modulo colore.py è stato realizzato in modo tale da poter definire semplicemente e velocemente i colori riguardanti il cubo di Rubik nell'intero codice scritto.

Tramite il metodo **Colore = NewType("Colore", Tuple[int,int,int])** è stata costruita una nuova tipologia di variabili definita Colore che associa un ipotetico valore in **formato RGB** al colore rappresentativo. Una volta preimpostata questa nuova tipologia di variabili è stato necessario definire delle istanze che rappresentassero i 6 colori possibili presenti in un normale cubo di Rubik. Infine, abbiamo definito una mappa che descrivesse e accoppiasse ogni faccia ad un colore possibile utilizzando il metodo **MAPPA_COLORI_FACCE = [("U", WHITE), ("F", GREEN), ("L", ORANGE), ("B", BLUE), ("R", RED), ("D", YELLOW)]**.

Tutto questo per rendere più semplice l'identificazione dei colori all'interno del programma steso.

Modulo movimenti.py

La classe movimenti è stata scritta per permettere di salvare e memorizzare in modo automatico le possibili combinazioni che identificano i movimenti eseguiti sul cubo di Rubik. Utilizzando il decoratore **@dataclass** è stata istanziata una classe Movimenti che presenta tre attributi utili a identificare la **faccia** interessata nel movimento, un valore booleano che definisce se il movimento deve essere **invertito** e un ultimo valore booleano che definisce se il movimento è da eseguire **due volte** consecutive.

Modulo pezzi.py

Questo modulo permette di definire un nuovo tipo di variabili utili alla gestione del cubo: **Angolo** e **Bordo**, che presentano a loro interno una stringa identificativa e un dizionario che associa una chiave anch'essa testuale ad un colore che rappresenta il pezzo.

Successivamente vengono definiti due dizionari di conversione utili a trasformare un angolo o un bordo in una sequenza di movimenti utili alla risoluzione predisposta dall'algoritmo in esame. **BORDO_A_UF** e **ANGOLO_A_UFR** sono due dizionari che, ricevuta una stringa identificativa dell'angolo o del bordo, ritornano un esatto movimento utile all'algoritmo.

Modulo risolutore.py

La classe pensa esclusivamente alla realizzazione della soluzione del cubo di Rubik. **Genera_soluzione()** riceve in ingresso il cubo in esame e ritorna una sequenza di movimenti utili a risolvere il cubo passo per passo secondo l'algoritmo di risoluzione che compone lo storico del cuboStorico. Ogni metodo successivamente implementato compone l'algoritmo intelligente che risolve e gestisce il cubo lungo la sua risoluzione. I passi di soluzione sono concatenati: solo una volta dopo aver risolto il passo corrente dell'algoritmo il sistema può proseguire con la soluzione, altrimenti il sistema continua a iterare fino alla risoluzione del passo per poi passare al successivo fino alla soluzione completa.

Modulo cubo.py

Il modulo rappresenta lo scheletro che permette a tutto il programma di funzionare: consente al cubo di esistere e di effettuare operazioni sullo stesso. Il metodo costruttore (**__init__**) consente di definire le dimensioni del cubo e di inizializzare le possibili facce del cubo con chiamata a metodo secondario. **get_sticker()** permette di rilevare il colore dello sticker posizionato su un quadrato del cubo tramite distinzione sul tipo di quadrato in esame. I metodi **get_bordo** e **get_angolo** rilevano il bordo o l'angolo indicativamente dal pezzo del cubo che viene passato alla funzione. Il primo metodo riguardante i movimenti (**esegui_movimenti**) consente di eseguire una serie di movimenti sul cubo; **e_risolto()** è la funzione che controlla se ogni pezzo di ogni faccia è risolto e ritorna un booleano. Il metodo **genera_faccia()** consente di generare una singola faccia del cubo mentre **ruota_faccia()** esegue operazioni di inversione e trasposizione al fine di ruotare la faccia passata al metodo stesso. **modifica_faccia_adiacente()** è il metodo principale che gestisce l'intero cubo quando viene eseguito un movimento che comporta eventuali modifiche delle facce adiacenti alla faccia in movimento; il metodo **ruota()** esegue la rotazione di una faccia in base alle caratteristiche del movimento necessario e **ruota_verticalmente()** esegue operazioni al fine di eseguire una effettiva rotazione verticale. Infine, **trasposizione()** è un metodo risultato utile all'effettuare trasposizioni.

Modulo cubo_storico.py

Questo modulo risulta essere una traccia passata dei movimenti sul cubo di Rubik. Essendo una classe "figlia" della classe Cubo invoca lo stesso metodo costruttore per i propri attributi ma, nel caso corrente, associa nel caso possibile le facce disponibili e inizializza uno storico di sistema. Viene definito un metodo **get_lista_mov_storici** al fine di ritornare le sequenze di movimenti nella storia del cubo; i due metodi successivi (**get_bordo** e **get_angolo**) sono identici alla classe padre mentre invece, l'ultimo metodo ideato (**esegui_movimenti**), permette di salvare lo storico dei movimenti in modo controllato da chiamata.

Modulo interfaccia.py

Il modulo interfaccia.py è stato implementato al fine di controllare e gestire l'interfaccia utente durante l'esecuzione e l'animazione della risoluzione del cubo di Rubik. L'interfaccia basa principalmente i suoi metodi e funzioni sulla libreria Pygame che permette di gestire situazioni più dinamiche come i movimenti del cubo. Come prima cosa sono state create delle variabili al fine di dimensionare la finestra dell'interfaccia grafica.

Una volta impostate le dimensioni volute è stato necessario definire la classe che gestisce la vera e propria interfaccia: class Gui, la quale a sua volta presenta i principali metodi importanti e utilizzati all'interno del modulo e dei moduli che la istanziano.

La classe **Gui** prevede come metodo principale un metodo che istanzia all'interno della classe una copia esatta di un cubo che gli viene passato nel metodo costruttore **__init__(self, Cubo: cubo)** e che imposta la altezza e la larghezza della interfaccia secondo il metodo offerto dalla libreria Pygame. Il secondo metodo (**esegui**) presente nella classe Gui permette di eseguire l'interfaccia e di effettuare gli eventuali controlli sui tasti premuti al fine di esercitare operazioni sul cubo che gli viene passato in modo tale da modificarlo o risolverlo e animarlo. L'ultimo metodo (**disegna_cubo**) permette di disegnare l'effettivo cubo di Rubik all'interno della finestra dell'interfaccia andando a ciclare ogni singola faccia presente e ogni singola riga presente nella faccia; una volta arrivato all'esatta posizione del quadrato in esame viene eseguito un controllo riguardante la faccia che comprende quel quadrato e viene spostato sull'interfaccia in base alla posizione corretta graficamente parlando.

Modulo controllore.py

Il modulo controllore.py mette a disposizione delle altre classi dei metodi utili a controllare le singole sequenze di movimenti da eseguire o utili a risolvere il cubo di Rubik in esame. La prima funzione disponibile (**trasf_in_movimenti**) permette di trasformare in una lista di movimenti una stringa passata composta da caratteri identificativi. Successivamente è stata realizzata una funzione inversa (**trasf_in_stringa**) che trasforma una lista di movimenti in una stringa che definisce i movimenti da eseguire. Infine, è stato realizzato un metodo (**inverti_movimenti**) che permettesse di invertire una sequenza di movimenti passati alla funzione in modo tale da ottenerne una lista invertita e ritornarla.

Modulo ripulitore.py

Il modulo ripulitore.py permette di gestire le singole stringhe di movimenti al fine di "ripulirle" e ricomporle per la loro successiva gestione all'interno del codice.

Modulo generatore.py

Il modulo generatore.py è una classe implementata al fine di controllare e generare una sequenza di movimenti randomici atti al mischiare nel modo più possibilmente casuale il cubo di Rubik. Il primo metodo (**gen_n_seq_mov_casuali**) permette di generare una serie lunga **n** di movimenti casuali invocando il metodo descritto successivamente (**gen_seq_mov_casuali**), il quale permette infatti di generare una sequenza composta da 40 movimenti randomici a discrezione del programmatore.

Modulo stats.py

La classe stats.py è stata realizzata al fine di avere una visione generale di quanto efficace sia il programma e l'algoritmo che abbiamo implementato. Una volta passati al metodo costruttore (**__init__self, cubo: Type[CuboStorico], mischiare: Callable, risolutore: Callable**) i principali metodi e classi utili al modulo intero, si è potuto procedere alla definizione di un vero e proprio libro di lavoro su cui poter scrivere e memorizzare informazioni sull'algoritmo. Il metodo **gen_titoli** permette di definire i titoli delle colonne del foglio di lavoro che andrà riempito mentre il metodo **genera_statistiche** permette di generare tante statistiche quante decise dal programmatore (200 risoluzioni), **genera_grafio_dispersione** permette di creare al fine di vedere esattamente le potenzialità dell'algoritmo realizzato e infine il metodo **apri_workbook** consente di aprire e caricare il foglio di lavoro per le statistiche da rappresentare e salvare.

Modulo main.py

Il modulo main.py utilizza una convenzione scoperta comune all'interno del linguaggio Python in quanto applica un controllo riferito allo script di esecuzione del programma (**if __name__ == "__main__"**:). Una volta eseguito il controllo definito non resta che definire la dimensione del cubo da generare, passarlo alla funzione riguardante l'interfaccia utente e infine eseguire l'intera interfaccia descritta.