Esempio di sviluppo di un progetto

Giulio Iannello

September 8, 2022

1 Fasi preliminari

Nella prima fase sono state svolte la azioni preliminari indicate nella "Guida per la configurazione e lo sviluppo di un'progetto".

Si è poi proceduto a implementare le funzionalità richieste in modo graduale per piccoli passi. In molti casi si sono introdotte funzionalità minimali o anche la sola interfaccia di una classe, un metodo o una funzione rinviando a passi successivi un'implementazione più completa. Dopo ogni passo, prima di effettuare un commit sono stati effettuati dei test per verificare la correttezza di quanto fatto.

Nel caso si siano verifcati errori, essi sono stati corretti prima del commit, mantre, nel caso i test abbiano evidenziato la necessità di estendere la soluzione introdotta che però aveva superato i test, tale estensione è stata oggetto del passo successivo con un commit separato.

Nel seguito di questo documento, per ciascun commit effettuato vengono riportate alcune condierazioni che completano quanto di può ricavare dall'esame del codice e dei commenti. In paticolare sono riportati i casi di test utilizzati che mostrano come essi si vadano arricchendo man mano che vengono introdotte nuove funzionalità o vengono completate quelle introdotte in misura parziale.

2 Specifica

Il programma deve implementare il gioco della tombola.

In particolare deve:

- poter estrarre a sorte i numeri da 1 a 90;
- poter creare gruppi di 6 cartelle con le caratteristiche specificate di seguito;
- poter registrare N giocatori;
- poter assegnare un numero M di cartelle a ciascun giocatore (ogni giocatore può avere assegnate un numero diverso di cartelle, per semplicità si può supporre che M non possa essere maggiore di un valore predefinito);
- poter verificare se un numero estratto è presente in una cartella o nel cartellone che comprende tutti i numeri elencati consecutivamente e divisi in sei sotto-cartelle di 15 numeri ciascuna;
- se il numero è presente, verificare se si è fatto ambo, terna, quaterna, cinquina o tombola, se uno di questi risultati risulta non ancora raggiunto;

- iniziare il gioco estraendo un numero alla volta e verificando ogni volta se qualche giocatore o il tabellone ha realizzato un ambo, una terna, una quaterna, una cinquina o una tombola; per semplicità si può supporre che al più un giocatore può realizzare uno di questi risultati dopo ogni estrazione;
- terminare il gioco quando un giocatore o il tabellone ha realizzato una tombola.

Un gruppo di 6 cartelle deve verificare le seguenti condizioni:

- ogni cartella ha 3 righe e 9 colonne;
- ogni cartella ha 15 caselle marcate da un numero da 1 a 90 e 12 caselle vuote;
- devono esserci esattamente 5 numeri su ogni riga;
- devono esserci da 1 a 3 numeri su ogni colonna;
- la prima colonna contiene i numeri da 1 a 9, la seconda i numeri da 10 a 19, la terza i numeri da 20 a 29,... la nona colonna contiene i numeri da 80 a 90;
- quando presente in una cartella, il numero 90 occupa sempre la casella nell'angolo in basso a destra (riga 3, colonna 9);
- ogni numero da 1 a 90 deve essere presente su una e una sola cartella.

2.1 Modalità di gioco

Per giocare digitare da linea di comando:

```
python tombola.py -g numero_giocatori -n lista_numero_di_cartelle_per_giocatore
```

Ad esempio il comando:

```
python tombola.py -g 3 -n 3 6 4
```

Avvia il gioco con 3 giocatori che hanno rispettivamente 3, 6 e 4 cartelle.

Per elencare tutte le opzioni possibili digitare:

```
python tombola.py -h
```

Premere `return` per estrarre un numero. Dopo ogni numero viene visualizzato il corrispondente risultato per ciascun giocatore.

3 Commit iniziale

Da un esame della specifica di può concludere che il gioco della Tombola richiede l'esecuzione di una seria di passi. È stata pertanto abbozzata la struttura del programma principale sotto forma di commenti.

Questa attività ha evidenziato i seguenti concetti che possono suggerire l'introduzione di alcune classi:

- giocatore
- cartella
- tabellone

- gruppo di cartelle (che verificano i vincoli)

Inoltre inizialmente occorre ricavare dalla linea di comando il numero di giocatori e il numero di cartelle da assegnare a ciascun giocatore.

Poiché sono stati utilizzati solo commenti non sono stati introdotti casi di test.

4 Secondo commit

Acquisizione da riga di comando del numero di giocatori e del numero di cartelle per giocatore.

Casi di test:

```
python Tombola
python Tombola -n 3 2
python Tombola -n 3 2 -g 3
```

Il terzo test pur non generando errori, ha rivelato che non è gestito il caso in cui il numero dei giocatori non corrisponde alla lunghezza della lista di cartelle da assgnare. La soluzione a questa carenza sarà oggetto del commit successivo in quanto la funzionalità di acquisire i dati dalla linea di comando è stata correttamente implementata.

5 Terzo commit

Verifica che i dati da linea di comando siano coerenti e segnalazione nel caso non lo siano.

Casi di test:

```
python Tombola
python Tombola -n 3 2
python Tombola -n 3 2 -g 3
python Tombola -n 3 2 4 -g 3
```

È stato introdotto un ulterire caso di test.

6 Quarto commit

Riesaminando la struttura del programma principale si è osservato che prima di poter estrarre un numero occorre disporre del "sacchetto". Si è pertanto integrata la struttura del programma principale con il commento:

```
# Crea sacchetto da cui estrarre i numeri
```

che non era presente nella prima bozza. Di conseguenza alle classi identificate in precedenza si è aggiunta la classe:

- sacchetto

Sono state quindi create le classi:

- Sacchetto
- Tabellone

- Cartelle - Gruppo_cartelle
- Giocatore

in quattro file distinti. Per ciascuna classe sono stati definiti i metodi che vengono richiesti dalla striuttura del programma principale con un implementazione minimale, spesso vuota.

Sono state inoltre abbozzate alcune fasi del programma principale che richiamano i metodi delle classi su elencate.

Non sembra necessario per il momento aggiungere altri casi di test.

7 Quinto commit

È stato ulteriormente raffinato il programma principale implementando la distribuzione delle cartelle dal monento che tale funzione richiede solo la conoscenza delle informazioni passate sulla linea di comando.

Sono state introdotte delle stampe per effettuare il testing.

8 Sesto commit

Definita la rappresentazione di `Tabellone` che consiste in due matrici 18x5. Il tabellone è costituito da 6 gruppi di 3 righe ciascuno, ogni riga contiene 5 numeri.

La prima matrice è introdotta nel caso si voglia visualizzare il tabellone. La seconda consente di ricordare i numeri estratti e di verificare se si è ottenuto un risultato:

- per verificare se si è fatto ambo, terna, quaterna o cinquina basta sommare gli elementi di una riga;
- per verificare se si è fatto tombola occorre che tutte le 3 righe di un gruppo abbiano fatto cinquina.

Implementato il metodo `segna_numero` di `Tabellone`.

Per effettuare un test sono state introdotte delle stampe nei metodi di `Tabellone` che sono state poi lasciate commentate una volta verificata la correttezza. Si noti che per testare il metodo `segna_numero` di `Tabellone` e stato necessario che il metodo `estrai` di Sacchetto restituisse un qualunque numero da 1 a 90.

9 Settimo commit

Implementata la classe `Sacchetto` usando il generatore visto a lezione leggermente migliorato.

Modificato il metodo `segna_numero` di `Tabellone` per restituire *tombola* dopo un certo numero di estrazioni. Si noti che la condizione usata non è corretta, ma provvisoria per consentire un rapido test. La condizione sarà raffinata a un commit successivo.

Introdotta la funzione `is_migliore` nel programma principale per confrontare i risultati dal tabellone e dai giocatori.

10 Ottavo commit

È stata implementata correttamente la verifica del risultato ottenuto dal tabellone (metodo `segna_numero`). Per verificare se si è fatto tombola oltre alla somma dei risultati sulla riga aggiornata si è calcolato la somma dei risultati del gruppo di righe cui tale riga appartiene.

Poiché non è banale introdurre stampe di facile esame per verificare la correttezza di tale implementazione, essa è stata verificata tramite debugger, esaminando cosa accadeva per i primi numeri estratti.

È stato quindi aggiornato il programma principale per visualizzare i risultati ottenuti.

I casi di test utilizzati in precedenza sono ancora sufficienti. Essi non hanno dato luogo a errori, ma hanno evidenziato che vengono riportati tutti i risultati ottenuti sul tabellone a prescindere se essi sono stati già ottenuti. Il miglioramento di questo comportamento (segnalare un risultato solo se non è già stato ottenuto) sarà oggetto di un commit successivo.

11 Branch `prove`

Volendo capire dopo quiante estrazioni il tabellone fa tombola è necessario modificare il programma principale.

Non volendo includere tale funzione nel programma finale si è deciso di creare un branch da utilizzare per effettuare prove.

Nel nuovo branch è stato aggiunto un contatore del numero di estrazioni effettuate il cui valore viene visualizzato quanto si fa tombola. È stato quindi effettuato un commit sul branch `prove`.

Si torna quindi al branch principale per riprendere lo sviluppo.

12 Branch `main`, nono commit

Implementato nel programma principale il controllo che eventuali nuovi risultati migliorano il risultato precedente. È stata modificata la funzione `is_migliore` che ora verifica solo se il nuovo risultato è migliore del precedente.

Il controllo viene effettuarto anche per i giocatori, anche se al momento l'implementazione del metodo `segna_numero` della classe `Giocatore` è banale (restituisce sempre il risultato "nullo").

13 Decimo commit

Definita la rappresentazione della classe `Giocatore` e implementato il metodo `segna_numero`.

Aggiunto metodo `segna_numero` alla classe `Cartella`.

Aggiornati alcuni commenti in vari file.

14 Undicesimo commit

Definita la rappresentazione della classe `Cartella`.

Implementato il metodo `segna_numero`. Rimandato il testing perché tutte le cartelle sono vuote. Il testing andrà effettuato dopo aver implementato la generazione delle cartelle.

15 Branch `develop`

Resta da implementare la generazione delle cartelle secondo i vincoli imposti dalla specifica. Si tratta di un compito non banale che può richiedere più passi e diverse fasi di testingù/debugging. Per questo motivo è preferibile aprire un branch di sviluppo in modo da lasciare nel branch principale un'implementazione funzionante che può essere anche leggermente migliorata in parallelo allo sviluppo della generazione delle cartelle.

Dopo un esame attento del problema si è individuata la seguente strategia per risolverlo:

- si individuano prima le caselle delle cartelle su cui posizionare i numeri in modo da rispettare i vincoli successivamente si assegnano i numeri alle caselle precedentemente individuate; la prima fase si scompone in tre sottofasi;
- *prima sottofase*: nella prima cartella si blocca la posizione (0,0), poi le posizioni (1,1), (2,2), (0,3), (1,4), ..., (2,8); si sipete la cosa nella cartella successiva iniziando da (1,0); si ripete poi nella terza partendo da (2,0) e così fino alla sesta cartella; a questo punto sono state bloccate 54 posizioni e ne restano da posizionare 36 di cui 3 nella prima colonna, 4 nelle colonne dalla seconda all'ottava, 5 nella nona colonna;
- *seconda sottofase*: partendo da una cartella scelta a caso si bloccano in cartelle successive le 3 posizioni della prima colonna; si ripete il procedimento per le posizioni rimaste delle colonne successive, partendo sempre dalla cartella successiva all'ultima considerata;
- al termine della seconda sottofase: (i) poiché dopo la prima sottofase tutte le cartelle hanno esattamente una casella bloccata in ogni colonna il vincolo sulle colonne è soddisfatto, (ii) dopo la prima sottofase c'è almeno una cartella che ha boccato la posizione (2,8) che va obbligatoriamente riservata al numero 90 (iii) poiché dopo la prima sottofase tutte le cartelle hanno bloccato 9 posizioni, e poiché nella seconda sottofase le caselle sono state bloccate considerando una cartella alla volta, dopo la seconda sottofase tutte le cartelle hanno bloccato 15 posizioni; pertanto anche tale vincolo è soddisfatto; non è invece in generale soddisfatto il vincolo sulle righe;
- *teza sottofase*: nel caso in una cartella il vincolo sulle righe non sia soddisfatto, si procede a spostare le posizioni bloccate dalle righe che hanno più di 5 posizioni bloccate a quelle che ne hanno meno fino a soddisfare il vincolo; in questa sottofase occorre fare attenzione a non spostare la posizione (2,8) in una delle cartelle dove essa è bloccata;
- infine si procede ad assegnare i numeri da 1 a 90 alle posizioni bloccate; si procede preliminarmente ad assegnare 90 alla cartella dove la posizione (2,8) è bloccata; successivamente, per garantire una distribuzione casuale, si parte dai numeri da 1 a 9 in ordine casuale; si passa poi ai numeri da 10 a 19, sempre in ordine casuale e così via fino ai numeri da 80 a 89.

16 Branch `develop`, primo commit

Definita la strategia di generazione di un gruppo di cartelle che verifichi i vincoli di specifica si procede alla sua implementazione. Si noti che la strartegia sopra descritta è descritta anche in un

commento del metodo `genera_gruppo` della classe`Gruppo_cartelle` in modo da rendere il codice auto-esplicativo. La strategia si compone di due fasi, la prima delle quali è a sua volta decomposta in tre sottofasi.

Implementata la prima sottofase della prima fase che include il nuovo metodo `blocca_posizione` della classe `Cartella`. È stata anche leggermente modificata la rappresentazione della cartella nella fase di generazione.

17 Branch `develop`, secondo commit

Implementata la seconda sottofase della prima fase di generazione di un gruppo di cartelle. L'implementazione ha richiesto una ridefinizione del metodo `blocca_posizione` della classe `Cartella`.

Aggiunto un generatore di numeri da 0 a n-1 nel file sacchetto.py.

Aggiunto un metodo ausiuliario alla classe `Gruppo_cartelle` per facilitare il testing.

18 Branch `develop`, terzo commit

Implementata la terza sottofase della prima fase di generazione di un gruppo di cartelle.

Poiché nella terza sottofase le cartelle devono essere modificate una ad una, l'implementazione ha richiesto una ridefinizione del metodo `check_vincoli`. Il metodo si occupa ora di controllare che una determinata cartella verifichi i vincoli, ed è stato introdotto il nuovo metodo `check_vincoli_gruppo` che si occupa invece di controllare che tutte le cartelle di un gruppo verifichino i vincoli.

La logica con cui è stata implementata la terza sottofase è spiegata dai commenti nel codice.

19 Branch `main`, dodicesimo commit

Sebbene non sia terminata la generazione delle cartelle secondo i vincoli, si è deciso intanto di completare l'implementazione del gioco fermando l'esecuzione dopo ogni estrazione e aspettando che l'utente prema il tasto INVIO prima di estrarre il numero successivo.

Si è quindi tornati al branch `main`, si è effettuata la modifica, seguita da un commit.

20 Branch `develop`, quarto commit

Implementata la seconda fase di generazione di un gruppo di cartelle.

La logica con cui è stata implementata la seconda fase è spiegata dai commenti nel codice.

Aggiunto il metodo `stampa` alla classe `Cartella` e alla classe `Tabellone` e il metodo `stampa_cartelle` alla classe `Giocatore` per verifica.

Modificato il programma principale per segnalare chi a ottenuto un risultato e per visualizare lo stato del tabellone e delle cartelle.

21 Branch main, merge da `develop`

Avendo completato l'implementazione della generazione di un gruppo di cartelle, si procede sul branch `main` a un merge da `develop`.

Il merge dà luogo a conflitti sul programma principale (su entrambi i branch si sono fatte modifiche al file `Tombila.py`). I conflitti vengono risolti manualmente. Si tratta in realtà di un falso conflitto e possono essere mantenere le righe da entrambi i branch.

Una volta eliminata la segnalazione di conflitto dal codice, con il comando:

```
git add Tombola.py
```

si indica che il conflitto è risolto e si può procedere al commit che chiude il merge.

22 Creazione del repository remoto su GitHub

Per portare il progetto su GitHub bisogna:

- creare un repository vuoto su GitHub dandogli un nome (nel nostro caso `esempio_progetto`;
- eseguire nella directory del progetto i seguenti comandi:

  ```
  git remote add origin https://github.com/iannellog/esempio_progetto.git
  ```

  ```
  git push --all origin
  ```

Il primo comando collega il repository locale a quello remoto, mentre il secondo comando popola il repository remoto effettuando il push di tutti i branch.

Attenzione: per eseguire il secondo comando occorre autenticarsi per dimostrare di avere i diritti di modifica su GitHub.

23 Commit finali

Con il comando:

```
pip freeze > requirements.txt
```

è stato creato un file che contiene i moduli python non standard necessari al progetto (nel nostro caso solo `numpy`.

Sono stati eseguiti alcuni commit finali per aggiungere ai file tracciati una sottodirectory `doc/` che contiene questa documentazione in formato pdf.

Erroneamente in un primo momento si era esclusa la sottodirectory `doc/` tramite .gitignore e successivamente la si è reinserita e la si è dovuta tracciare aggiungendola all'area di staging con il comando:

```
git add doc/
```