Progetto Forza 4 con visualizzazione su console

Metodologie Di Programmazione a.a. 2021/2022

Stefano Bollella

matricola: 2025438

Introduzione

L'oggetto dell'elaborato è la simulazione del gioco Forza 4, pensato per l'esecuzione su console.
A seguito di una fase preliminare di analisi del gioco, sono stati individuati i principali concetti costituenti la sua struttura, ciò ha permesso di elaborare delle classi in grado di descrivere tali concetti all'interno del dominio del problema.
Si è cercato di sviluppare l'intero progetto secondo i principi della programmazione orientata agli oggetti e i principi SOLID. In particolare, per tutte le classi implementate, si è tentato di applicare il principio di single responsibility, secondo cui ogni singola classe dovrebbe occuparsi di un singolo problema.
Nell'elaborato, si è preferito adottare un approccio incentrato sull'utilizzo delle interfacce per favorire la modularità del software, quindi, una migliore ridistribuzione delle responsabilità tra le varie classi, e per ridurre l'accoppiamento tra classi.

Descrizione delle classi

La classe **Player** rappresenta il concetto di giocatore all'interno del sistema.

Gli attributi della classe sono:

`String name;` rappresenta il nome del giocatore.

`String color;` rappresenta il colore della pedina.

L'interfaccia pubblica della classe è costituita da:

`public String getName();` restituisce uno dei due colori delle pedine di gioco.
`public String getColor();` restituisce il nome del giocatore.

Dato che la classe Player realizza l'interfaccia **Tokenable**, essa implementa i metodi dichiarati all'interno dell'interfaccia stessa.

La classe **Grid** rappresenta il concetto di griglia di gioco all'interno del sistema.

L'attributo della classe è:

```
Tokenable[][] tokenGrid;
```

`tokenGrid` è una array bidimensionale di tipo interfaccia **Tokenable**, che rappresenta lo stato della griglia di gioco e
memorizza al proprio interno la posizione delle pedine.
Le pedine sono rappresentate da riferimenti ad oggetti di tipi di classi, che implementano l'interfaccia Tokenable, in questo caso si è scelto di utilizzare riferimenti ad oggetti di tipo Player, invece di creare una classe apposita per rappresentare la pedina di gioco.
In relazione alla scelta fatta un riferimento nullo equivale ad una cella vuota.

L'interfaccia pubblica della classe è costituita da:

```
public Grid( Tokenable[][] tokenGrid);
```

Il costruttore della classe Grid assegna alla variabile d'istanza `tokenGrid` un array bidimensionale del tipo interfaccia Tokenable come unico parametro di costruzione.

Dato che la classe Grid realizza l'interfaccia **GridManager**, essa va ad implementare tutti quei metodi elencati dall'interfaccia.

```
public void addToken(Tokenable token, int column);
```

`addToken` permette di aggiungere una nuova pedina in una delle colonne della griglia di gioco `tokenGrid`**.**

Il parametro `column` di tipo int rappresenta la colonna scelta dal giocatore corrente, mentre il parametro `token` del tipo interfaccia Tokenable, che referenzia un oggetto di tipo Player, rappresenta la pedina da aggiungere alla griglia di gioco. Quindi, ad uno degli elementi dell'array bidimensionale `tokenGrid` viene assegnato un riferimento ad un oggetto di tipo Player.

Il nome del metodo `checkFull` è overloaded.

```
public boolean checkFull();
```

`checkFull` verifica che la griglia di gioco non sia piena, cioè se all'interno dell'array bidimensionale `tokenGrid` non sia presente alcun riferimento nullo. Il metodo restituisce true se la griglia di gioco è piena, altrimenti false.

```
public boolean checkFull(int column);
```

`checkFull(int column)` controlla che la colonna `column`, passata al metodo come parametro, non sia piena. Il metodo restituisce true se tutti i riferimenti della colonna non sono nulli, altrimenti false.

```
public boolean isDiagonal(Tokenable token);

public boolean isVertical(Tokenable token);

public boolean isHorizontal(Tokenable token);
```

isDiagonal, isVertical, isHorizontal sono i metodi per la ricerca di pedine dello stesso colore allineate in tre possibili modi : diagonale, verticale e orizzontale.

Questi metodi sono stati implementati in maniera tale da verificare se, all'interno dell'array bidimensionale `tokenGrid`, siano presenti quattro elementi allineati in un certo modo e che referenziano lo stesso oggetto referenziato da `token`.
`token` è l'unica variabile parametro passata come argomento.

Quindi la verifica di un allineamento di quattro pedine dello stesso colore, all'interno del sistema di funzionamento del gioco, comporta che quattro elementi dell'array Bidimensionale debbano referenziare lo stesso oggetto di tipo Player, cioè uno dei due giocatori della partita.

L'implementazione privata di questi metodi utilizza dei riferimenti del tipo interfaccia Tokenable per elaborare la griglia di gioco. In questo caso, i metodi invocati con i riferimenti di tipo Tokenable sono i metodi implementati all'interno della classe Player, che realizza l'interfaccia Tokenable come chiarito in precedenza. In questo caso, si ha un polimorfismo seppur debole.

La classe **GameState** rappresenta lo stato della partita. Un oggetto di tipo GameState all'interno del sistema di gioco permette di mantenere aggiornate tutte quelle informazioni necessarie al prosieguo della partita, poiché con l'alternarsi dei turni di gioco queste informazioni, come lo stato della griglia di gioco, variano.

Gli attributi della classe sono:

`Tokenable currentPlayer;` il giocatore che deve effettuare la propria mossa nel turno corrente.

`Tokenable secondPlayer;` il secondo giocatore che ha effettuato la propria mossa nel turno precedente.

`GridManager gameGrid;` la griglia di gioco.

L’interfaccia pubblica della classe è costituita da:

```
public GameState(Tokenable player1, Tokenable player2, GridManager
gameGrid);
```

Il costruttore della classe permette di inizializzare un nuovo oggetto di tipo GameState, le variabili parametro passate al metodo `player1` e `player2` sono due variabili di riferimento di tipo Tokenable, che referenziano due oggetti di tipo Player e rappresentano i due giocatori della partita. La variabile parametro `gameGrid` del tipo interfaccia GridManager è una variabile di riferimento che referenzia un oggetto di tipo GameGrid e rappresenta la griglia di gioco.

La classe GameState realizza l’interfaccia **GameStateInformation**, quindi implementa i metodi dichiarati all’interno dell’interfaccia.

```
public void move(int column);
```

Un giocatore, per effettuare la propria mossa, deve scegliere la colonna dove far cadere una delle sue pedine. Questo comportamento si traduce nell’individuare il primo elemento nullo della colonna scelta dal giocatore e nell’assegnare un riferimento allo stesso oggetto referenziato dalla variabile d’istanza currentPlayer. L’oggetto referenziato da CurrentPlayer è un oggetto di tipo Player e rappresenta il giocatore del turno corrente che ha appena effettuato la sua mossa.
Il numero della colonna scelta `column` viene passato come argomento al metodo. Se la mossa del giocatore corrente va a buon fine, allora il metodo move aggiorna anche le variabili currentPlayer e secondPlayer.

```
public boolean checkWin();
```

`checkWin` Controlla se il giocatore corrente ha vinto la partita, cioè se sono state allineate quattro pedine dello stesso colore in una delle possibili direzioni: orizzontale, verticale oppure diagonale. Restituisce true se il giocatore ha vinto, altrimenti false.

```
public Tokenable getCurrentPlayer();
```

`getCurrentPlayer` restituisce un riferimento di tipo Tokenable, che rappresenta il giocatore corrente.

```
public Tokenable getSecondPlayer();
```

`getSecondPlayer` restituisce un riferimento di tipo Tokenable, che rappresenta il secondo giocatore.

```
public GridManager getGameGrid();
```

`getGameGrid` restituisce un riferimento di tipo GridManager, che rappresenta la griglia di gioco della partita.

L'implementazione privata di questi metodi utilizza riferimenti del tipo Tokenable e GridManager per aggiornare la griglia di gioco tramite il metodo `move` oppure per controllare se il giocatore corrente ha vinto tramite il metodo `checkWin`. I metodi invocati con i riferimenti di tipo Tokenable e GridManager sono i metodi implementati, rispettivamente, all'interno della classe Player, che realizza l'interfaccia Tokenable, e la classe Grid, che realizza la classe GridManager. Anche in questo caso, si ha un polimorfismo seppur debole.

La classe **GameData** implementa tutti quei comportamenti necessari alla scrittura e lettura dei dati di una partita su file.

Gli attributi della classe sono:

`String fileName;` rappresenta il nome del file dove leggere e scrivere i dati riguardanti le partite.

l'interfaccia pubblica della classe :

```
public GameData(String fileName);
```

Il costruttore inizializza un nuovo oggetto di tipo GameData, la variabile parametro `fileName` di tipo String è il nome del file passato come argomento al metodo.

```
public void saveGame(String nameGame, GameStateInformation
gameStateToSave);
```

`saveGame` scrive lo stato della partita su file tramite la variabile parametro `gameStateToSave` di tipo GameStateInformation passata come argomento. Il metodo estrapola da `gameStateToSave` tutte le informazioni necessarie al salvataggio della partita. La variabile parametro `nameGame` è il nome della partita, scritto su file insieme alle altre informazioni. Anche in questo caso, si ha un polimorfismo seppur debole.

```
public ArrayList<String> readGame(String nameGame);
```

`readGame` legge da file lo stato di una partita precedentemente salvata, questa viene recuperata all'interno del file tramite il suo nome `nameGame` passato come argomento al metodo.

```
public boolean searchGame(String nameGame);
```

`searchGame` cerca all'interno del file il nome della partita passato come argomento al metodo e restituisce true, se la partita è stata trovata, altrimenti false.

La classe **GameManager** è la classe starter del progetto, essa contiene il metodo main. Inoltre, al suo interno, implementa un metodo per la gestione della partita e un metodo per la visualizzazione a schermo della griglia di gioco.

L'interfaccia pubblica della classe è costituita da:

```
public static void main(String[] args);
```

Il metodo main implementa un menu di gioco, all'interno del quale uno dei due giocatori può scegliere se avviare una nuova partita oppure caricarne una tra quelle salvate in precedenza.

```
public static void runGame(GameStateInformation newGameState,GameData
gameDataFile, Scanner input);
```

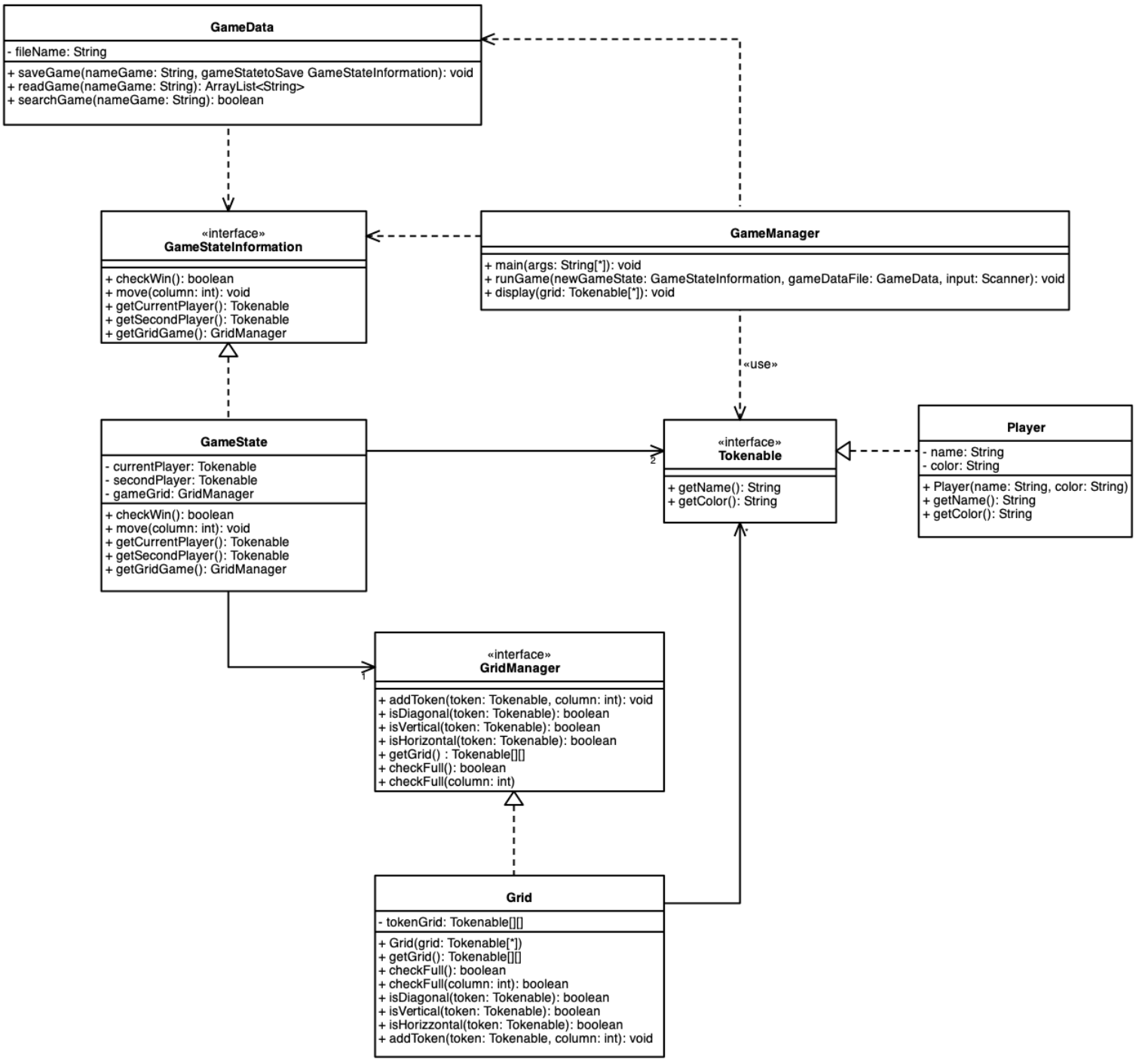
`runGame` avvia una nuova partita, quindi alterna i turni di gioco tra i due giocatori. Tramite una variabile di riferimento `newGameState` di tipo GameStateInformation, che referenzia un oggetto di tipo GameState, permette di aggiornare lo stato della partita turno dopo turno. La variabile di riferimento `gameDataFile` di tipo GameData consente di scrivere lo stato della partita su file.

```
public static void display(Tokenable[][] grid);
```

`Il metodo display` tramite la variabile parametro grid, un array bidimensionale di tipo Tokenable, elabora lo stato della griglia di gioco, in modo tale da formattarlo per la visualizzazione a schermo.

La classe **FullColumnException** che estende la classe RuntimeException
è una classe di eccezione creata per descrivere una condizione di errore.
Viene lanciata nel momento in cui un giocatore tenta di inserire una pedina all'interno di una colonna piena .

Diagramma UML delle classi



Descrizione delle funzionalità

Gestione della partita

Per simulare una nuova partita di gioco si è pensato all'implementazione di un metodo chiamato runGame della classe GameManager, questo metodo permette di alternare turni di gioco nei quali il giocatore corrente può effettuare la propria mossa o gestire l'eventuale interruzione dell'esecuzione della partita. All'interno del corpo del metodo si ha un primo ciclo while, che in maniera iterativa permette di alternare i turni di gioco.

```
while (exitGame && !newGameState.getGrid().checkFull){ ...}
```

L'uscita dal ciclo stabilisce la fine della partita.

Le condizioni di fine partita si verificano quando la variabile `exitGame` assume un valore booleano false, ciò accade quando durante la partita uno dei due giocatori decide di uscire oppure vince la partita. L'altra condizione necessaria si ha quando la griglia di gioco risulta essere ormai piena, quindi si è raggiunta la parità. Questo controllo viene fatto invocando il metodo `checkFull` della classe Grid, il quale restituisce true se la griglia di gioco è piena.

Per gestire il singolo turno di gioco si è scelto di implementare un altro ciclo while annidato.
Inoltre, lo stesso consente di far fronte all'esecuzione di mosse vietate oppure all'inserimento di valori in input non idonei da parte dei due giocatori
Al verificarsi di una di queste due situazioni, al giocatore corrente viene permesso di ripetere la mossa.

```
while(turnCompleted) {. . .}
```

All'interno di questo ciclo while viene implementato un blocco try-catch, utilizzato per catturare due determinate eccezioni.

```
try {
     . . .
}
catch (InputMismatchException e) {
. . .
}
catch (FullColumnException e) {
 . . .
}
```

La prima eccezione è `InputMismatchException` viene lanciata all'interno del blocco try quando il giocatore corrente tenta di inserire un valore diverso da un intero. La seconda eccezione `FullColumnException,` corrispondente ad una mossa vietata dal gioco, viene lanciata all'interno del blocco try quando il giocatore sceglie di inserire la propria pedina in una colonna piena. Entrambe le eccezioni vengono gestite in maniera tale da permettere al giocatore di ripetere la mossa.

All’interno del blocco try viene data la possibilità al giocatore di effettuare la propria mossa oppure di mettere in pausa il gioco.

```
int choiceColumn = in.nextInt();
```

In base al valore inserito in input da parte del giocatore si possono avere diverse opzioni.

Se il valore inserito è uguale alla costante intera PAUSE_MENU, viene data la possibilità al giocatore di aprire un menu, che viene implementato all’interno di un ciclo while. Questo espediente consente di simulare la messa in pausa del gioco.
L’uscita dal menu si ha nel momento in cui il giocatore sceglie di ritornare alla partita corrente oppure di uscire dalla partita stessa. Inoltre, all’interno del menu, il giocatore può salvare la partita corrente dandole un nome.

```
 if (choiceColumn == PAUSE_MENU){. . .}
```

Se il giocatore sceglie un valore corrispondente ad una delle colonne della griglia, viene invocato il metodo `move` della classe GameState per permettere al giocatore corrente di effettuare la propria mossa.
Se la colonna scelta è una colonna piena, viene lanciata l’eccezione `FullColumnException.` Invece, se la mossa va a buon fine, il metodo `move` aggiorna lo stato di gioco, il giocatore corrente, che ha effettuato la mossa, diventa il secondo giocatore e il secondo giocatore diventa il giocatore corrente del turno successivo.
Successivamente viene invocato il metodo `checkWin` della classe GameState per determinare se la mossa del giocatore è quella vincente, in caso di vittoria la partita termina.

```
else if(choiceColumn >= 0 &&  choiceColumn < COLUMNS){ . . .}
```

Se l’inserimento non è alcuna di queste casistiche, allora il turno si ripete, lasciando inalterato lo stato di gioco, e viene richiesto all’utente di effettuare nuovamente la propria mossa.

Inserimento delle pedine all'interno della griglia di gioco

Per posizionare una pedina in una colonna scelta dal giocatore viene implementato il metodo `addToken` della classe Grid. Prima che la pedina venga inserita, viene effettuato un controllo sulla colonna, invocando il metodo `checkFull(int column),` che ritorna un valore booleano true se la colonna selezionata è una colonna piena.

```
if(!this.checkFull(column)) {
   . . .

}
else {throw new FullColumnException("the selected column is full");}
```

In caso di colonna piena viene lanciata l'eccezione FullColumnException, catturata nel metodo runGame della classe GameManager.
In caso contrario, all'interno del corpo dell'if, viene implementato un ciclo for che, in maniera iterativa partendo dal basso della colonna, controlla l'elemento i-esimo di questa. Se l'elemento è un riferimento nullo, quindi una cella vuota, allora è possibile inserire la pedina del giocatore corrente. Le pedine vengono aggiunte alla prima occorrenza di una cella vuota, partendo dal basso nella colonna selezionata.

```
for(int i = grid.length - 1; i >= 0; i -- ) {

            if(this.grid[i][column] == null){

               this.grid[i][column] = token;
              return;
       }
 }
```

Ricerca delle quattro pedine dello stesso colore allineate

Una delle caratteristiche principali del programma è la ricerca di quattro pedine allineate all'interno della griglia di gioco.
Si è scelto di non implementare un unico metodo per la ricerca, ma di suddividere questa operazione in tre differenti metodi `isDiagonal, isVertical, isHorizontal`.Tutti seguono lo stesso schema logico di ricerca, ma vengono implementati in maniera differente.

Nel caso di pedine allineate in diagonale, si hanno una ricerca delle pedine su una diagonale ed una ricerca delle pedine in direzione ortogonale alla prima.
L'individuazione dell'allineamento delle pedine viene effettuato scorrendo riga per riga tutti gli elementi della matrice partendo dal basso.

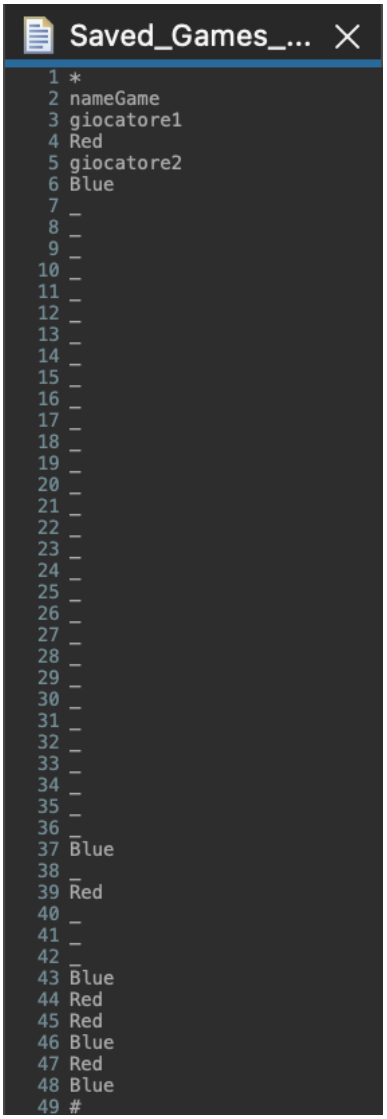
```
 for(int i = grid.length -1; i > grid.length - FOUR; i--) {
            for(int j = grid[0].length -1; j >=  grid[0].length - FOUR; j--
){
    . . .
  }
}

for(int i= grid.length -1; i > grid.length - FOUR; i--) {
          for(int j= grid[0].length - FOUR; j >= 0; j--) {
     . . .
  }
}
```

Per ogni elemento selezionato si conduce un controllo in direzione della sua diagonale, per verificare la presenza di altre tre pedine omologhe. Questo controllo avviene anche per la diagonale opposta.
Gli elementi della matrice compresi nella ricerca sono quelli appartenenti ad una sottomatrice ben definita; infatti, per alcuni elementi della matrice non è possibile allineare lungo le loro diagonali quattro pedine. Questi costituiscono una zona d'ombra che viene eliminata dalla ricerca.
Inoltre, si è tenuto conto anche di un eventuale allineamento di pedine in numero superiore alle quattro unità.

I due restanti metodi `isVertical, isHorizontal` sono stati implementati in maniera analoga, infatti questi seguono la stessa logica per la ricerca delle pedine allineate.

Salvataggio della partita su file



Un'altra funzionalità del programma è il salvataggio della partita su file. Il metodo implementato per questo scopo è saveGame della classe GameData.
Come specificato nella descrizione delle classi, questo metodo riceve come parametri il nome della partita scelta dall'utente e un riferimento di tipo GameStateInformation, che rappresenta lo stato della partita.
Per poter scrivere queste informazioni su file si è scelto di istanziare un oggetto della classe FileWriter, in questo caso viene utilizzato un costruttore specifico della classe FileWriter. Vengono passati come argomenti il nome del file e un valore booleano true, quest'ultimo serve ad aggiungere i dati, alla fine del file, a quelli preesistenti.

```
FileWriter out = new FileWriter(fileName, true);
```

Per scrivere su file viene invocato il metodo `write(String str)` della classe java.io.Writer .

Per il salvataggio su file dei dati della partita, quali nome della partita, nome dei due giocatori, i rispettivi colori e lo stato della griglia di gioco, si è scelto di formattare i dati su un'unica colonna per facilitare il loro recupero in un secondo momento.
Ogni partita all'interno del file è separata da un carattere sentinella '#', mentre il carattere underscore '_' viene utilizzato per indicare una casella vuota della griglia di gioco.
Inoltre, all'interno del file viene utilizzato un ulteriore carattere sentinella ' * ' per determinare la posizione del nome della partita.

Ripristino di una partita salvata su file

Per quanto riguarda il recupero dei dati di una partita da file si è scelto di implementare il metodo readGame della classe GameData, i dati recuperati vengono salvati all'interno di un vettore di tipo ArrayList<String> escluso il caratteri sentinella ' * ' e '#'. Questo metodo viene invocato quando il giocatore seleziona all'interno del menu di gioco l'opzione di caricamento di una partita da file.
Per leggere i dati da file si impiega un oggetto di tipo FileReader utilizzato come parametro per costruire un oggetto di tipo Scanner.