Forza 4

Relazione progetto Metodologie di Programmazione AA.2020-2021

Lucia Fores

Luglio 2021

Sommario

Forza 4 è un famoso gioco da tavolo per due giocatori in cui gli sfidanti cercano di allineare quattro pedine del proprio colore in verticale, orizzontale o diagonale. Nel seguente elaborato viene mostrata e discussa un'implementazione del gioco in Java.

Indice

1 Introduzione

Forza 4 è un gioco di allineamento per due giocatori in cui si cerca di mettere in linea, verticalmente, orizzontalmente o diagonalmente, quattro pedine.
La più grande differenza con altri giochi di allineamento sta nel fatto che in Forza 4, durante la partita si tiene conto della forza di gravità.
Nell'implementazione presentata è possibile effettuare una partita di Forza 4 su console.

1.1 Regole del gioco

Due giocatori si sfidano su una scacchiera 6×7 a chi riesce per primo ad allineare quattro pedine del proprio colore in verticale, orizzontale o diagonale.
All'inizio della partita viene scelto l'ordine dei giocatori tramite una conta e per tutta la partita i giocatori si alternano mettendo durante il proprio turno una pedina nella scacchiera; proprio perché in Forza 4 si deve tenere conto della forza di gravità, i giocatori possono solamente scegliere la colonna in cui inserire la propria pedina: questa infatti scenderà fino alla fine della colonna, nel caso in cui questa sia completamente vuota, oppure fino all'ultima pedina inserita.
La partita può avere due esiti:

- Vittoria per uno dei due giocatori
  Uno dei due giocatori è riuscito ad allineare quattro pedine del proprio colore
- Pareggio
  La scacchiera risulta completamente piena ma nessuno dei due giocatori è riuscito ad allineare quattro pedine

1.2 Contenuto dell'elaborato

Nel presente elaborato viene descritta un'implementazione del gioco in Java: il progetto è stato svolto dalla studentessa Lucia Fores del corso di Metodologie di Programmazione AA.2020-2021.
Nel Capitolo 2 viene fornita una descrizione di tutte le classi implementate per il funzionamento del gioco, vengono forniti i diagrammi UML delle classi, un diagramma UML della gerarchia delle classi e viene fornita una breve descrizione di ciò che le classi di test si occupano di dimostrare.
Nel Capitolo 3 vengono descritte le funzionalità del programma: in particolare viene descritto come le regole del gioco sono state implementate e vengono descritti tutte i ragionamenti alla base delle decisioni prese; viene infine mostrato un esempio di flusso di gioco di una partita standard.
Nel Capitolo 4 vengono riportati i referenti di sviluppo del progetto.

2 Descrizione delle classi

Nell'implementazione presentata sono state modellate come classi tutte le componenti del gioco:

- La griglia di gioco
- I giocatori
- Le pedine

È stata quindi creata una classe per giocare una partita, sono state introdotte delle eccezioni personalizzate per gestire alcune situazioni critiche del gioco.
Vengono fornite classi di test, una per ogni componente del gioco, diverse demo di partite per testare le situazioni critiche e una demo di gioco rappresentante una partita generica tra due giocatori.
In questa sezione dell'elaborato vengono descritte le classi che modellano le componenti di gioco: per ogni classe vengono forniti un diagramma UML e una breve descrizione delle variabili e dei metodi implementati; vengono inoltre descritte le eccezioni, di cui viene presentata la struttura di implementazione e la causa che ne scatena il sollevamento.
Vengono poi descritte tutte le classi di test e demo riportando per ognuna tutte le casistiche testate.
Infine viene presentato un diagramma UML che rappresenta la gerarchia delle classi così da mostrare la struttura architetturale del progetto.

2.1 Griglia di gioco

La griglia di gioco è stata modellata nella classe **PlayingGrid**.

2.1.1 Diagramma UML della classe

Viene qui di seguito presentato il diagramma UML della classe

Figura 1: Diagramma UML della classe PlayingGrid

2.1.2 Variabili di istanza

- ```
  private final int ROWS = 6
  ```

  La scacchiera di gioco può avere solo 6 righe, per questo motivo la variabile ROWS è stata definita come final;

- ```
  private final int COLUMNS = 7
  ```

  Analogamente a quanto fatto per le righe della scacchiera di gioco, anche le colonne possono essere solo 7 di conseguenza anche questa variabile è stata definita final;

- ```
  private ColoredDisc[][] playingGrid
  ```

  La scacchiera di gioco è stata modellata come una matrice di pedine colorate (ColoredDisc);

- ```
  private int freeSpaces
  ```

  La variabile è un contatore che tiene il conto di quanti spazi liberi (quindi di quante caselle senza pedine colorate) sono ancora presenti nella scacchiera di gioco.

2.1.3 Metodi

- ```
  public PlayingGrid()
  ```

  Uno dei due costruttori della classe: questo costruttore viene invocato nel caso in cui si stia costruendo la griglia di una nuova partita;

- ```
  public PlayingGrid(String saveFile)
  ```

  Il secondo costruttore della classe: questo costruttore viene invocato quando si deve ricostruire la griglia di gioco di una partita precedentemente salvata;

- ```
  public int getFreeSpaces()
  ```

  Il metodo serve per ottenere il numero di spazi liberi disponibili in un dato momento dell'esecuzione del programma;

- ```
  public void updateGrid()
  ```

  Il metodo è utilizzato per aggiornare il numero di spazi liberi di una griglia di gioco ogni volta che una mossa valida viene effettuata da un giocatore;

- ```
  public boolean addColoredDisc(ColoredDisc disc)
  ```

  Il metodo viene invocato ogniqualvolta si vuole inserire una nuova pedina nella griglia di gioco: il metodo inserisce le pedine tenendo conto della forza di gravità e ritorna l'esito dell'inserimento;

- ```
  public boolean areFourVerticalAligned(
  ColoredDisc disc)
  ```

  Uno dei controlli per testare una possibile configurazione di vittoria: il metodo controlla se ci sono quattro pedine dello stesso colore (compresa l'ultima pedina inserita del giocatore, che è quella passata come parametro) allineate verticalmente;

- ```
  public boolean areFourHorizontallyRightAligned(
  ColoredDisc disc)
  ```

  Uno dei controlli per testare una possibile configurazione di vittoria: il metodo controlla se ci sono tre pedine dello stesso colore dell'ultima inserita allineate alla destra di quest'ultima;

- ```
  public boolean areFourHorizontallyLeftAligned(
  ColoredDisc disc)
  ```

  Uno dei controlli per testare una possibile configurazione di vittoria: il metodo controlla se ci sono tre pedine dello stesso colore dell'ultima inserita allineate alla sinistra di quest'ultima;

- ```
  public boolean[] areFourDiagonallyLeftAligned(
  ColoredDisc disc)
  ```

  Uno dei controlli per testare una possibile configurazione di vittoria: il metodo controlla se ci sono quattro pedine dello stesso colore allineate sulla diagonale di sinistra della griglia (con "diagonale di sinistra" si intende la diagonale che parte da in alto a sinistra e arriva in basso a destra).
  Il metodo ritorna un array di booleani poiché testa separatamente se le tre pedine precedentemente inserite nella griglia sono allineate nella diagonale al di sopra dell'ultima pedina inserita o al di sotto di questa;

- ```
  public boolean[] areFourDiagonallyRightAligned(
  ColoredDisc disc)
  ```

  Uno dei controlli per testare una possibile configurazione di vittoria: il metodo controlla se ci sono quattro pedine dello stesso colore allineate sulla diagonale di destra della griglia (con "diagonale di destra" si intende la diagonale che parte da in alto a destra e arriva in basso a sinistra).
  Analogamente a quanto detto sopra, il metodo ritorna un array di booleani poiché testa separatamente se le tre pedine precedentemente inserite nella griglia sono allineate nella diagonale al di sopra dell'ultima pedina inserita o al di sotto di questa;

- ```
  public boolean isTheWinningMove(int rowIndex,
  int columnIndex)
  ```

  Il metodo si occupa di chiamare in sequenza tutte i metodi che testano le varie condizioni di vittoria testandole sull'ultima pedina inserita, ossia quella in posizione (rowIndex, columnIndex) nella griglia di gioco.
  Nel caso in cui una delle condizioni venga soddisfatta, il metodo ritorna l'esito positivo del controllo e mostra di che tipo è la configurazione vincente;

- ```
  public boolean isItATie()
  ```

  Il metodo controlla se la partita è finita con un pareggio ossia se, dopo l'ultima mossa effettuata che non è risultata essere vincente, ci sono ancora spazi disponibili nella griglia per effettuare una nuova mossa o se questi sono tutti esauriti;

- ```
  public void playingGridPrinter()
  ```

  Il metodo è usato per la stampa della griglia di gioco;

- ```
  public int[][] playingGridToJSONPlayingGrid()
  ```

  Il metodo è utilizzato per tradurre la griglia di gioco (matrice di oggetti ColoredDisc) in una matrice di interi così da agevolare il salvataggio in un file JSON quando si vuole salvare lo stato della partita;

- ```
  public int[][]
  LoadJSONPlayingGridToIntegerPlayingGrid(String
  saveFile)
  ```

  Il metodo è usato in fase di caricamento della partita e si occupa di recuperare la matrice di interi salvata nel file JSON;

- ```
  public static JSONObject savingPlayingGrid(
  JSONObject gameData, int[][] JSONPlayingGrid)
  ```

  Il metodo è usato per il salvataggio della griglia di gioco che è stata precedentemente tradotta in una matrice di interi: i dati della griglia vengono salvati nell'oggetto JSON gameData che contiene tutti i dati della partita che si sta salvando in un dato momento del flusso di gioco.

2.2 Giocatore

Il giocatore è stato modellato nella classe **Player**.

2.2.1 Diagramma UML della classe

Viene qui di seguito presentato il diagramma UML della classe

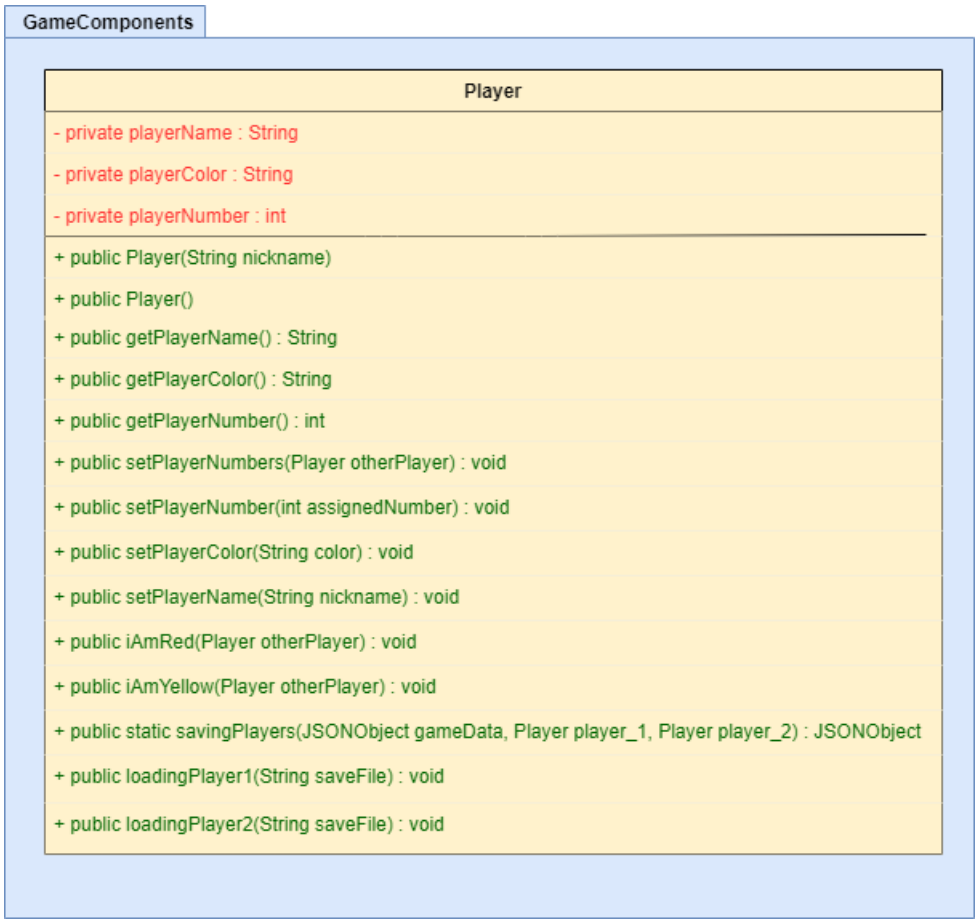

Figura 2: Diagramma UML della classe Player

2.2.2 Variabili di istanza

- ```
  private String playerName
  ```

  Il nome scelto dal giocatore ad inizio partita;
  ```

- ```
  private String playerColor
  ```

  Il colore delle pedine del giocatore;

- ```
  private int playerNumber
  ```

  Numero che indica l'ordine del giocatore nella partita.

2.2.3 Metodi

- ```
  public Player(String nickname)
  ```

  Uno dei costruttori della classe: questo costruttore viene invocato quando si sta creando un nuovo giocatore in una nuova partita; non appena creato l'unica informazione presente sul giocatore è il nome;

- ```
  public Player()
  ```

  Il secondo costruttore della classe: viene invocato quando si crea un giocatore i cui attributi verranno valorizzati con i dati di un giocatore il cui stato è stato salvato in una partita precedente;

- ```
  public String getPlayerName()
  ```

  Il metodo è utilizzato per recuperare il nome del giocatore;

- ```
  public getPlayerColor()
  ```

  Il metodo è utilizzato per recuperare il colore delle pedine del giocatore;

- ```
  public getPlayerNumber()
  ```

  Il metodo è utilizzato per recuperare la posizione nell'ordine di gioco del giocatore;
  ```

- ```
  public void setPlayerNumbers()
  ```

  Il metodo è utilizzato per assegnare ad inizio partita l'ordine di gioco dei giocatori;

- ```
  public void setPlayerNumber()
  ```

  Il metodo è utilizzato per assegnare un ordine di gioco dei giocatori già prestabilito in una partita precedentemente salvata;

- ```
  public void setPlayerColor()
  ```

  Il metodo è utilizzato per assegnare il colore delle pedine del giocatore che è stato precedentemente scelto da quest'ultimo in una partita precedentemente salvata;

- ```
  public void setPlayerName()
  ```

  Il metodo è utilizzato per assegnare il nome che il giocatore aveva scelto in una partita precedentemente salvata;

- ```
  public void iAmRed()
  ```

  Il metodo è utilizzato per assegnare il colore "rosso" al giocatore che può scegliere il colore delle proprie pedine e per assegnare il colore "giallo" al giocatore che non ha avuto modo di scegliere;

- ```
  public void iAmYellow()
  ```

  Il metodo è utilizzato per assegnare il colore "giallo" al giocatore che può scegliere il colore delle proprie pedine e per assegnare il colore "rosso" al giocatore che non ha avuto modo di scegliere;

- ```
  public static JSONObject savingPlayers(
  JSONObject gameData, Player player_1, Player
  player_2)
  ```

  Il metodo è utilizzato per salvare i dati relativi ai due giocatori della partita nell'oggetto JSON gameData in cui sono presenti tutti i dati relativi alla partita che si sta salvando in un dato momento del flusso di gioco;

- ```
  public void loadingPlayer1(String saveFile)
  ```

  Il metodo è utilizzato in fase di caricamento per recuperare i dati salvati relativi al giocatore 1 dal file JSON;

- ```
  public void loadingPlayer2(String saveFile)
  ```

  Il metodo è utilizzato in fase di caricamento per recuperare i dati salvati relativi al giocatore 2 dal file JSON;
  ```

2.3 Pedine

Le pedine di gioco sono state modellate nella classe **ColoredDisc**.

2.3.1 Diagramma UML della classe

Viene qui di seguito presentato il diagramma UML della classe

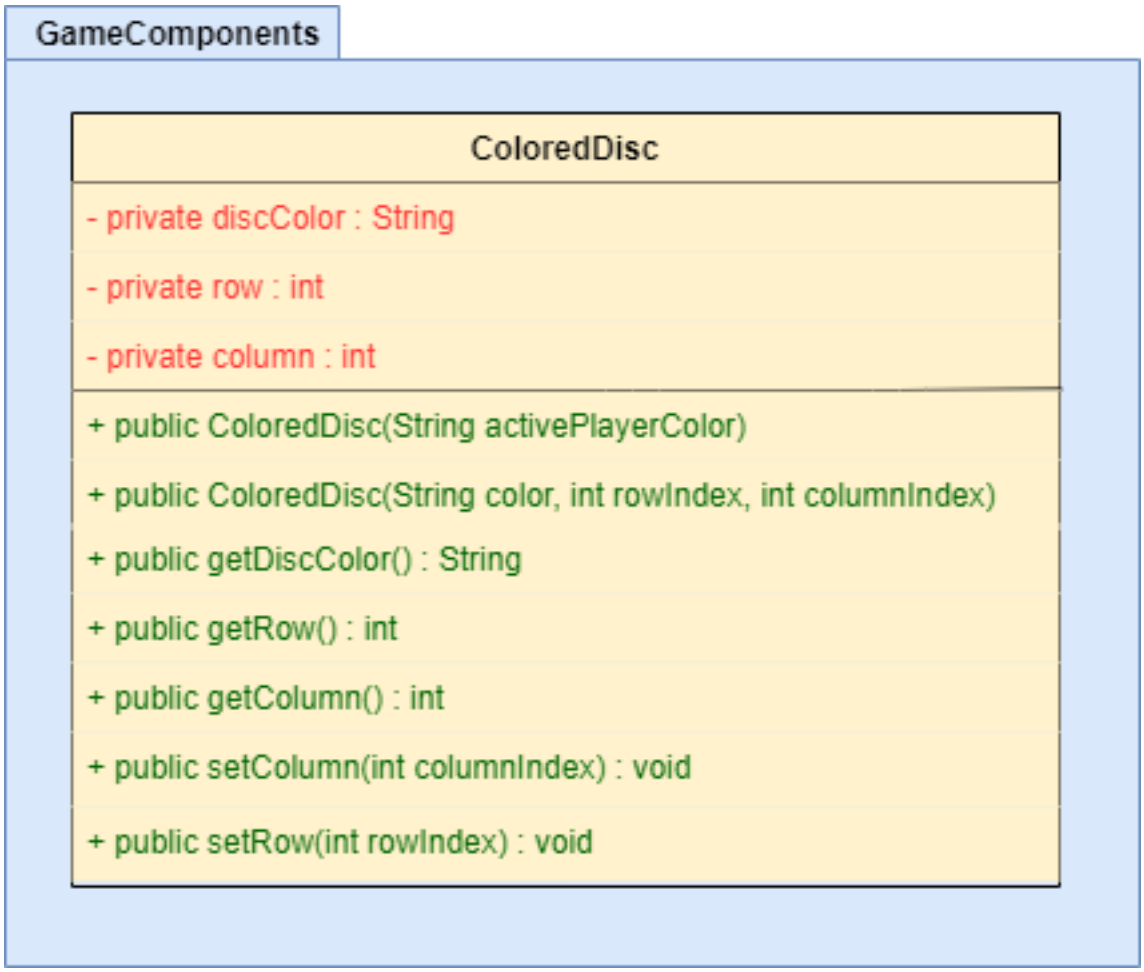

Figura 3: Diagramma UML della classe ColoredDisc

2.3.2 Variabili di istanza

- ```
  private String discColor
  ```

  La variabile indica il colore della pedina;

- ```
  private int row
  ```

  La riga in cui viene posizionata la pedina nella griglia di gioco;

- ```
  private int column
  ```

  La colonna in cui viene posizionata la pedina nella griglia di gioco.

2.3.3 Metodi

- ```
  public ColoredDisc(String activePlayerColor)
  ```

  Uno dei costruttori della classe: questo costruttore viene invocato ogni volta che un giocatore sta effettuando una nuova mossa, il colore della pedina creata è quello che il giocatore che sta effettuando la mossa ha scelto;

- ```
  public ColoredDisc(String color, int rowIndex,
  int columnIndex)
  ```

  Il secondo costruttore della classe: viene invocato durante il caricamento di una partita per ricostruire lo stato della griglia di gioco aggiungendo al posto giusto le pedine inserite dai giocatori nella partita precedentemente salvata;

- ```
  public String getDiscColor()
  ```

  Il metodo è utilizzato per ottenere il colore della pedina;

- ```
  public String getRow()
  ```

  Il metodo è utilizzato per ottenere la riga in cui si trova la pedina nella griglia;
  ```

- ```
  public String getColumn()
  ```

  Il metodo è utilizzato per ottenere la colonna in cui si trova la pedina nella griglia;

- ```
  public void setColumn()
  ```

  Il metodo è utilizzato per indicare la colonna della griglia in cui deve essere inserita la pedina;

- ```
  public void setRow()
  ```

  Il metodo è utilizzato per indicare la riga della griglia in cui deve essere inserita la pedina
  ```

2.4 Partita

La partita di Forza 4 è stata modellata nella classe **ConnectFourGame**.

2.4.1 Diagramma UML della classe

Viene qui di seguito presentato il diagramma UML della classe

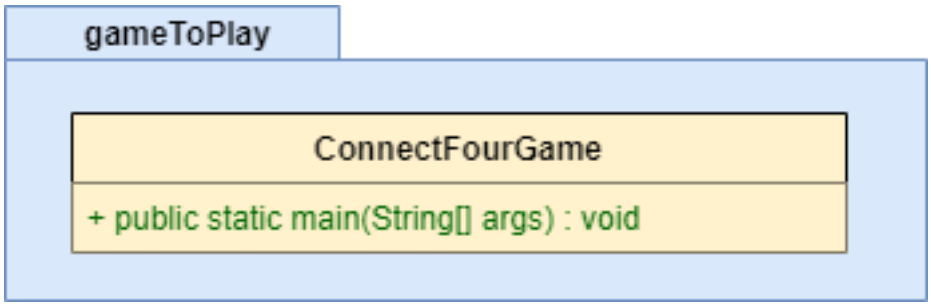

Figura 4: Diagramma UML della classe ConnectFourGame

2.4.2 Metodi

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

  Il metodo permette di giocare ad una partita di Forza 4; in particolare permette ai giocatori di scegliere se iniziare una nuova partita o caricarne una precedentemente salvata: nel caso in cui i giocatori decidano di iniziare una nuova partita il programma procede chiedendo ai giocatori i nomi, assegna loro un ordine e permette ai giocatori di scegliere il proprio colore; a turno, fin quando non si arriva ad una vittoria o ad un pareggio, poi viene permesso ai giocatori di eseguire la propria mossa; a termine della partita il programma mostra l'esito ottenuto dai giocatori.
  Nel caso in cui i giocatori decidano di caricare una partita salvata allora il programma si occupa di riprendere la simulazione della partita nell'esatto punto in cui i giocatori ne avevano salvato lo stato.

2.5 Eccezioni

Vengono ora descritte tutte le eccezioni personalizzate che sono state inserite nell'implementazione; tutte le classi di eccezione sono costruite nello stesso modo, quindi per evitare troppe ripetizioni viene qui di seguito dato un modello di implementazione di un'eccezione adottato per tutte le eccezioni inserite.
Viene inoltre fornito un diagramma UML generico applicabile a tutte le eccezioni inserite nel progetto.
Per ogni eccezione viene poi riportata l'occasione in cui può essere sollevata e la super-classe che estende.
  ```

2.5.1 Modello di implementazione

Ogni classe di eccezione ha al suo interno soltanto due costruttori:

- ```
  public nomeEccezione()
  ```

  Il costruttore senza parametri, viene invocato per creare un'eccezione che non mostra un messaggio all'utente

- ```
  public nomeEccezione(String message)
  ```

  Il costruttore con parametro una stringa, questo invoca il costruttore della super-classe e mostra un messaggio all'utente quando viene creata l'eccezione

2.5.2 Diagramma UML delle eccezioni

Al fine di evitare ripetizioni, proprio perchè le eccezioni hanno sempre la stessa struttura viene qui di seguito presentato un modello di diagramma UML per le classi di eccezione (si noti che in questo momento non viene rappresentata l'ereditarietà delle eccezioni che verrà mostrata quando si presenterà il diagramma UML della gerarchia delle classi)

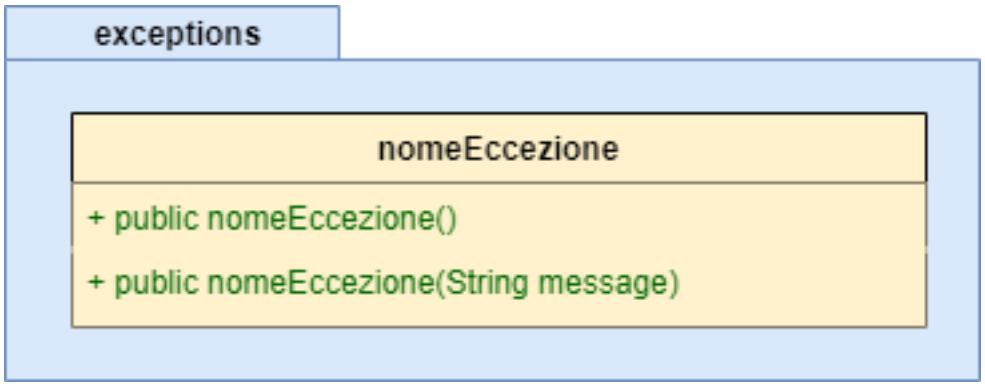

Figura 5: Modello del diagramma UML delle eccezioni

2.5.3 Lista delle eccezioni personalizzate

- ```
  DirectoryNotFoundException extends
  FileNotFoundException
  ```

  Questa eccezione viene sollevata durante una partita quando un utente del programma prova a mostrare i file di salvataggio andandoli a recuperare da una cartella che non esiste;

- ```
  FullColumnException extends
  IllegalArgumentException
  ```

  Questa eccezione viene sollevata quando si prova ad inserire una pedina in una colonna già completamente occupata;

- ```
  ColorAlreadyDecidedException extends
  IllegalArgumentException
  ```

  Questa eccezione viene sollevata quando l'utente del programma prova ad assegnare ai giocatori dei colori quando questi sono stati già assegnati in precedenza;

- ```
  IllegalAnswerException extends
  InputMismatchException
  ```

  Questa eccezione viene sollevata quando l'utente del programma durante una partita fornisce una risposta che non rientra nella forma aspettata dal programma;

- ```
  IllegalColumnException extends
  IllegalArgumentException
  ```

  Questa eccezione viene sollevata quando l'utente del programma prova ad inserire una pedina in una colonna che non esiste;
  ```

- ```
  IllegalPlayerException extends
  IllegalArgumentException
  ```

  Questa eccezione viene sollevata quando un giocatore non abilitato a compiere le scelte cerca di farlo: per essere più precisi, la scelta del colore del giocatore può essere effettuata solo dal giocatore #1, di conseguenza se il giocatore #2 prova ad effettuare la scelta l'eccezione viene lanciata;

- ```
  IllegalRowException extends
  IllegalArgumentException
  ```

  Questa eccezione viene sollevata quando un utente del programma prova ad inserire una pedina in una riga che non esiste;

- ```
  NumberAlreadyAssignedException extends
  IllegalArgumentException
  ```

  L'eccezione viene lanciata quando un utente del programma prova ad assegnare nuovamente l'ordine di gioco dei giocatori quando questo è già stato assegnato in precedenza.

2.6 Gerarchia delle classi

Viene ora presentato il diagramma UML che rappresenta la gerarchia delle classi così da mostrare l'architettura del progetto.
Poiché le singole classi sono già state presentate in maniera dettagliata in sede di descrizione delle stesse in questo diagramma le classi non verranno ripresentate con tutti i loro attributi e i loro metodi.
  ```

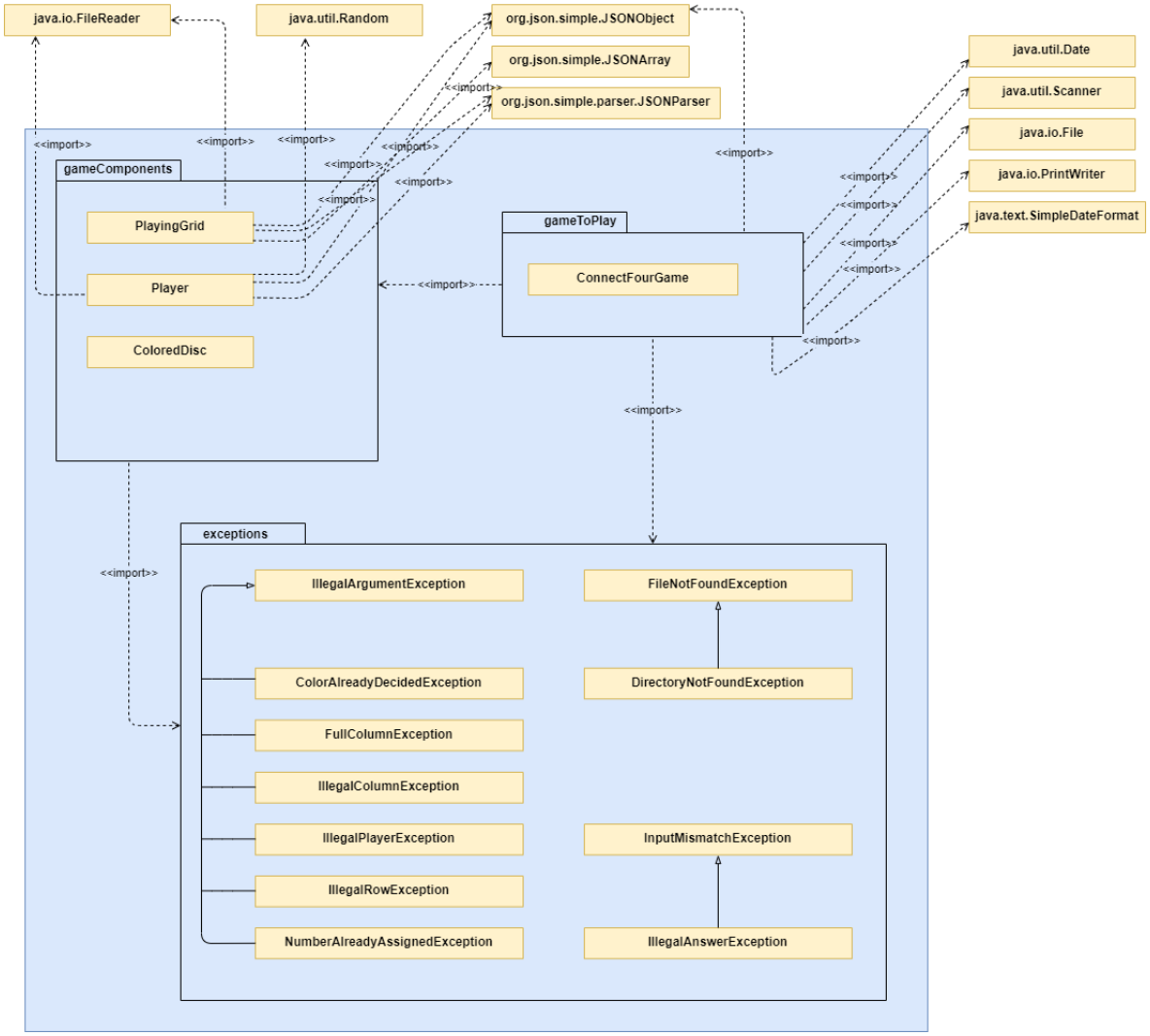

Figura 6: Diagramma UML della gerarchia delle classi

2.7 Classi di test

La correttezza dell'implementazione è stata testata tramite classi il cui compito è quello di mostrare il comportamento degli oggetti e della partita in situazioni critiche.
In particolare vengono ora descritte le situazioni affrontate in ogni classe:

- **PlayingGridTest**
  Nella classe viene testata la correttezza della classe PlayingGrid che modella la griglia di gioco.

Vengono testate la correttezza del costruttore che crea una griglia per una nuova partita, la correttezza del metodo per inserire una pedina nella griglia e il conseguente sollevamento dell'eccezione dovuto al tentativo di inserire una pedina in una colonna già piena, la correttezza del metodo che aggiorna il numero di spazi liberi nella griglia e il corretto funzionamento di tutti i metodi che controllano le possibili configurazioni di vittoria;

- **PlayerTest**
  Nella classe viene testata la correttezza della classe Player che modella un giocatore.
  Vengono testate la correttezza del costruttore che crea un giocatore per una nuova partita, la correttezza del metodo che assegna un ordine ai giocatori e il conseguente sollevamento dell'eccezione dovuta al tentativo di riassegnare un ordine ai giocatori quando questo è già stato assegnato, la correttezza dei metodi per scegliere i colori e il conseguente sollevamento delle eccezioni dovute al fatto che il giocatore che sta scegliendo il colore non è il giocatore #1 e il sollevamento dell'eccezione dovuta al fatto che il giocatore #1 sta provando a scegliere nuovamente il colore che è già stato scelto;

- **ColoredDiscTest**
  Nella classe viene testata la correttezza della classe ColoredDisc che modella una pedina di gioco.
  Vengono testate la correttezza del costruttore che crea una nuova pedina in una partita, la correttezza del costruttore usato per creare una pedina i cui attributi sono già tutti decisi e viene testata la correttezza dei metodi per assegnare una colonna e una riga alla pedina e il conseguente sollevamento delle eccezioni dovute al tentativo di assegnare valori al di fuori dei limiti della griglia;

- **GameDemo_1**
  Nella prima simulazione di gioco si testa il corretto sollevamento dell'eccezione dovuta ad una risposta errata del giocatore quando il programma chiede all'utente di scegliere se avviare una nuova partita o caricare una partita precedentemente salvata;

- **GameDemo_2**
  Nella seconda simulazione di gioco si testa il corretto sollevamento dell'eccezione dovuta alla mancanza della directory contenente i file di salvataggio;

- **GameDemo_3**
  Nella terza simulazione di gioco si testa il corretto sollevamento dell'eccezione dovuta alla richiesta del giocatore di caricare un file di salvataggio facendo riferimento ad un identificatore sbagliato;

- **GameDemo_4**
  Nella quarta simulazione di gioco si testa il corretto sollevamento dell'ec-

cezione dovuta alla richiesta del giocatore di caricare un file di salvataggio che non esiste;

- **GameDemo_5**
  Nella quinta simulazione di gioco si testa il corretto sollevamento dell'eccezione dovuta alla scelta di un giocatore di un colore non contemplato dal programma, ossia diverso da "red" o "yellow";

- **GameDemo_6**
  Nella sesta simulazione di gioco si testa il corretto sollevamento dell'eccezione dovuta ad una risposta errata di un giocatore durante il flusso di gioco;

- **GameDemo_7**
  Nella settima simulazione di gioco viene proposta un'ipotetica partita vinta dal giocatore #2

Si noti che in tutte le simulazioni di gioco proposte la partita si ferma non appena viene sollevata un'eccezione: questa scelta è dovuta solamente al fatto di voler mettere in luce una sola criticità per simulazione, nella vera partita il flusso, tranne quando si hanno problemi con il caricamento, non viene mai interrotto.

3 Descrizione delle funzionalità

In questa sezione dell'elaborato viene descritto come sono state riportate le regole del gioco nell'implementazione: per ognuna delle funzionalità viene anche mostrato un esempio di esecuzione.
Dopo aver preso in esame ogni funzionalità singolarmente viene riportato un esempio completo di partita.

3.1 Implementazione delle regole di gioco - Funzionalità -

- **Scelta tra nuova partita e caricamento**
  Come prima cosa all'avvio del programma viene chiesto ai giocatori se vogliono iniziare una nuova partita o caricarne una precedentemente salvata

```
WELCOME TO CONNECT 4
Hello player, do you want to begin a new game or load a previous one?
Write 'new' to begin a new game or write 'load' to load a previous one
```

Figura 7: L'utente può scegliere se iniziare una nuova partita o caricarne una salvata

  - **Inizio di una nuova partita**
    1. **Identificazione dei giocatori**
       Non appena viene iniziata una nuova partita viene chiesto ai giocatori di inserire i propri nomi per la partita

```
WELCOME TO CONNECT 4
Hello player, do you want to begin a new game or load a previous one?
Write 'new' to begin a new game or write 'load' to load a previous one
new
Hello player, what's your name?
Giocatore
And how's your rival named?
Rivale
```

Figura 8: Agli utenti viene chiesto a turno il proprio nome

    2. **Assegnazione dell'ordine di gioco**
       Scelti i nomi ai giocatori viene assegnato randomicamente l'ordine di gioco; la scelta di assegnare l'ordine in questo modo è dovuta al tentativo di emulare la conta che viene fatta per assegnare l'ordine nel gioco reale

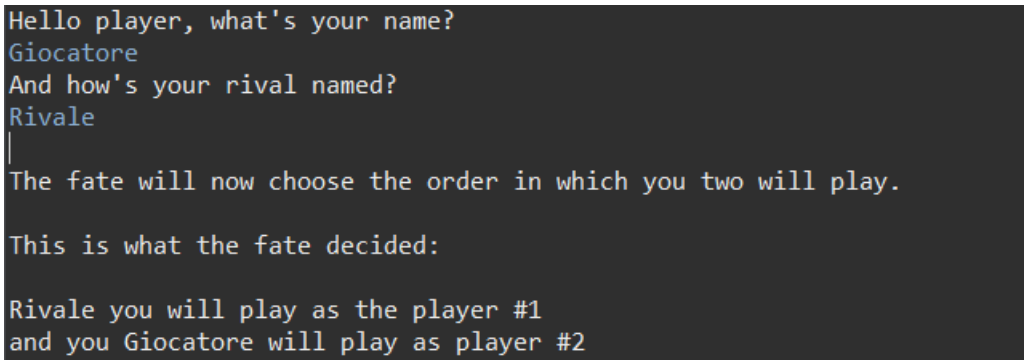

Figura 9: L'ordine è assegnato randomicamente

3. **Scelta colore e spiegazione del gioco**
   Assegnato l'ordine, il giocatore scelto come primo ha la possibilità di scegliere il colore delle proprie pedine, il colore del rivale viene assegnato di conseguenza.
   Scelto il colore, il regolamento del gioco viene presentato ai due giocatori

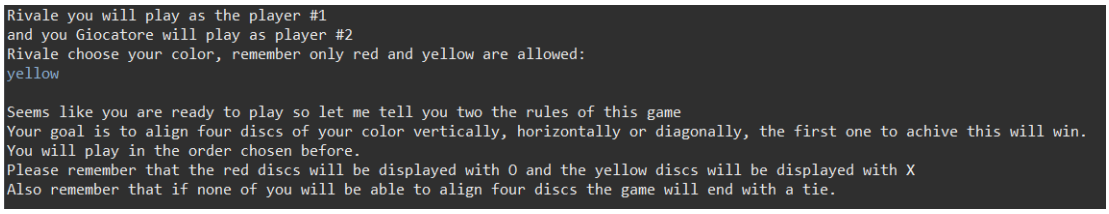

Figura 10: Il giocatore può scegliere tra "red" o "yellow"

4. **Partita**
   Scelte tutte le caratteristiche dei giocatori inizia la vera partita: a turno viene chiesto ai giocatori dove vogliono inserire la propria pedina, se vogliono salvare o se vogliono uscire dal gioco.
   Dopo ogni mossa effettuata viene mostrata la griglia aggiornata con la pedina inserita

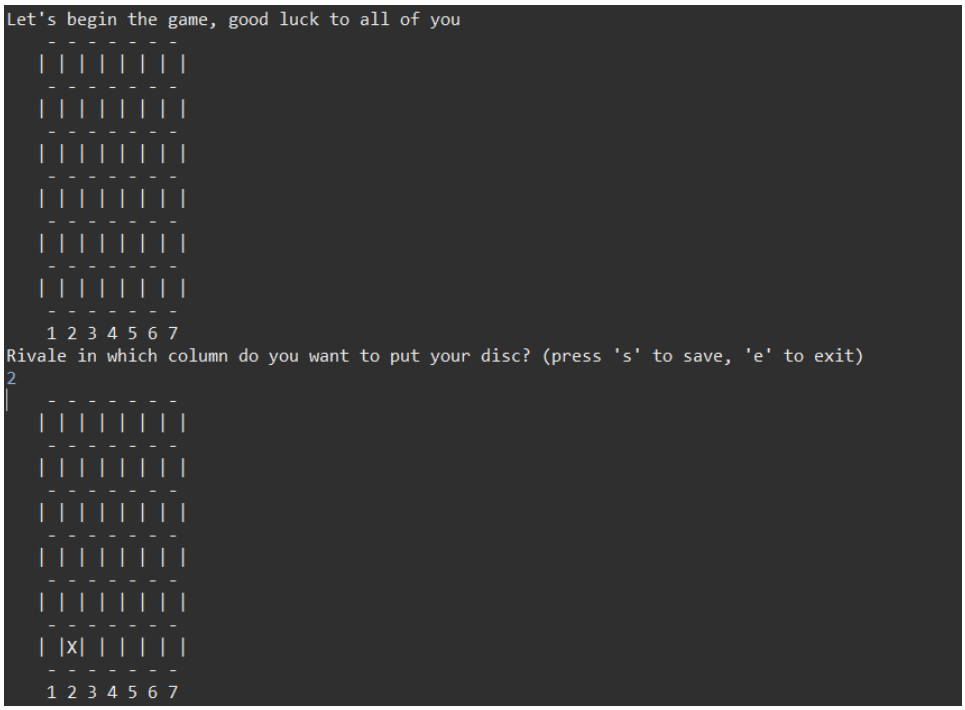

Figura 11: La pedina è stata inserita nella colonna 2

– **Caricamento di una partita salvata**

1. **Scelta del salvataggio da caricare**
   Se i giocatori decidono di caricare una partita precedentemente salvata, il programma procede con la richiesta di quale salvataggio caricare; agli utenti viene anche data la possibilità di non caricare nessuna partita e iniziarne una nuova

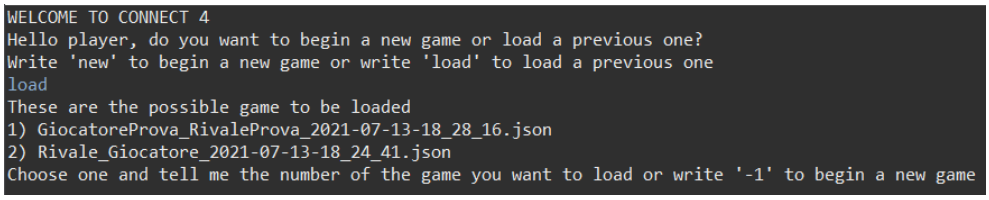

Figura 12: All'utente vengono mostrati tutti i salvataggi possibili da caricare

2. **Caricamento della partita salvata**
   L'utente inserisce quindi l'identificativo numerico collegato al salvataggio che vuole caricare: se l'identificativo inserito è presente nella lista mostrata all'utente e il salvataggio è presente nella cartella dei salvataggi, la partita viene caricata e la sua esecuzione riprende nel momento in cui era stata interrotta

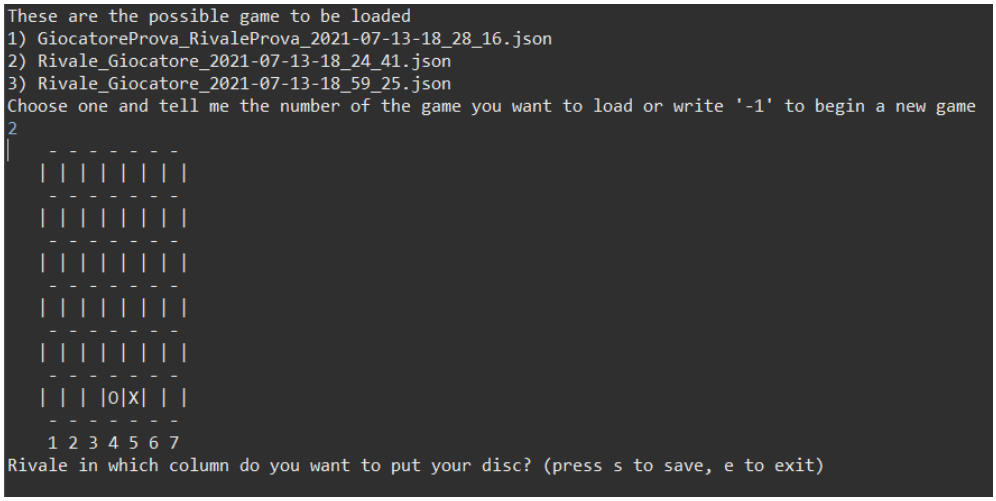

Figura 13: La partita riprende nel punto dell'ultimo salvataggio

3.2 Demo di gioco

Viene qui di seguito riportata una generica partita che può avvenire tra due giocatori.
La partita verrà iniziata come "nuova partita", dopo diverse mosse verrà salvata, il gioco verrà interrotto e poi verrà ricaricata e terminata.

```
WELCOME TO CONNECT 4
Hello player, do you want to begin a new game or load a previous one?
Write 'new' to begin a new game or write 'load' to load a previous one
new
Hello player, what's your name?
GiocatoreDemo
And how's your rival named?
RivaleDemo

The fate will now choose the order in which you two will play.

This is what the fate decided:

GiocatoreDemo you will play as the player #1
and you RivaleDemo will play as player #2
GiocatoreDemo choose your color, remember only red and yellow are allowed:
red
```

Figura 14: Viene creata una nuova partita

```
Seems like you are ready to play so let me tell you two the rules of this game
Your goal is to align four discs of your color vertically, horizontally or diagonally, the first one to achive this will win.
You will play in the order chosen before.
Please remember that the red discs will be displayed with O and the yellow discs will be displayed with X
Also remember that if none of you will be able to align four discs the game will end with a tie.

Let's begin the game, good luck to all of you
   - - - - - - -
   | | | | | | |
   - - - - - - -
   | | | | | | |
   - - - - - - -
   | | | | | | |
   - - - - - - -
   | | | | | | |
   - - - - - - -
   | | | | | | |
   - - - - - - -
   | | | | | | |
   - - - - - - -
   1 2 3 4 5 6 7
GiocatoreDemo in which column do you want to put your disc? (press 's' to save, 'e' to exit)
1
   - - - - - - -
   | | | | | | |
   - - - - - - -
   | | | | | | |
   - - - - - - -
   | | | | | | |
   - - - - - - -
   | | | | | | |
   - - - - - - -
   | | | | | | |
   - - - - - - -
   |O| | | | | |
   - - - - - - -
   1 2 3 4 5 6 7
```

Figura 15: Vengono effettuate le prime mosse

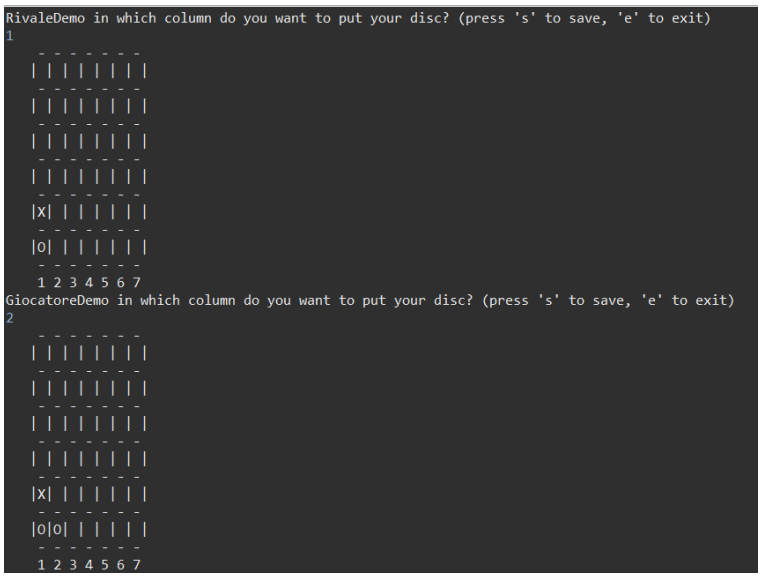

Figura 16: Vengono effettuate altre mosse

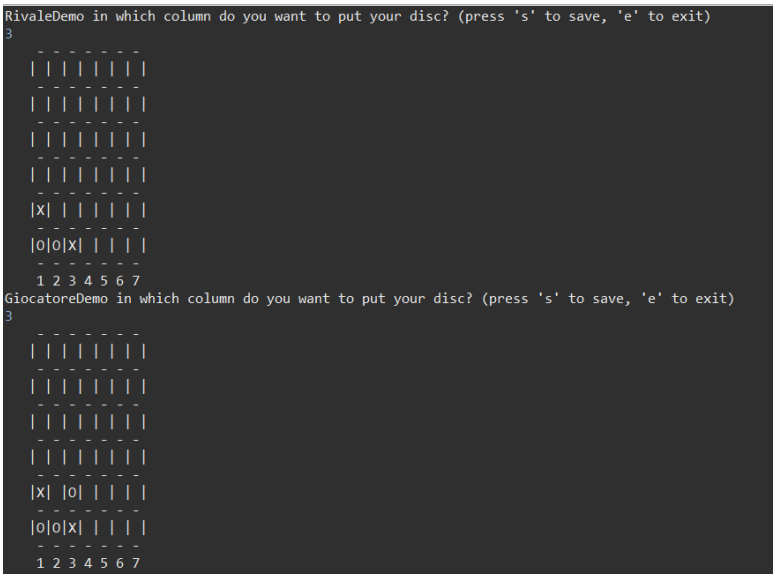

Figura 17: Vengono effettuate altre mosse

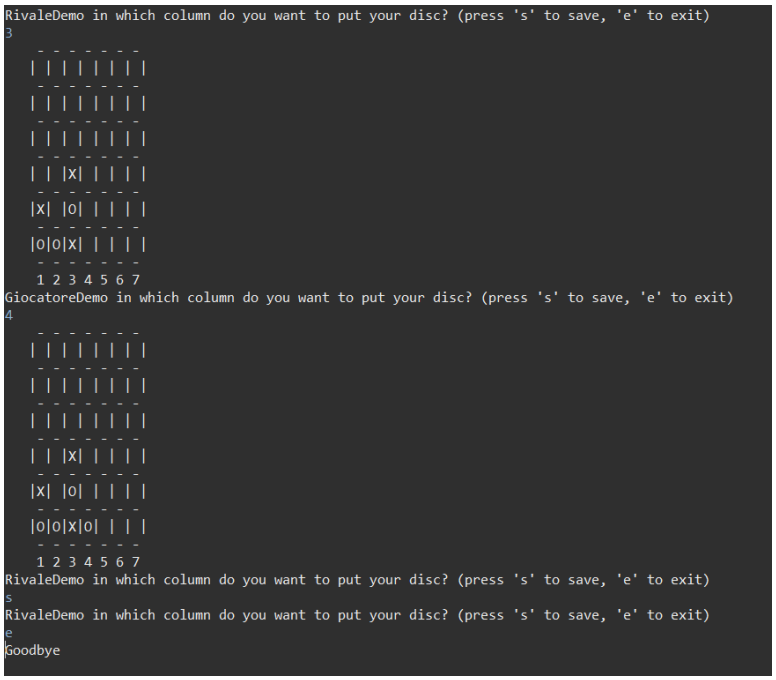

Figura 18: Vengono effettuate le ultime mosse prima del salvataggio, la partita viene salvata e si esce dal gioco

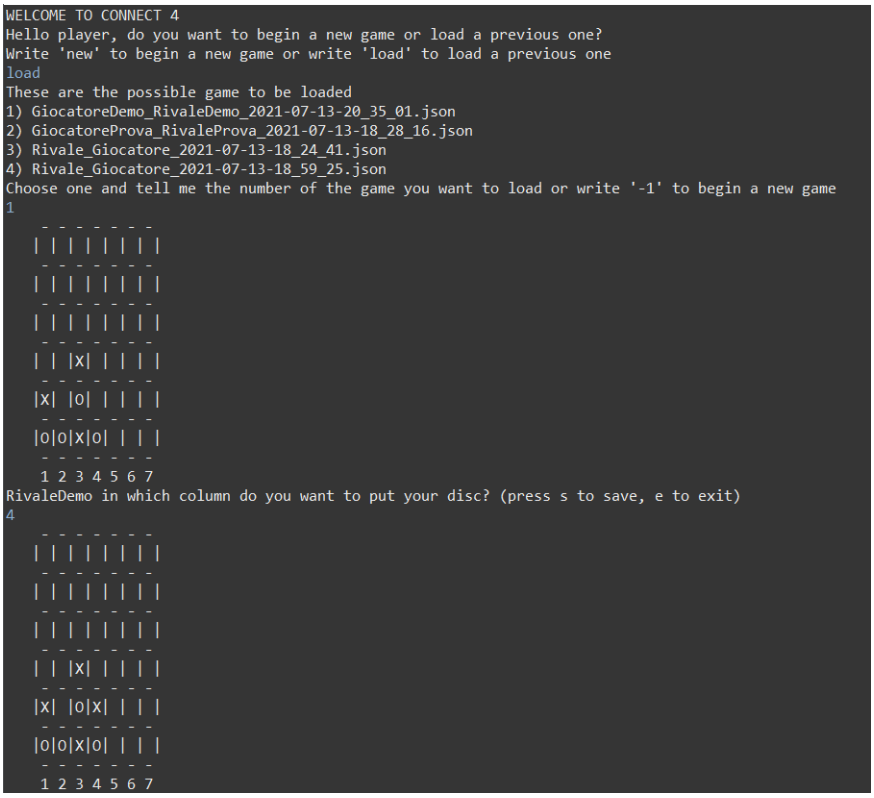

Figura 19: Il gioco viene riavviato e si sceglie di caricare una partita: la partita riprende dal punto in cui era stata interrotta

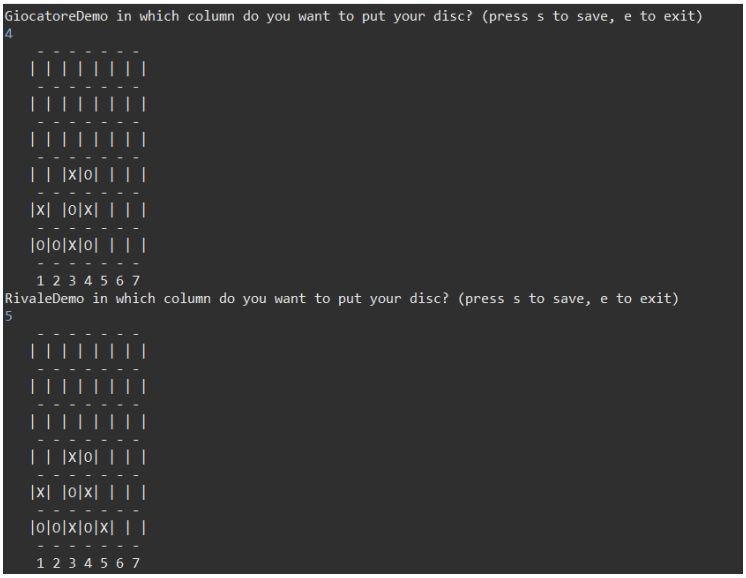

Figura 20: I giocatori continuano ad alternarsi nel gioco

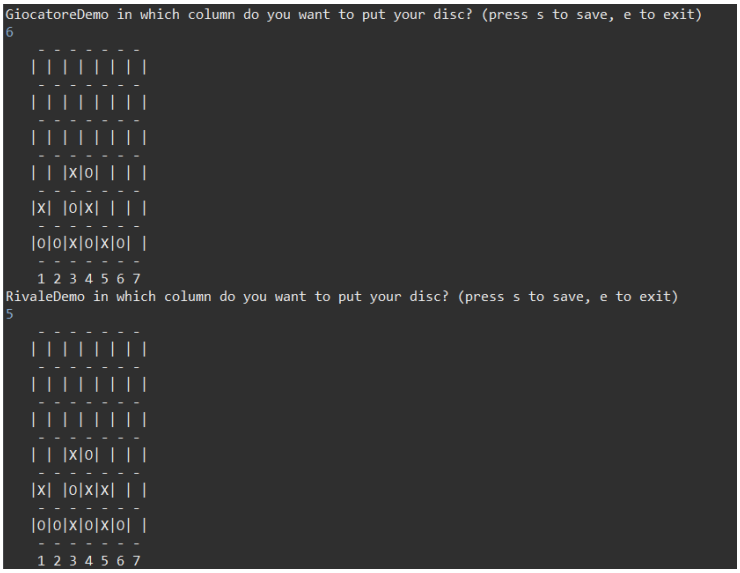

Figura 21: Vengono effettuate delle mosse

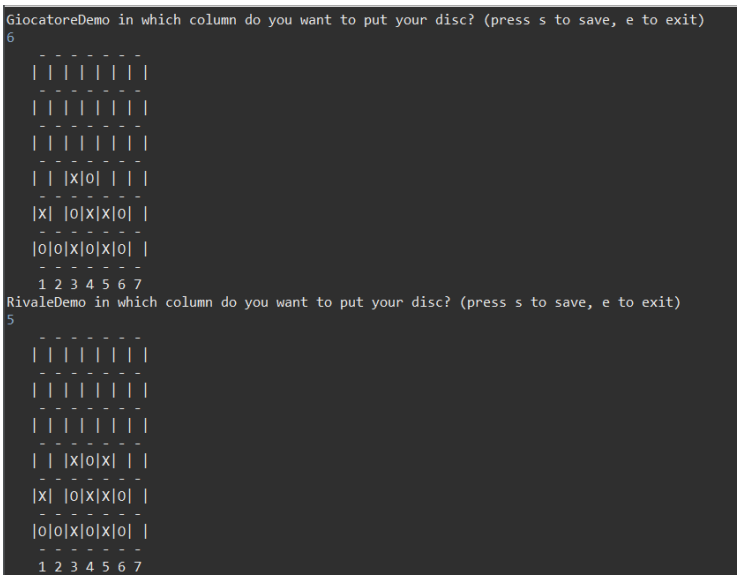

Figura 22: Vengono effettuate altre mosse

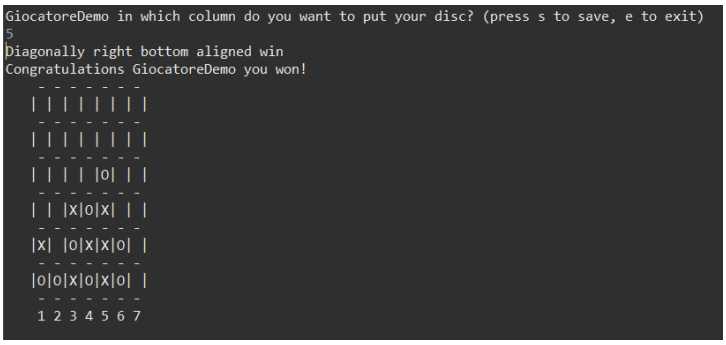

Figura 23: GiocatoreDemo effettua la mossa vincente: la partita termina e viene mostrata la configurazione vincente ai giocatori

3.3 Scelte di implementazione

Nell'implementazione presentata è stato scelto di modellare come oggetti le sole componenti di gioco in quanto si è pensato di ricalcare una partita che sarebbe avvenuta con il gioco in scatola nella vita reale dove, una volta montata la griglia e divise le pedine, i due giocatori avrebbero proceduto con il giocare tra di loro: è stato quindi scelto di collegare tutte le varie componenti di gioco nella partita vera e propria che è quindi stata implementata come il main su cui gira l'intero progetto.

Sempre per simulare una reale partita tra due giocatori è stato scelto di dare la possibilità agli utenti di scegliere il colore per le proprie pedine: in particolare la scelta può essere effettuata solo dal giocatore che è stato scelto casualmente (per simulare la conta che avverrebbe nella vita reale) come il primo ad effettuare le mosse.
Per gestire i turni tra i giocatori è stato fatto affidamento agli spazi liberi disponibili nella griglia: infatti se sono disponibili un numero pari di spazi liberi vuol dire che il giocatore primo nell'ordine di gioco deve effettuare una mossa, se sono disponibili un numero dispari di spazi liberi vuol dire che è il giocatore secondo nell'ordine di gioco a dover effettuare una mossa (basti pensare infatti che inizialmente prima che la prima mossa venga effettuata sono disponibili 42 spazi liberi e subito dopo questa mossa ne sono disponibili 41); questo meccanismo è stato poi sfruttato anche nel caricamento della partita così da poter riprendere il flusso di gioco dall'ultimo giocatore che non aveva ancora effettuato una mossa.
La forza di gravità è stata gestita pensando al fatto che quando una pedina viene inserita in una colonna è come se controllasse per ogni cella di quella colonna se nella cella è presente una pedina, continuando a scendere fino a quando non raggiunge la fine della griglia o l'ultima pedina inserita al suo interno.
Per quanto riguarda il salvataggio della partita è stato scelto di salvare i dati relativi allo stato della griglia e ai giocatori in un file JSON il cui nome viene costruito concatenando i nomi dei giocatori con la data del giorno in cui viene salvata la partita con l'orario in cui viene avviata: in questo modo è possibile salvare più partite che vedono protagonisti gli stessi giocatori (o giocatori che hanno scelto gli stessi nomi) e i salvataggi vengono resi distinguibili tra di loro; il primo salvataggio della partita viene effettuato non appena questa viene avviata e ogni volta che l'utente sceglie di salvare, lo stesso file JSON viene sovrascritto con il nuovo stato della griglia.
La partita quindi va avanti fino ad esaurimento dei posti liberi nella griglia (si tratta di un pareggio) o fino a vittoria di uno dei due giocatori, in ogni caso viene mostrato agli utenti il motivo che ha portato alla conclusione della partita: in caso di pareggio viene mostrato semplicemente un messaggio ai giocatori che attesta quanto avvenuto; in caso di vittoria viene segnalato quale è stata la configurazione di vittoria (allineamento verticale, orizzontale verso destra, orizzontale verso sinistra, sulla diagonale sinistra verso il basso, sulla diagonale sinistra verso l'alto, sulla diagonale destra verso il basso, sulla diagonale destra verso l'alto) e viene mostrata la griglia con le quattro pedine allineate.
Le configurazioni di vittoria vengono testate effettuando controlli sulla matrice che rappresenta la griglia di gioco.
Non potendo simulare i colori su console è stato scelto di rappresentare le pedine rosse con O e quelle gialle con X.

4 Referenti di sviluppo

Il progetto è stato realizzato dalla studentessa Lucia Fores (matricola 1836451) del corso di Metodologie di Programmazione per l'AA.2020-2021.