Relazione progetto TMB: una tombola distribuita

Corso di Programmazione e Analisi di Dati, primo modulo - AA 2021-22
Irene Mondella - 584285

Sommario

Introduzione

Il mio programma implementa una semplice versione del gioco della tombola, impostato secondo una logica produttore-consumatore: il produttore è il banditore che estrae i numeri e li comunica ai consumatori, e ogni consumatore è un diverso giocatore, che ha una cartella e legge i numeri estratti.

Nello specifico, il programma contiene cinque file contenenti ciascuno una diversa classe: il file Banditore.py contiene una classe thread del produttore (Server), Giocatore.py contiene una classe thread del consumatore (Player), il file Partita.py contiene la classe dell'oggetto condiviso Game, il file main.py contiene la classe Interface che implementa l'interfaccia grafica del gioco, il file Test.py, infine, contiene la classe Test che permette di effettuare semplici test su diverse parti del codice.

Classe Game

La classe Game permette di creare una nuova partita: contiene infatti gli oggetti condivisi necessari per far comunicare il banditore e i giocatori, nonché diversi oggetti Lock, Condition o Event del modulo threading per accedere alle risorse condivise in mutua esclusione. Nello specifico, il costruttore inizializza:

- numbers: una coda di un elemento, in cui il banditore mette di volta in volta l'ultimo numero estratto e da cui i giocatori lo leggono;
- game_lock: un oggetto Condition necessario per accedere alla coda in mutua esclusione;
- estratti: una lista vuota a cui verranno aggiunti numeri man mano che vengono estratti;
- global_wins: una matrice con una riga per ogni tipo di vincita (ambo, terno, quaterna, cinquina, decina e tombola); ogni riga contiene:
    - il nome della vincita in formato stringa;
    - un intero che tiene conto di tutti i giocatori che effettuano quella vincita contemporaneamente;
    - una lista degli ID dei giocatori che effettuano quella vincita contemporaneamente;
    - un flag che indica se quella vincita è già stata effettuata a un'estrazione precedente: è False finché la vincita non è ancora stata effettuata, poi diventa True;
    - il montepremi corrispondente;

- wins_lock: un oggetto Lock necessario per accedere alla matrice in mutua esclusione;
- victory_event: un oggetto Event necessario per sincronizzare le azioni effettuate da banditore e giocatori sulla matrice condivisa in caso di vincita;
- players_read_number: un contatore che si aggiorna ogni volta che un giocatore legge l'ultimo numero;
- counter_lock: un oggetto Lock necessario per accedere al contatore players_read_number in mutua esclusione;
- awards: un dizionario che conterrà il montepremi in denaro di ogni giocatore man mano che egli effettua delle vincite.

La classe Game contiene, inoltre, i seguenti metodi:

- il metodo write prende come parametro un numero per metterlo in coda: per far ciò, acquisisce il game_lock, se la coda è già piena aspetta, altrimenti vi inserisce il numero, notifica gli altri thread in attesa e rilascia il game_lock.
- Il metodo read legge, usando il metodo get della classe predefinita Queue, il numero messo in coda, e lo restituisce; se però la coda è vuota (perché un altro thread giocatore ha già letto il numero e svuotato la coda), il metodo read va a leggere l'ultimo numero della lista estratti. Se anch'essa è vuota (cosa che accade solo alla prima estrazione) oppure se il suo ultimo numero è uguale a quello appena letto dal giocatore, allora il thread si mette in attesa finché la coda non verrà riempita.
- Il metodo locked_win controlla, dopo aver acquisito il wins_lock, il flag della tombola contenuto nella matrice global_wins, per capire se è stata fatta tombola oppure no.
- Il metodo check_victories controlla la matrice global_wins per vedere se ci sono state vincite: in tal caso aggiorna la matrice delle vincite e il dizionario contenente i montepremi dei giocatori.

Classe Server

La classe Server eredita i metodi della classe Thread del modulo threading.

- Il metodo di inizializzazione prende come parametro un oggetto Game, che rappresenta la partita in corso. Inoltre, definisce due variabili: tabellone, cioè una lista di numeri da 1 a 90, e new_number, che conterrà, di volta in volta, l'ultimo numero estratto, e viene inizializzata a None.
- Il metodo draw si occupa di estrarre un numero dal tabellone e poi chiamare il metodo write della classe Game per scriverlo in una coda condivisa con i giocatori. Nello specifico, il banditore estrae un indice della lista tabellone, e assegna a new_number l'elemento della lista che ha quell'indice. Dopodiché, rimuove il numero dal tabellone. In questo modo, all'estrazione successiva non potrà essere estratto lo stesso numero.
- L'ultimo metodo è una sovrascrittura del metodo run, che descrive l'ordine delle operazioni eseguite dal thread. Nello specifico, una volta che il thread server viene avviato (tramite il metodo start della classe Thread), entra in un ciclo while che, a ogni iterazione, controlla che non sia ancora stata fatta tombola. Questo controllo viene effettuato tramite il metodo locked_win della classe Game. Perciò, il server rimane in esecuzione finché non viene fatta tombola, e solo a quel punto esce dal ciclo while e termina.
  All'interno del ciclo while si aprono due possibilità:

- se la lista degli estratti è vuota, viene sorteggiato il primo numero tramite il metodo draw;
- altrimenti, si controlla che tutti i giocatori abbiano letto l'ultimo numero estratto: se non è così si ricomincia il ciclo; in caso contrario, viene sorteggiato un nuovo numero e il server procede con il controllo delle vittorie tramite il metodo check_victories della classe Game.

Classe Player

La classe Player eredita i metodi della classe Thread del modulo threading.

- Il metodo di inizializzazione prende come parametro un oggetto Game, lo stesso che viene passato anche al Server, perché rappresenta la partita in corso, e un intero corrispondente al codice identificativo del giocatore.
- Il metodo create_card crea una lista ordinata di 15 numeri casuali da 1 a 90, ovvero i numeri della cartella. Dopodiché, crea un'altra lista con gli stessi numeri ma divisi in cinque righe da tre elementi ciascuna: tali righe andranno poi a costituire le cinque colonne della cartella nell'interfaccia grafica.
- Il metodo number_check controlla se il numero è presente nella cartella e, in tal caso, ne individua la riga tramite il suo indice e aggiorna il contatore di numeri estratti in quella riga (row_0, row_1 o row_2).
- Il metodo communicate_victories si occupa di aggiornare la matrice global_wins della classe Game nel caso in cui il thread giocatore abbia effettuato una vincita. Si distinguono sei possibilità, in base alla vincita da comunicare (ambo, terno, quaterna…).
  Per prima cosa, il metodo verifica che la vincita in questione abbia il flag a False, cioè che non sia ancora stata fatta: infatti, non è possibile fare ambo se qualcun altro lo ha già fatto a un'estrazione precedente. Dopodiché, il metodo controlla se il giocatore ha effettuato quella vincita, cioè se rispetta le condizioni di vittoria (ad esempio, se ha due numeri nella stessa riga, nel caso dell'ambo). In tal caso, aggiorna opportunamente la matrice global_wins.
- L'ultimo metodo è una sovrascrittura del metodo run, e ha una struttura simile al metodo run della classe Server. Nello specifico, una volta che il thread Player viene avviato, entra in un ciclo while che, a ogni iterazione, controlla che non sia ancora stata fatta tombola. Perciò, il giocatore rimane in esecuzione finché non viene fatta tombola, e solo a quel punto esce dal ciclo while e termina.
  All'interno del ciclo, il giocatore chiama il metodo read di Game per leggere l'ultimo numero estratto; se esso è None, esce dal ciclo e termina, perché significa che la partita è stata interrotta prima della fine; altrimenti, il thread comunica di aver letto l'ultimo numero aggiornando l'apposito contatore e poi chiama i metodi number_check e communicate_victories.

Classe Interface

La classe Interface implementa l'interfaccia grafica del gioco, utilizzando la libreria Tkinter.

- Il costruttore prende come parametro un oggetto Tk di Tkinter, che di fatto va a costituire una nuova finestra. Questa finestra viene poi configurata e arricchita con widget di varia

natura.
Inoltre, il metodo __init__ crea un oggetto Game, un oggetto Server e ulteriori widget (che non vengono subito inseriti nella finestra perché non necessari nella fase iniziale).
Nello specifico, i widget che vengono subito inseriti sono:

  - Un frame contenente le etichette che comunicano i premi in palio;
  - Una entry che tiene traccia del numero di giocatori prenotati;
  - enroll_button: un bottone che permette di prenotare un nuovo giocatore;
  - delete_button: un bottone che permette di eliminare un giocatore prenotato;
  - ok_button: un bottone che permette di confermare il numero di giocatori selezionato e di specificare i loro nomi;

- Il metodo enroll_handler è collegato a enroll_button: aggiunge un nuovo giocatore ogni volta che il bottone viene cliccato;
- Il metodo delete_handler è collegato a delete_button: elimina un giocatore ogni volta che il bottone viene cliccato;
- Il metodo ok_handler è collegato a ok_button: ogni volta che il bottone viene cliccato, verifica che il numero di giocatori sia maggiore di zero; in tal caso, modifica l'interfaccia per impedire di cambiare il numero di giocatori, crea delle entry in cui l'utente dovrà inserire i nomi dei giocatori e inserisce il bottone di inizio partita, start_button. Se invece il numero di giocatori è minore o uguale a zero, crea una finestra di errore.
- Il metodo start_handler è collegato a start_button: ogni volta che il bottone viene cliccato, verifica che sia stato indicato il nome di ogni giocatore; in tal caso, chiama il metodo avvio_giocatori, dà avvio al thread server e chiama il metodo interface_update. Se invece mancano uno o più nomi, crea una finestra di errore.
- Il metodo interface_update modifica l'interfaccia distruggendo i bottoni di inizio partita, creando una entry con i numeri estratti, chiamando il metodo update_numbers e inserendo un bottone per interrompere la partita.
- Il metodo update_numbers aggiorna continuamente l'interfaccia, mostrando sempre gli ultimi tre numeri estratti finché non viene effettuata una tombola; a quel punto, modifica ulteriormente l'interfaccia per comunicare che la partita è terminata e dare la possibilità di iniziarne un'altra.
- Il metodo stop_handler è collegato a stop_game_button: distrugge l'interfaccia;
- Il metodo new_game_handler è collegato a new_game_button: distrugge le finestre delle cartelle, ripristina l'interfaccia allo stato iniziale per permettere di iniziare una nuova partita e crea un nuovo oggetto Game e un nuovo thread Server;
- Il metodo interrupt_handler è collegato a interrupt_button: modifica l'interfaccia per chiedere conferma di voler interrompere la partita;
- Il metodo no_handler ripristina l'interfaccia all'aspetto precedente;
- Il metodo yes_hanlder fa terminare i thread e distrugge l'interfaccia;
- Il metodo avvio_giocatori:
  - crea i thread giocatori in quantità richiesta, passando come parametri la partita e l'indice identificativo;
  - li avvia;
  - crea una nuova finestra per ogni giocatore;

    - chiama il metodo create_card della classe Player per assegnare a ogni giocatore una cartella;
    - inserisce i numeri nelle cartelle tramite il metodo grid; in particolare, per ogni numero nella cartella, crea un frame contenente un'etichetta con il numero. Ogni etichetta viene poi aggiunta a una lista globale, contenente le label con i numeri di tutte le cartelle.
    - Chiama i metodi change_label_color e show_victories.
- Il metodo change_label_color cambia il colore delle etichette contenenti i numeri estratti in ogni cartella; viene richiamato ogni millisecondo;
- Il metodo show_victories controlla se è stata effettuata una vittoria e, in tal caso, lo comunica tramite una label nella cartella del giocatore vincitore; viene richiamato ogni millisecondo.