JConnect4

Francesco James Fanti; 1911623

Giugno 2022

Contents

1 Introduzione

La sfida propostami per il progetto è stata quella di creare una applicazione per riga di comando in stile retro-games.
Il progetto consiste in una semplice applicazione che permette di giocare al classico gioco analogico *forza 4* (connect4) in due giocatori.
Si può anche giocare contro un giocatore PC.

1.1 Note

Gli screenshot allegati rappresentano una versione beta del gioco, ci sono state leggere variazioni dal punto di vista prettamente grafico.

Figure 1: Inizio di una nuova partita

2 Classi

LEGENDA:

La seguente sezione è una descrizione molto dettagliata di classi con attributi e metodi; i tipi di dato sono, talvolta, omessi al fine di ottenere una spiegazione che trascenda le specifiche del linguaggio di programmazione, ma contempli il ragionamento progettuale che c'è dietro.
Per i metodi ho utilizzato la seguente sintassi:
nomeMetodo(eventuali dati in pseudocodice).

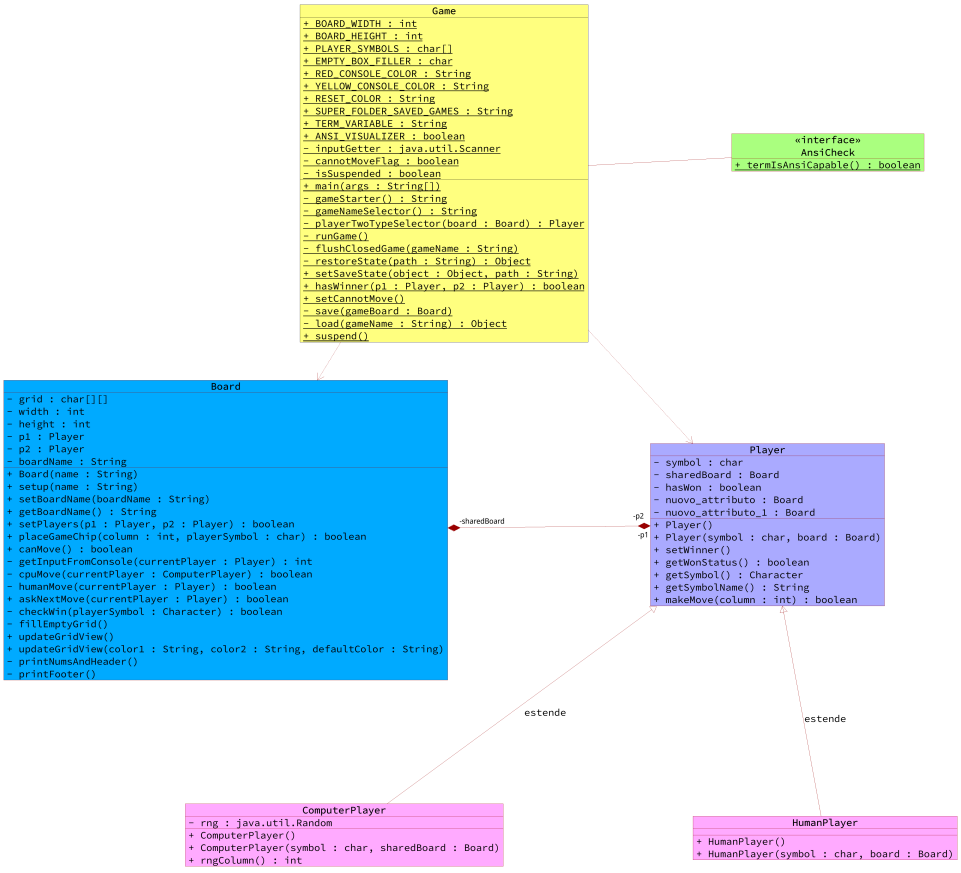


Figure 2: Diagramma UML

2.1 Board

Classe che rappresenta la tabella di gioco, con rispettivi attributi e metodi volti allo svolgimento della partita.

Attributi:

- una griglia (array bidimensionale)
  in cui i giocatori porranno le loro pedine.
- due istanze di giocatori (Player)
  che giocheranno la partita.
- le dimensioni della griglia.
- il nome della griglia di gioco
  (che rappresenterà il nome dell'eventuale file di salvataggio).

Metodi:

- **Costruttore**
  La classe è fornita di un costruttore che prende in input un parametro name,questo viene passato successivamente al metodo setup().
  Successivamente inizializza l'oggetto con altezza e larghezza (in questo caso usando due costanti; il programma si apre alla possibilità di overload del costruttore per prendere in input larghezze e altezze diverse; tale estensione richiederebbe controlli sul conteggio dei punti e sulla posizione delle chip sulla board).
- **setup**
  chiama tre metodi per impostare la griglia di gioco, popolando l'array con caratteri rappresentanti le caselle vuote e assegnando valori ad attributi:

  1. il **setter** setBoardName
     Che prende in input la stringa nome e imposta l'attributo boardName con la seguente stringa.
  2. fillEmptyGrid
     Che scorre ogni casella della griglia e la "riempie" con un dato carattere usato per rappresentare una casella vuota; in questo caso ho utilizzato l'asterisco.

3. updateGridView()
   Che incapsula due metodi privati (che si occupano di finezze estetiche e saranno trattati successivamente):
   (a) printNumsAndHeader()
   (b) printFooter()

   esistono due versioni (in overload) di questo metodo, uno che stampa le chip colorate e una che le stampa senza colore. Siccome la griglia appena inizializzata è vuota (quindi sicuramente non ha chip colorate sopra, posso usare questo metodo senza controllare se il terminale su cui eseguo supporta i colori oppure no occupandomene soltato al prossimo aggiornamento della griglia) Inizialmente avevo utilizzato un metodo display() che faceva la stessa cosa, ma ho preferito riutilizzare codice già scritto per le sezioni successive.

- updateGridView(colori)
  Overload del metodo sopra che consente di chiamarlo facendogli stampare i caratteri dei player colorati se il terminale supporta gli escape code ANSI in questo caso nell'operazione di print circonda gli elementi da stampare con gli escape code e resetta il colore a quello standard quando non sta stampando elementi colorati (resettando sempre alla fine per evitare casi limite).

- getBoardName()
  Getter, ritorna il nome della board.

- setPlayers(giocatori)
  Controlla che i giocatori passati abbiano simboli differenti, in caso positivo li assegna all'istanza della board e torna true, sennò torna false.

- placeGameChip(colonna, simboloGiocatore)
  Prende in input una colonna e il simbolo di un giocatore (che riempirà quella colonna) controlla che la colonna sia in un range adeguato, in caso positivo prova a porre la chip (il simbolo) del giocatore nel punto piu basso di quella colonna e, se viene trovato un simbolo, sale di uno finché quella condizione è vera. Se la colonna non è piena il simbolo viene posizionato e viene tornato true, altrimenti è lanciata una IndexOutOfBoundException.

- canMove()
  Controlla se la board è piena, per farlo controlla che la riga più alta della board abbia almeno uno spazio vuoto (la griglia si riempie dal basso verso l'alto, quindi in un caso limite le ultime mosse possibili saranno per forza fatte sull'ultima riga). Se la board è piena viene controllato se almeno un giocatore ha vinto, altrimenti stampa un messaggio in cui è dichiarato nessun vincitore.

- getInputFromConsole(giocatore)
  Si occupa di prendere da console l'input dell'utente (un giocatore), il giocatore ha due scelte:

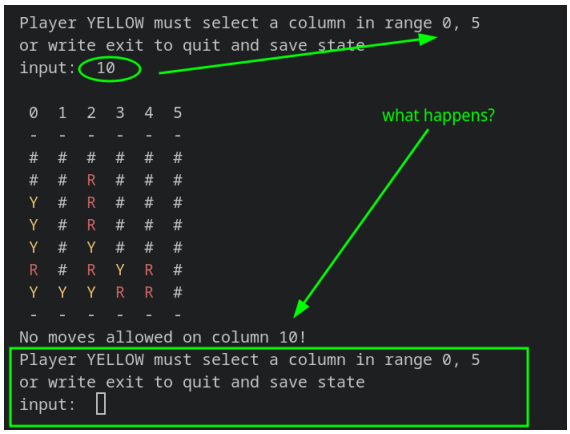
  - selezionare una colonna (dove eseguire placeGameChip)
  - uscire e salvare lo stato della partita su un file.
    Se viene selezionata una colonna ritorna il numero della colonna, se viene selezionato di uscire ritorna -1, se nessuna condizione dovesse essere rispettata torna -4.
    La scelta del tipo numerico come tipo di ritorno è stata dettata dalla possibile eventualità di gestire piu di un tipo di errore o situazione diversa dall'inserimento del chip sulla board. L'uso di numeri negativi mi è sembrato utile perché nessuna colonna negativa verrà potenzialmente mai inserita in input, essendo la griglia numerata tramite numeri positivi (un controllo ulteriore sull'input di quei numeri specifici risolverebbe anche il caso accidentale di inserimento).

- cpuMove()
  Tramite un generatore casuale in un range di colonne (alcune possibili, altre che lo costringerebbero a riprovare) genera una colonna in cui inserire la chip.
  Controlla se il giocatore ha vinto, in caso positivo lo segna come vincitore e torna false, sennò torna true.

- humanMove()
  Prende input da console tramite getInputFromConsole()
  salva questo dato in due variabili
  (per comodità di lettura: non avrebbe senso, in caso getInputFromConsole tornasse -1, chiamarlo column):

1. in caso di -1 [”exit”]
   sospende il gioco e torna false.
2. in caso di -4 [errore generico]
   stampa che c’è stato un errore, sospende e torna false.
3. altrimenti aspetta che venga passata una colonna valida,
   (a) se il player non ha vinto torna true,
   (b) altrimenti lo setta vincitore e torna false.

- askNextMove(giocatore) Si occupa di chiedere al giocatore uno [p1] la mossa da eseguire, volta per volta.
  Controlla se la board non è piena (torna false e interrompe, in quel caso).
  Successivamente controlla se il giocatore è un giocatore vero o un giocatore cpu, in questo caso chiede una cpuMove all’istanza di ComputerPlayer[p2]
  altrimenti chiede una HumanMove all’istanza di HumanPlayer[p2].
  Il risultato viene dalla chiamata di un metodo ed è assegnato ad una variabile locale (moveResult)
  torna moveResult.



- fourChecker(posizioni da controllare, target)
  Prende in input i caratteri di quattro posizioni sulla griglia, torna vero se sono uguali al simbolo target, falso altrimenti.

- checkWin(simbolo giocatore)
  Controlla se esiste una combinazione vincente (fila di 4 simboli uguali) in verticale, orizzontale, e nelle due direzioni diagonali della griglia. La griglia viene analizzata nei range in cui queste combinazioni sono possibili:

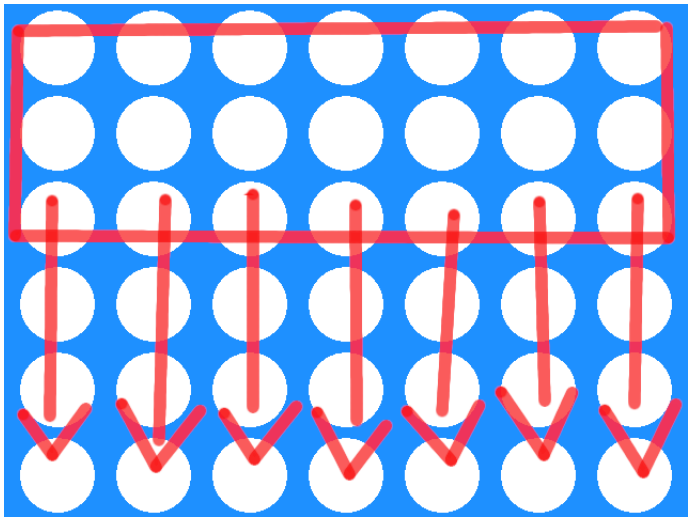


Figure 3: Controllo verticale.

1. In verticale quindi si ferma 3 posti prima del bordo, per scorrere dal punto x al punto x+3 verso il basso per vedere *slice di 4*.
2. In orizzontale si ferma 3 posti prima del bordo verso sinistra per scorrere verso destra *in slice di 4*.
3. Sulle diagonali scorre soltanto gli indici in cui è possibile una diagonale lunga 4 e controlla *slice diagonali di 4* nelle due direzioni.

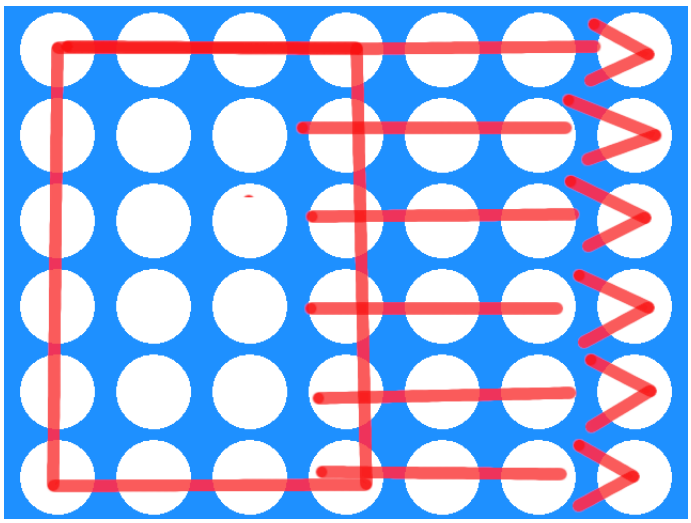


Figure 4: Controllo orizzontale.

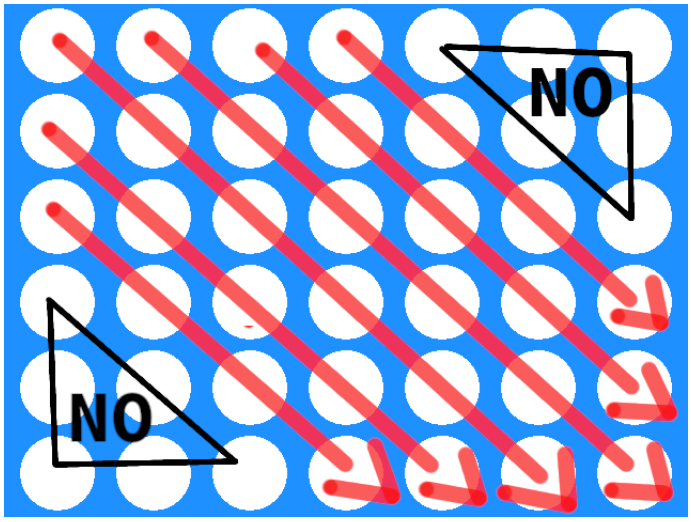


Figure 5: Controllo diagonale discendente.

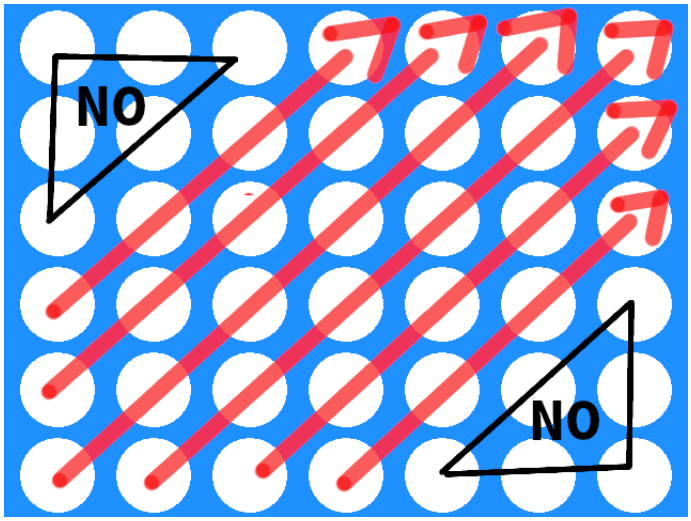


Figure 6: Controllo diagonale ascendente.

Ad ogni chiamata del metodo viene controllato se tutte le posizioni negli slice analizzati (che ricordiamo essere slice sull'array 2d "grid", attributo di questa classe) corrispondono al simbolo cercato, in caso positivo torna true [*il gioco ha un* ***vincitore***], altrimenti false.

2.2 Player

La classe rappresenta un giocatore della partita, in senso generico, verrà specializzato successivamente in giocatori umani o giocatori pc.

Attributi:

- un simbolo identificativo,
- l'istanza della Board su cui si svolge il gioco,
- un flag per la vittoria del giocatore.

Metodi:

- **Costruttore**(simbolo e board)
  Setta il simbolo e la board con i parametri passati e imposta il flag di vittoria a falso.
- setWinner()
  Rende vera la flag di vittoria.
- getWonStatus()
  Getter per la flag, ritorna la flag hasWon.
- getSymbol()
  Ritorna il simbolo del giocatore.
- getSymbolName()
  Controlla il simbolo del giocatore, assumendo che il gioco abbia solo due giocatori *in quanto connect4 ha sempre 2 giocatori*, controlla se il simbolo corrisponde al giocatore giallo, se sì torna "YELLOW" altrimenti torna "RED".
- makeMove(colonna)
  Salva in una variabile il risultato del metodo placeGameChip dellistanza della board su cui si trova il giocatore; torna la variabile (aggiornando la vista della griglia a seconda della disponibilità o no dei colori ANSI).

2.3 HumanPlayer

Sottoclasse di Player; possiede l'attributo esclusivo **name** che rappresenta il nome del giocatore.

Attributi:

- name
  Il nome del giocatore. Viene stampato insieme al simbolo mentre si gioca o se il giocatore ha vinto; è esclusivo di giocatori umani.

Metodi:

- getName()
  Getter; restituisce il nome del giocatore;
- askName()
  chiede al giocatore il suo nome e imposta l'attributo con il dato raccolto.

2.4 ComputerPlayer

Sottoclasse di Player; possiede un metodo per simulare le mosse utente.

Metodi:

- rngColumn()
  Usa un generatore di interi (in un dato range) per generare un intero rappresentante la colonna su cui il giocatore cpu inserira una chip, può anche selezionare colonne non valide, il resto del programma gestira la situazione.
  Torna l'intero.

2.5 AnsiCheck

Interfaccia che ha un metodo statico per verificare se il terminale su cui gira il gioco ha la possibilità di interpretare gli escape code ANSI. per farlo controlla se la variable d'ambiente "TERM" esiste (su terminali windows non esiste) e controlla che esista System.console() (negli ide non esiste) così ho la certezza di porvare a stampare testo colorato solo su terminali unix (che de facto supportano tutti gli ansi color).

2.6 GameSaver

La classe definisce un oggetto che funge da wrapper per gli oggetti da serializzare e deserializzare; l'istanza viene costruita tramite il metodo save() della classe Game e deserializzata tramite il metodo load(). Gli oggetti di questo tipo hanno tre getter, che servono a inizializzare gli oggetti dichiarati nei metodi principali con i dati deserializzati dai file di salvataggio.

Attributi:

- l'istanza di Board del game corrente;
- l'istanza di Player *p1*;
- l'istanza di Player *p2*.

Metodi:
Un costruttore che crea un'oggetto con i tre attributi inizializzati; un getter per ogni attributo.

2.7 Game

La classe che rappresenta lo svolgimento del gioco e che contiene il main.

Costanti:

- larghezza, altezza e simboli dei giocatori;
- il simbolo che rappresenta la box sulla board vuota;
- gli escape code ANSI;
- un percorso indicante la cartella dove salvare i giochi interrotti;
- il contenuto della variabile d'ambiente *TERM* (per AnsiCheck);
- contenitore booleano con il risultato del controllo di AnsiCheck.

Attributi:

- uno scanner, per la gestione dell'input utente;
- un flag di impossibilità di movimento;
- un flag di sospensione.

Metodi:

- gameStarter()
  Si occupa di operare le azioni preliminari di starting del gioco:
  1. crea, se non esiste, la cartella dove salvare i file e/o la legge per vedere se ci sono file; elenca l'eventuale lista i file;
  2. permette la scelta di iniziare una nuova partita o, se possibile, riprendere un vecchio gioco.
- gameNameSelector()
  Chiede da console un nome per il nuovo gioco iniziato (che verrà assegnato alla board), ritorna il nome.
- playerTwoTypeSelector(board)
  Permette di selezionare il tipo di Player del giocatore2, può essere un vero giocatore o un giocatore pc.
- runGame()
  Metodo cardine del gioco, usato lungo il main.
  1. Chiama gameStarter() *ed eventualmente imposta il nome della board se è nuova, altrimenti carica quella salvata.*
  2. Costruisce i giocatori e li assegna alla board (se possibile, ovvero se non hanno simboli uguali) iniziano i turni;
     goOn, la variabile che si occupa di mandare avanti il gioco contiene il risultato booleano restituito da setPlayers() [vedi sopra].
     Finché [*while*] una delle tre condizioni non viene rotta:
     (a) non si ha un vincitore;
     (b) il gioco non è sospeso;
     (c) goOn contiene *falso*

si chiede una mossa per volta ai giocatori.
A while chiuso controlla il da farsi:

- se il gioco non è sospeso ma la board è piena, cancella eventuali salvataggio del gioco (in quanto non più giocabile);
- il gioco è sospeso, salva il gioco su un file e chiude

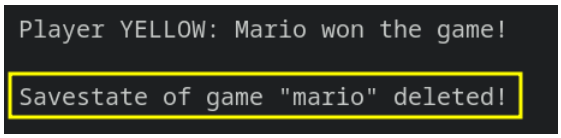


Figure 7: Il salvataggio è stato eliminato post controllo.

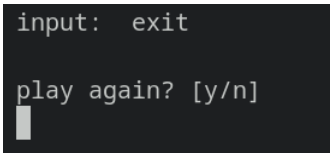


Figure 8: La partita è stata salvata dopo l'uscita.

Figure 9: è possibile giocare a ripetizione, tutte le volte che si vuole.

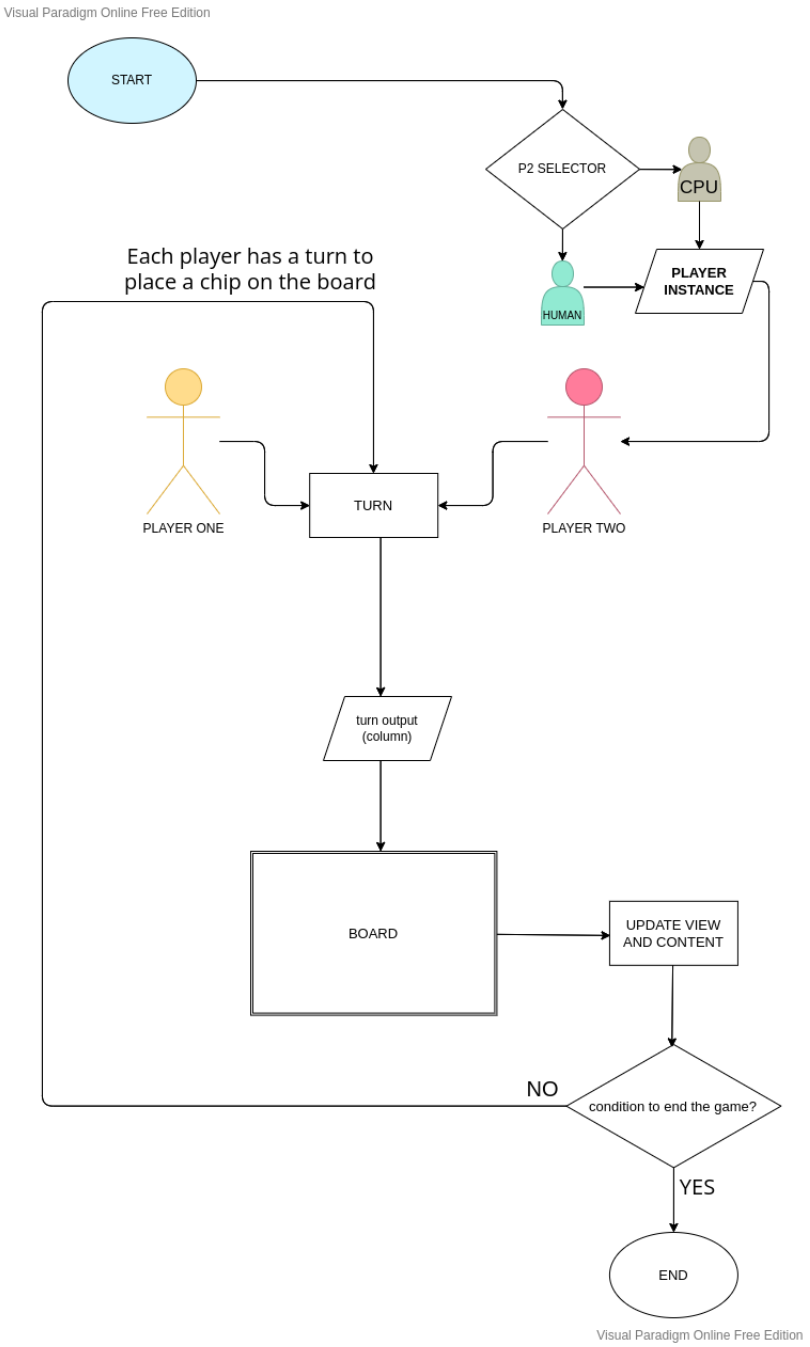
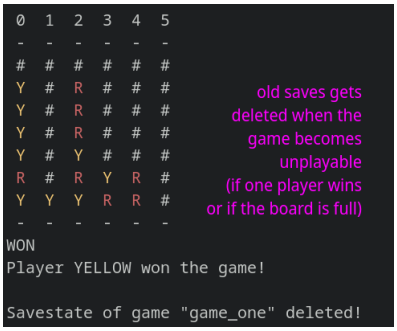


Figure 10: Diagramma di flusso della logica dietro runGame()

- flushClosedGames(nome gioco)
  Elimina il file il cui nome è stato passato come parametro.



- restoreState(path)
  Prende dati da un file sotto forma di fileinputstream (flusso di byte grezzi), ne ricava uno stream di dati di oggetto (objectinputstream) da cui viene deserializzato un oggetto con .readObject().
  Viene tornato l'oggetto (in tipo Object, da castare).

- setSaveState(oggetto, path)
  Serializza tramite un object output stream i dati dell'oggetto passato su un file, tramite fileinputstream; l'operazione avviene con .writeObject(object).

- hasWinner(giocatori)
  Controlla se uno dei due giocatori ha vinto, se sì stampa chi ha vinto e torna vero, falso altrimenti.

- setCannotMove()
  Setta true la flag di impossibilità movimento.

- save(board, p1, p2)
  Costruisce un oggetto GameSaver con i parametri passati, lo salva nel path ottenuto dalla cartella di salvataggio+nomeDellaBoard.

- load(nome gioco)
  Ripristina lo stato di un gioco salvato e torna l'oggetto deserializzato.

- suspend()
  Setta la flag di sospensione a true.

3 Funzionalità

3.1 Avvio

L'applicazione è avviabile tramite riga di comando:

1. previa compilazione:
    - `cd path/to/src`
    - `javac -d path/to/out Game.java`
    - `cd path/to/out`
    - `java Game`
2. oppure da file jar:
    - `cd path/to/jar`
    - `java -jar nomeApp.jar`

Il gioco gira su qualsiasi emulatore di terminale moderno e adatterà di conseguenza l'utilizzo dei colori (che non verranno visualizzati su terminali che non supportano le sequenze di escape ANSI).
Nonostante il gioco sia giocabile e godibile in ogni caso, consiglio l'utilizzo di un emulatore di terminale che supporti gli escape ANSI. Per i miei screenshot dimostrativi ho usato Alacritty.

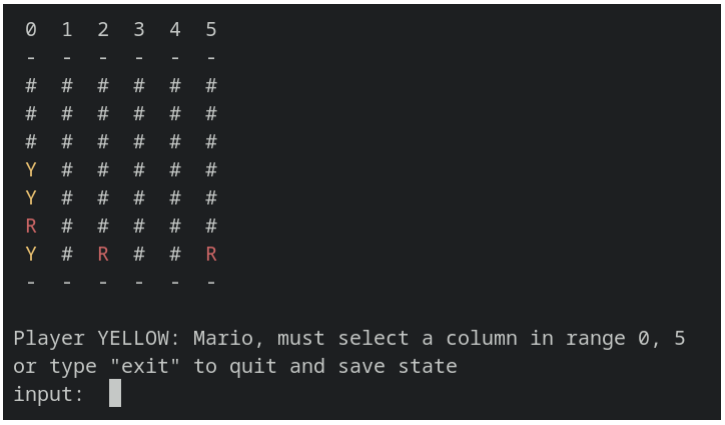


Figure 11: Eseguito su Alacritty, i giocatori hanno simboli colorati.

```
0  1  2  3  4  5
-  -  -  -  -  -
#  #  #  #  #  #
#  #  #  #  #  #
#  #  #  #  #  #
Y  #  #  #  #  #
Y  #  #  #  #  #
R  R  #  #  #  #
Y  R  #  #  #  #
-  -  -  -  -  -

Player YELLOW: Luigi, must select a column in range 0, 5
or type "exit" to quit and save state
input:
```

Figure 12: Eseguito sul terminale integrato di Eclipse IDE, niente colori.

3.2 Giocare

```
james@SpaceLink:~/eclipse-workspace/neoJConnect4/neoJConnect4/src/out(master⚡ ) » java Game
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on

====  WELCOME TO JConnect4  ====

=== there are no saved games ===

---> Choose a name for this gameboard <---
you can suspend the game and play again on it later
(please don't use spaces in your name but chars like "_" as a separator)

name:
```

Figure 13: Il primo avvio.

Avviando il gioco verrà controllato se esistono vecchie partite salvate, se sì, si potrà scegliere di digitarne il nome per riprenderle; altrimenti si potrà avviare una nuova partita scrivendo "new" e poi inserendo il nome della partita.
Se non ci sono partite salvate basterà scrivere il nome della partita (che corrisponde al nome della tabella di gioco) per iniziare a giocare!

Partita

Lo scopo di ogni partita è allineare quattro simboli uguali in orizzontale, in verticale o su una delle diagonali della tabella; vince il primo giocatore che ci

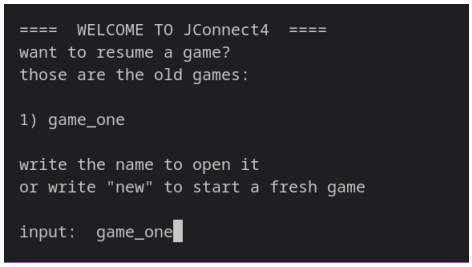


Figure 14: Il gioco *game_one* è salvato in memoria e sta per essere ripreso.

riesce. Ad inizio partita tutti i giocatori di tipo HumanPlayer selezioneranno il loro nome, successivamente, a turno, i giocatori (sia umani che pc) faranno le loro mosse sulla tabella.

Un giocatore umano può decidere di interrompere la partita salvandone lo

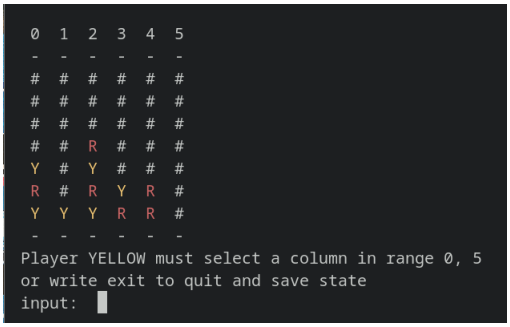


Figure 15: Di turno in turno viene richiesto al giocatore corrente di inserire una mossa (in caso di giocatore pc il procedimento è immediato)

stato, per farlo dovrà scrivere ”exit” invece che il numero di una colonna.

A partita vinta, o a tabella piena, il gioco viene interrotto e l'eventuale salvataggio viene cancellato, in quanto quella partita non è più giocabile.

```
 0  1  2  3  4  5
 -  -  -  -  -  -
 #  #  #  #  #  #
 #  #  #  #  #  #
 Y  #  #  #  #  #
 Y  #  #  #  #  #
 R  #  #  #  #  #
 Y  #  #  #  #  #
 Y  #  R  #  R  R
 -  -  -  -  -  -

Player YELLOW: Francesco, must select a column in range 0, 5
or type "exit" to quit and save state
input:  exit

play again? [y/n]
n
james@SpaceLink:~/eclipse-workspace/neoJConnect4/neoJConnect4/src/out(master⚡ ) »
```

Figure 16: Scrivendo "exit" il gioco viene salvato nella cartella Saved; viene chiesto all'utente se vuole giocare di nuovo.

```
 0  1  2  3  4  5
 -  -  -  -  -  -
 #  #  R  #  #  #
 R  #  Y  #  #  #
 Y  #  Y  #  #  #
 Y  R  R  Y  #  #
 Y  R  Y  Y  #  R
 R  Y  R  Y  #  R
 Y  Y  Y  R  R  R
 -  -  -  -  -  -
Player YELLOW: luigi won the game!

Savestate of game "ma" deleted!

play again? [y/n]
```

Figure 17: a partita finita il savestate è cancellato.

4 Risorse

Per la stesura del codice e la relazione del seguente documento ho utilizzato svariate risorse che mi sembra il caso di creditare, in ordine sparso, in chiusura.

- Visual Studio Code
  per la stesura del grosso del codice;
- Eclipse IDE
  per svariati check sul progetto e per i controlli di colori da terminale;
- GVim editor
  per la stesura del presente documento in LaTeX;
- Stack Overflow
  per i consigli sul testo colorato in console;
- Documentazione Oracle
  per la guida sulla serializzazione;
- Overleaf
  per i tutorial LaTeX.