ForzaQuattroASCII

Francesco Gigliotti

July 2021

1 Indroduzione

Il gioco Forza 4 consiste in una sfida tra due giocatori secondo le seguenti regole:

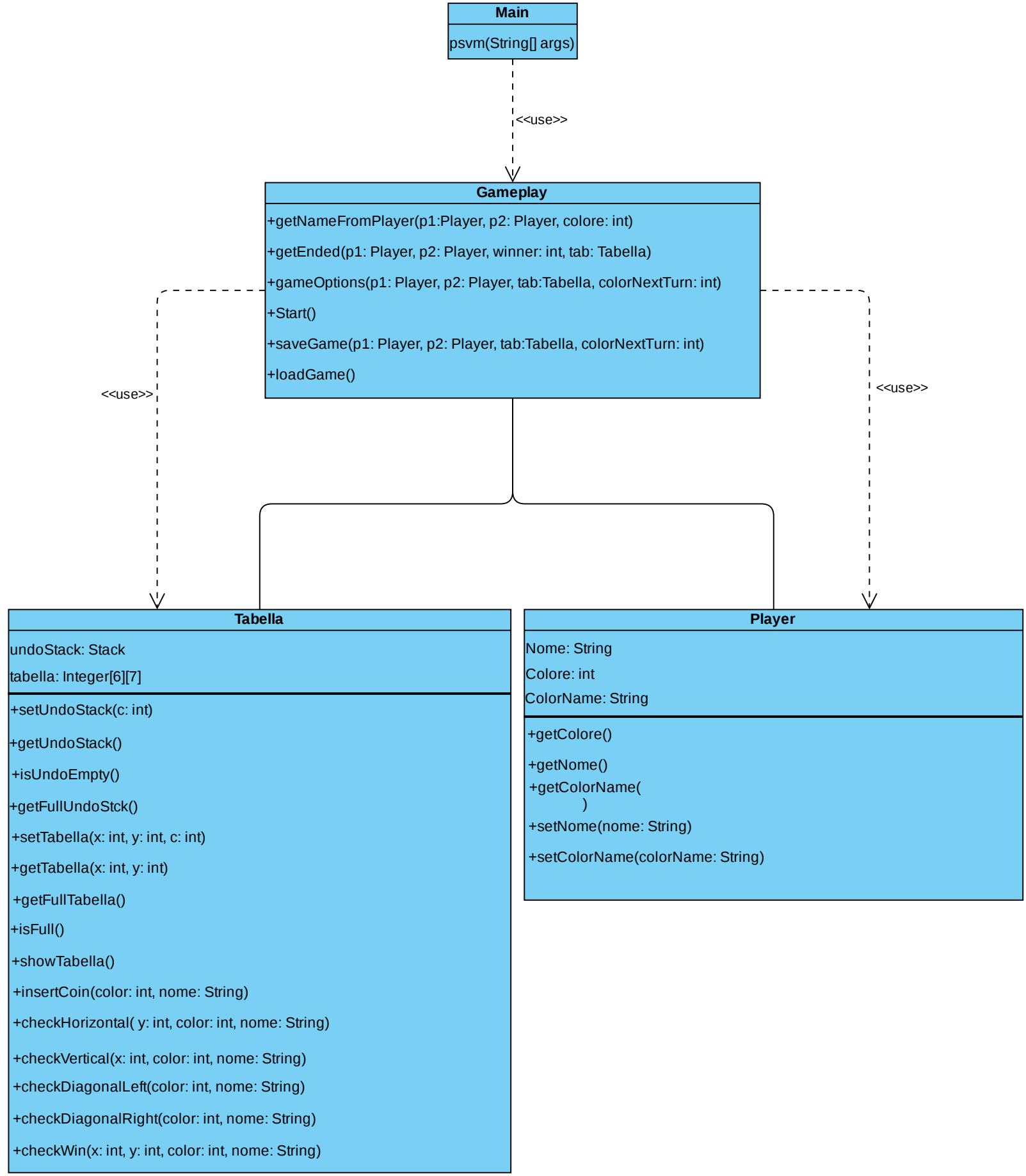
- Due giocatori operano su una griglia, disposta verticalmente e chiusa sul fondo, che ha dimensione di 6x7 (7 in verticale e 6 in orizzontale) caselle;
- I due giocatori dispongono di pedine diverse tra loro (1: Verde, 2: Giallo);
- I due giocatori a turno inseriscono una pedina in una delle colonne verticali della griglia: la pedina scende fino al fondo della colonna, oppure, fino ad appoggiarsi all'ultima pedina precedentemente inserita nella stessa colonna. In questo modo la pedina va ad occupare la casella più bassa della colonna, se la colonna era vuota, oppure la casella libera immediatamente superiore alle caselle già occupate in precedenza;
- È vietato inserire pedine in una colonna che sia completamente occupata;

Lo scopo dei giocatori è allineare in orizzontale, in verticale o in diagonale, quattro pedine dello stesso colore: la partita termina quando uno dei due giocatori raggiunge tale obiettivo o quando la griglia è completamente piena (caso di parità).

Il progetto è stato svolto da una singola persona.

Figura 1: Schermata principale

2 Descrizione delle classi



L'applicazione è composta da tre classi, ad esclusione del main.
Le classi sono:

- Gameplay: Si occupa dell' interazione con l'utente, quindi dell' avvio di una nuova partita o del proseguimento di una salvata, del caricamento e salvataggio dei dati di una partita, e dei menù di interazione con l'utente.
- Tabella: Contiene l'ogetto tabella (che sarebe la game board) e i metodi che servono per operare su di esso. Gestisce l'inserimento delle monete, i controlli di vittoria o pareggio, la visualizzazione della game board e salva le mosse effettuate dai giocatori, così da poter ripetere un' azione.
- Player: Assegna Nome e colore ai giocatori.

3 Descrizione delle funzionalità e GUI

Lanciando il main (o il file jar) ci si troverà di fronte una schermata del tipo:

Qui sarà possibile scegliere una tra le du opzioni, in caso venga selezionata un' opzione che non esiste,il programma lancerà un' eccezione e chiederà nuovamente di selezionare un' opzione.

1. Opzione (1): Lancierà il metodo Start() di Gameplay.
2. Opzione (2): Lancia il metodo loadGame() di Gameplay.

Metodo Start():
Il metodo Start() serve per far partire una nuova partita.
Il metodo chiederà ai due giocatori di inserire i loro nomi, dopodichè verificherà che non siano stati immessi nomi uguali (così da non creare confusione). Una volta accertata la bontà dei nominativi, il programma creerà la game board e deciderà casualmente il giocatore a cui spetta il primo turno.
Successivamente verrà lanciato il metodo gameOptions().

Metodo loadGame():
Il metodo loadGame() serve per caricare i dati salvati. Se non c' è il file "save.txt", dove sono salvate le informazioni sulla partita salvata, oppure il salvataggio risulta corrotto, il programma lancierà un' eccezione che avviserà l'utente del problema e, qiondi, chiederà di avviare una nuova partita (ovvero invocare il metodo Start()).

Se, invece, il caricamento dei dati avrà successo, come per Start() il metodo avvierà gameOptions().

Metodo gameOptions():

Questo metodo è colui che gestisce le interazioni con gli utenti.
Si presenta nel seguente modo:

Figura 2: Schermata di gioco

La schermata si compone di 4 componenti:

1. Titolo
2. Game board
3. Opzioni valide
4. Campo dove inserire l'opzione desiderata

Come prima se la scelta non è valida, il programma darà un eccezione e chiederà nuovamente un' opzione valida.

Opzioni:

1. Giocatore "nome giocatore" pronto!: selezionando questa opzione il giocatore a cui spetta il turno metterà la moneta nella colonna che preferisce. Viene invocato il metodo insertCoin() di Tabella, e nel caso tale metodo restituisca true allora il gioco terminerà chiamando il metodo gameEnded(), altrimenti verrà passato il turno all' altro giocatore.
2. Annulla ultima mossa: come è facile intuire, questo comando permette di tornare indietro alla mossa precedente (fino alla tabella vuota), ma solo se isUndoEmpty di Tabella non è true, infatti, nel caso fosse vuota la pila verrebbe comunicato all utente che l'opeazione non è effettuabile e

verrebbe ritornato alla Figura 2. Questa opzione chiama il metodo getUndoStack(), sempre di Tabella, che altro non fa chè prendere le coordinate della mossa precedente e, con il metodo setTabella(), re impostare a 0 il valore di quella coordinata rendendola, così, "vuota".

3. Salva ed esci: questa opzione permette di scrivere tutti i dati relativi alla partita su di un file txt chiamato "save.txt", una volta scritti tutti i dati l'applicazione termina e si chiude. Invoca il metodo saveGame() che si occupa di creare/riscrivere il file save.txt. L'organizzazione dei dati nel txt è descritta nel JavaDoc.

4. Esci senza salvare: chiude l'applicazione.

Classe Tabella:

Metodo insertCoin():
Il metodo insertCoin è il metodo che consente al giocatore di inserire la pedina nella colonna della game board desiderata.

Dopo che l' utente a cui spetta il turno dichiara di essere pronto, questo metodo stampa la tabella, in modo che l' utente possa decidere dove inserire la moneta, e chiede la colonna dove si dovrà inserire la pedina.
La schermata sarà quindi la seguente:

Figura 3: Schermata per l' inserimento della moneta

Ovviamente, anche in questo caso il metodo controllerà la validità dell' opzione data.

Dopo aver scelto la posizione, si entra in un loop while che si fermerà solamente se si è vinto, pareggiato o dopo che la pedina ha ragiunto l' ultima posizione disponibile. Ovviamente se la colonna è piena il metodo considererà lopzione non valida e chiedera di re inserire la moneta.

L'algoritmo consiste nel patire da metà tabella (int y = 3;), in modo tale da ridurre le operazioni da effettuare. Entrati nel loop, grazie al valore di findInsert pari a true, si verifica il contenuto della casella in cui mi trovo. Se il valore è 0 allora dovrò "scendere" (y+=1) e vedere se c'è qualcosa, se trovo un simbolo allora la posizione in cui ero prima di scendere è dove devo posizionare la pedina, altrimenti continuo a scendere finchè non trovo qualcosa o la fine (if y+1 == 7).Se il valore non è 0, allora, devo "salire" (y-= 1) finchè non trovo la posizione giusta con lo stesso concetto di cui sopra.

Matrice 3 righe 4 colonne

	0	1	2	3
0	23	15	3	25
1	3	12	66	78
2	16	21	12	3

Figura 4: Matrice per comprendere meglio il concetto di "salire" e "scendere"

Dopo aver trovato la posizione corretta, verranno salvate le coordinate della posizione dove verrà inserita la moneta nello stack "undoStack", dopodichè vienne effettivamente cambiato il simbolo della posizione trovata e viene settato findInsert a "false" in modo tale che se checkWin() dovesse dare esito negativo, comunque si uscirebbe dal loop. checkWin() è un metodo che non fa altro che chiamare altri metodi che tornano "true" se la partita è stata vinta da qualcuno o si è pareggiato.

Metodi checkHorizontal() e checkVertical():

Questi due metodi sono pressochè identici, salvo le differenze ovvie per la diversa posizione di rilevamento del tris. Pertanto descriverò di seguito solo checkHorizontal().

Il tipo di tris che verrà verificato è mostrato nel JavaDoc.

Per prima cosa viene creato uno Stack winCheck, che servirà per salvare le posizioni delle monete che formano il tris. Vengono, poi, inizializzate 3

variabili: res, count e X; res vine inizializzata a false, mentre count e X a 0. L'algoritmo è diviso in due parti: la prima che verifica il tris e la seconda, che qualora vi sia il tris ne cambia il valore delle monete che lo formano e cambia res in true.

Nella prima parte si eseguirà un loop while finchè il valore di x è meno di 6 (fino al limite dx) e cout è minore di 4 (quindi se si farà tris cout diventerà pari a 4 e il loop terminerà). Ovviamente, dato che si parte dal lato sx della matrice, se si arriva alla posizione 3 con un valore di cout pari a 0 si può uscire prima dal loop, in quanto non c'è lo spazio fisico per un tris.

Nel caso in cui si trova una moneta del colore cercato nella posizione in cui ci si trova, verranno salvate le coordinate della pedina nello stack, e cout e X verranno incrementate di 1. Se, però, prima di raggiungere con cout il valore di 4 viene trovata una moneta del colore sbagliato, cout verrà azzerato la pila sarà svuotata ma la X verrà comunque incrementata. Infatti se si dovesse arrivare alla posizione 3 con cout pari a 0, come detto sopra, il loop termina.

La seconda parte è più semplice, semplicemente se si è trovato un tris allora si prenderanno le coordinate delle monete che lo formano e verrà sostituito il contenuto delle celle con un 8, che poi dal metodo showTabella() verranno letti e stampati come X rosse, per evidenziare il tris.

Se è presente il tris, verrà visualizzata la tabella con il tris evidenziato e il nome del vincitore.

Ecco come si presenta la schermata di vittoria:

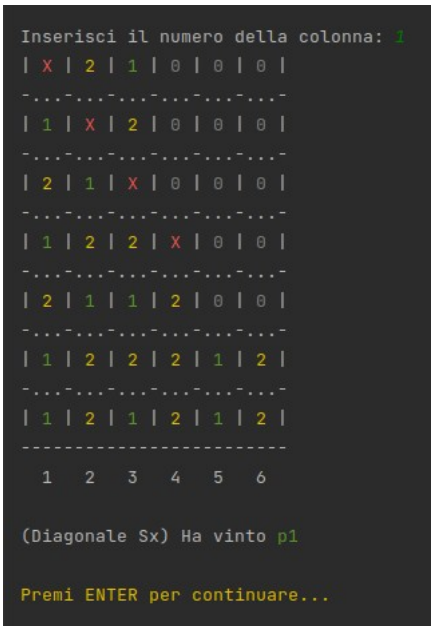


Figura 5: Schermata di vittoria

Metodi checkDiagonalLeft() e checkDiagonalRight():
Come sopra descriverò il funzionamento di uno solo dei due metodi, dto che sono molto simili.
N.B. Le diagonali sinistre (vedere JavaDoc per capire cosa si intende per diagonale sinistra) che partono da y == 4 sono inutili da controllare, in quanto non c'è spazio fisico a disposizione per poter fare tris. Stesso discorso per le diagonali sinistre che partono da x == 3.
Anche qui si usano Stack e cout, l'unica differenza sono le inizializzazioni di X che invece questa volta sono 3 valori: Y = 3, x = 0, y = 0. Questa volta il meccanismo si divide in 3 parti, anzichè 2 come sopra. L'ultima parte, però, è uguale all' ultima parte della precedente, pertanto non sarà descritta. La prima parte verificherà le posizioni che partono da y == 3 ma restano con x fissa a 0.
Perciò le seguenti diagonali:

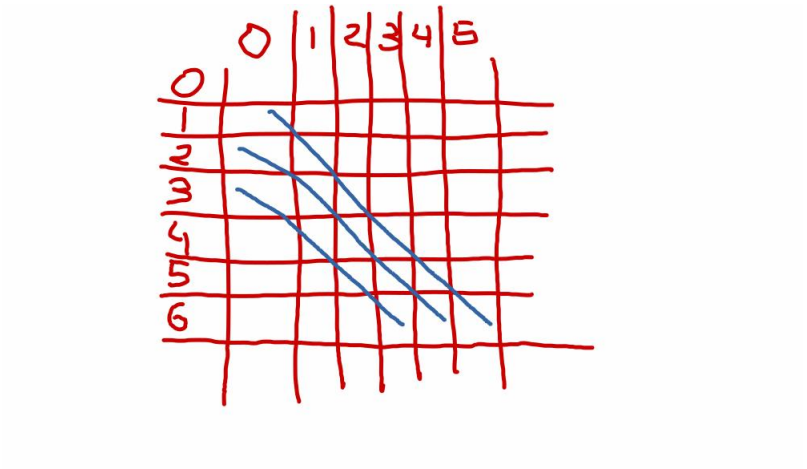


Figura 6: check del primo pezzo di diagonale sinistra

Sostanzialmente in questa parte si entra in un ciclo for con variabile y che viene inizializzata a Y (quindi 3)e si ferma una volta che y arriva a 7. All' interno del for si avrà lo stesso principio della prima parte descritta sopra per checkHorizontal(), quindi si salvano le coordinate nello stack, si contano i colori vicini e se si trova un tris si esce dal for, altrimenti si svuota lo stack e viene incrementata la x di 1.
Usciti dal for si vede se si è fatto tris, e in caso si esce, altrimenti count e x vengono azzerate, mentre Y viene decrementata per poter "salire" (si ricorda la Figura 4).

La seconda parte consiste nel verificare se si è fatto tris, e qual' ora non fosse così di controllare anche la parte mancante della tabella, cioè:

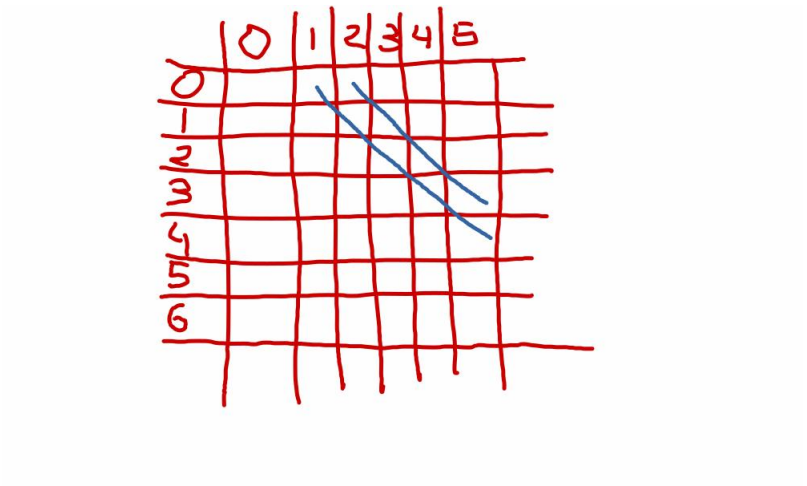


Figura 7: check del primo pezzo di diagonale sinistra

Il principo è lo stesso, ma questa volta il for gestira la x, mentre a fine for verrà incrementata Y, invece dopo il controllo del tris il ruolo della Y e della x sono invertiti (verra azzerata Y e incrementata x).

Metodo isFull():

In fine vediamo il metodo isFull(), che ci dirà se è avvenuto un pareggio. Questo metodo non deve fare altro che vedere se la cima della tabella è piena, se è così allora la tabella è piena altrimenti se c'è anche una sola casella libera darà false. Per fare ciò basta un ciclo for che va da 0 a 5 e invocando getTabella(x, 0) possiamo ottenere il contenuto della casella cercata, appena ne troviamo una vuota il metodo da false. essendo l'ultimo metodo ad essere eseguito dal metodo checkWin(), verrà chiamato solo se non ci sono vincitori e, pertanto, se dovesse dare true vorrebbe dire certamente che c'è stato un pareggio.