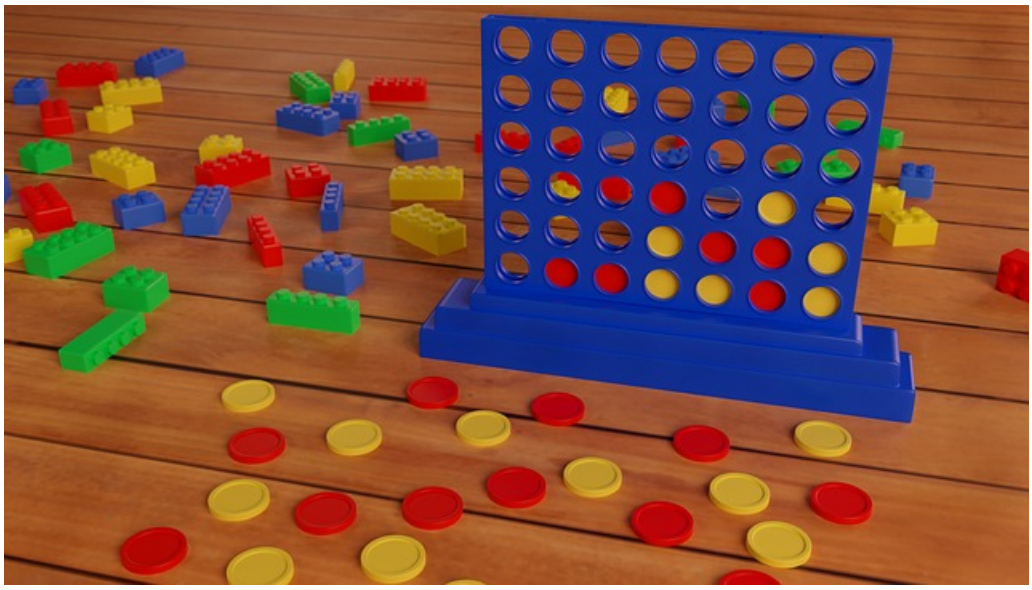
Forza4 Documentazione

Germano Anselmi (1994364), Mattia Di Marco (2019367)

13 Luglio 2022



Indice

1 INTRODUZIONE

Per il progetto di Metodologie di programmazione abbiamo scelto di lavorare al "Forza4", implementando anche la parte grafica.

Forza4 è un famoso gioco da tavolo a turni, su una griglia di 6 righe e 7 colonne, in cui due giocatori si sfidano cercando di unire 4 cerchi del proprio colore in linea retta (sia in orizzontale che in verticale che in obliquo).

Abbiamo diviso il progetto come segue:

- **Germano Anselmi** si è occupato della schermata di start con le relative classi e schermate associate;
- **Mattia Di Marco** si è occupato della schermata di gioco con le relative classi e schermate associate.

2 GUI

2.1 Schermata di start

La **schermata di start** prevede:

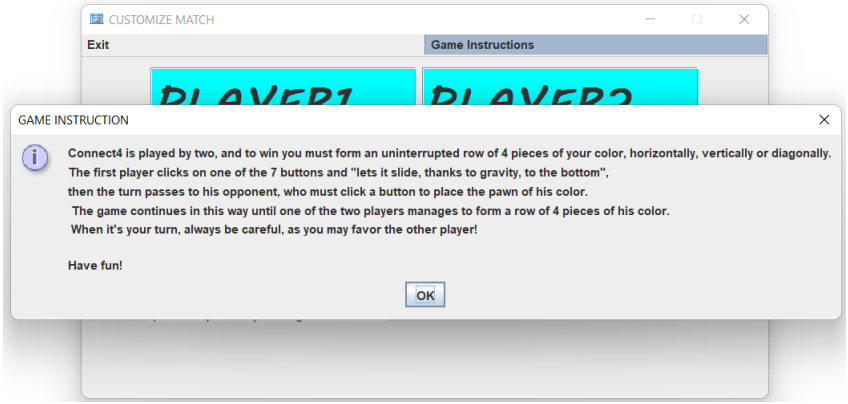
- Due bottoni colorati che rappresentano rispettivamente i colori delle pedine dei due giocatori;
- Un pannello di testo con 3 istruzioni da seguire prima di avviare la partita tramite il pulsante **Start**;
- Due aree di testo in cui inserire i nomi dei giocatori, di default verranno assegnati i nomi "Player1" e "Player2".

- Una volta inseriti i nomi basta cliccare sul pulsante **Save** per salvare le modifiche, mentre, per riprendere a giocare una partita salvata precedentemente, bisogna cliccare sul pulsante **Load**, il tasto andrà a cercare il file di salvataggio, se presente verrà avviata la partita salvata, altrimenti si aprirà una finestra di errore:

In alto è presente inoltre una MenuBar composta da due pulsanti:

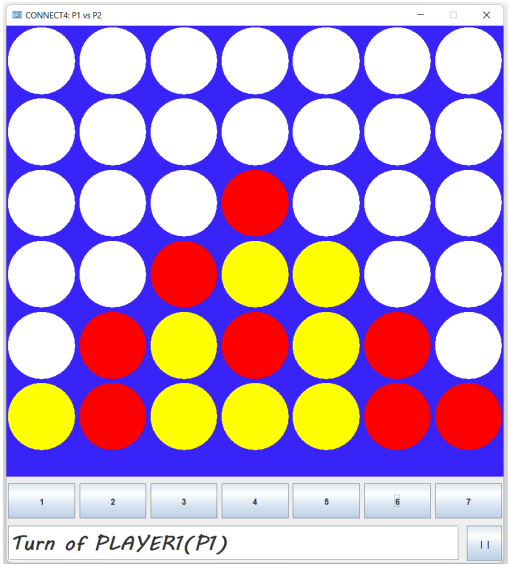
- **Exit**: cliccando sul pulsante si apre un'altra finestra in cui viene chiesto se si è sicuri di voler uscire:

- **Game Instructions**: cliccando sul pulsante si apre un'altra finestra in cui sono presenti le istruzioni di gioco:



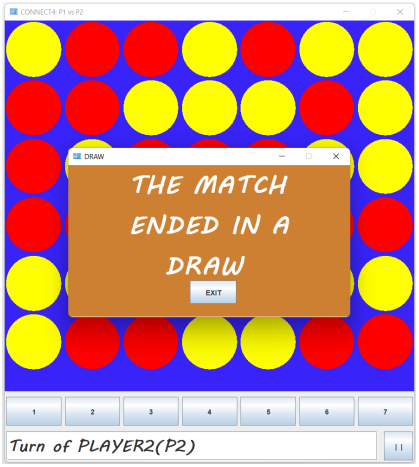
2.2 Schermata di gioco

La schermata di gioco si compone della parte grafica relativa alla griglia su cui si gioca. Alla fine di ogni colonna è presente un bottone, per un totale di sette, cliccandone uno dei quali, nel cerchio bianco più in basso nella colonna viene colorato del colore del giocatore che ha eseguito la mossa. Dopo svariate mosse tra i due giocatori la schermata sembrerà qualcosa di simile a questo:



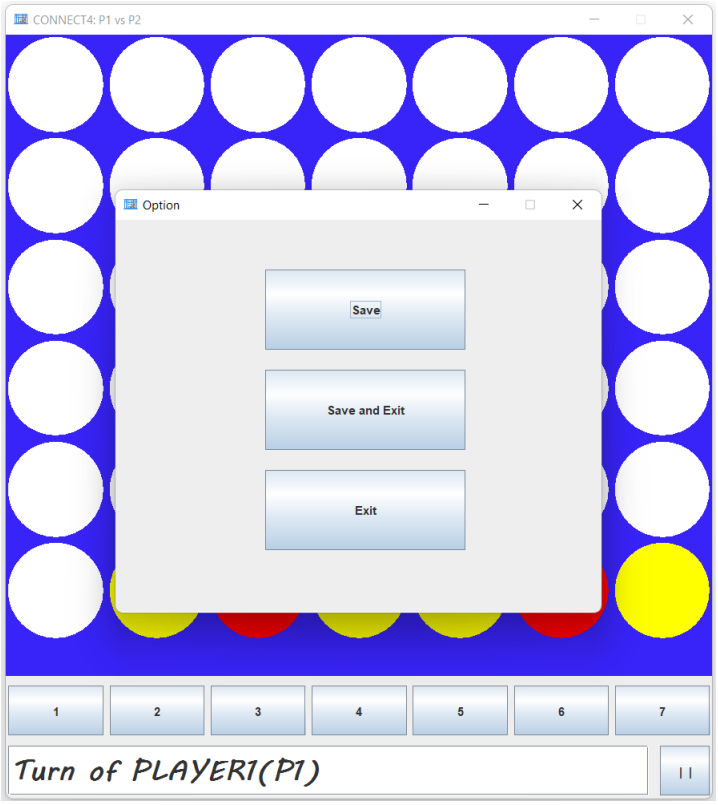
Nel caso di vittoria o pareggio vengono visualizzate due schermate separate, nel primo caso una con il nome scelto dal giocatore vincitore nell'altra semplicemente viene comunicato che la partita è finita in un pareggio.

(a) *Vittoria*



(b) *Pareggio*

È presente, inoltre, un TextArea dove viene mostrato di chi è il turno, o eventualmente, se si è cliccato su un bottone di una colonna già completamente riempita, un messaggio che segnala l'invalidità della mossa. Il bottone in basso a destra inoltre serve, a discrezione dei giocatori, a salvare la partita oppure uscire senza salvare.



In questa schermata sono presenti tre bottoni:

- il bottone in alto che salva la partita ma permette di continuare a giocarla;
- il bottone centrale che salva la partita e chiude il programma;
- l'ultimo bottone che invece chiude il programma senza salvare.

3 DESCRIZIONE DELLE CLASSI E DELLE FUNZIONALITÀ

Nel capitolo che segue andremo a presentare la descrizione delle classi, suddivise in front-end e back-end, e le relative funzioni, grazie all'aiuto del UML per rappresentare la gerarchia tra le stesse.

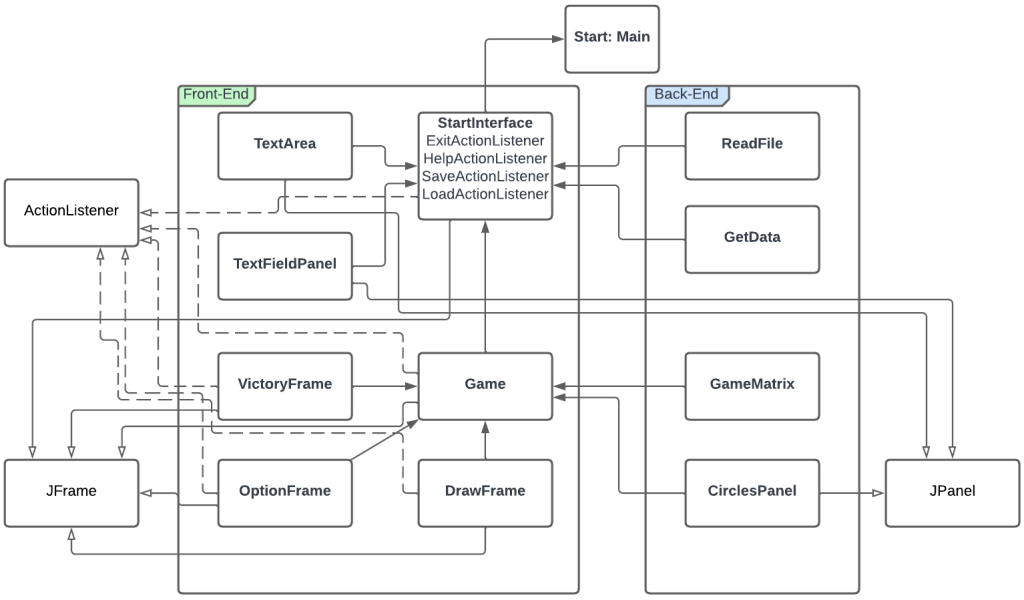


Figura 1: UML

3.0.1 Start.java

Semplicemente il main del progetto, quando viene runnato il codice questa classe chiama a sua volta la classe StartInterface.

3.1 Front-End

A seguito sono descritte le classi di front-end, ovvero quelle che si occupano della parte grafica del progetto.

3.1.1 StartInterface.java

La seguente classe si occupa della schermata iniziale del gioco, estendendo JFrame, e all'interno vengono istanziati vari *attributi* impostate con visibilità *private*:

- **matrix** di tipo int[][], contiene al suo interno la matrice iniziale di gioco che verrà sovrascritta in caso di salvataggio della partita;
- **defaultP1Name** di tipo String, contiene una stringa di default('PLAYER1'), in caso di mancato salvataggio o di mancato inserimento dei nomi, automaticamente, una volta cliccato su start, il gioco verrà avviato con il nome 'PLAYER1', per il giocatore1;
- **defaultP2Name** di tipo String, contiene una stringa di default('PLAYER2'), in caso di mancato salvataggio o di mancato inserimento dei nomi, automaticamente, una volta cliccato su start, il gioco verrà avviato con il nome 'PLAYER2', per il giocatore2;
- **player1Name** di tipo String, la variabile seguente contiene il testo iniziale del primo textFieldPanel;
- **player2Name** di tipo String, contiene il testo iniziale del secondo textFieldPanel;
- **turn** di tipo int, contiene un intero che rappresenta il turno del giocatore:
  - Nel caso del giocatore1 si ha turn = 1;
  - Nel caso del giocatore2 si ha turn = 2;
- **textFieldPanel** di tipo TextFieldPanel, rappresenta la schermata di testo in cui inserire i nomi dei due giocatori;
- **textArea** di tipo TextArea, rappresenta la schermata di testo in cui sono presenti le istruzioni su cosa fare prima di avviare la partita;
- **buttonP1** di tipo JButton, rappresenta il bottone che ha il colore del giocatore 1;
- **buttonP2** di tipo JButton, rappresenta il bottone che ha il colore del giocatore 2;

- **start** di tipo JButton, rappresenta il bottone che ha il compito di avviare la partita;
- **loadGame** di tipo JButton, rappresenta il bottone che ha il compito di ricaricare una partita precedentemente salvata;
- **save** di tipo JButton, rappresenta il bottone che ha il compito di salvare i nomi dei due giocatori;
- **menuBar** di tipo JMenuBar, rappresenta la barra dei menu in alto, all'interno contiene help e exit;
- **help** di tipo JMenuItem, rappresenta un bottone considerato come elemento del menuBar, se cliccato uscirà una schermata con le istruzioni del gioco;
- **exit** di tipo JMenuItem, rappresenta un bottone considerato come elemento del menuBar, se cliccato l'esecuzione del programma verrà terminata.

Successivamente viene creato il costruttore StartInterface(), all'interno vengono svolte varie operazioni per creare la vera e propria schermata iniziale del gioco.

Innanzitutto viene settato il titolo del Frame, in questo caso "CUSTOMIZE MATCH", alla riga successiva viene definito il metodo *setDefaultCloseOperation(DISPOSE_ON_CLOSE)* che permette, una volta cliccato sulla "x" in alto a destra, di chiudere la finestra corrente.

Viene creato il **menuBar**, tramite i metodi *setJMenuBar* che prende *createMenuBar()*, subito dopo viene settato il tipo di Layout del Frame, in questo caso *FlowLayout()*(crea un layout con allineamento centrato e uno spazio orizzontale e verticale predefinito di 5 unità).

A questo punto le variabili istanziate inizialmente vengono modificate nel costruttore creando il vero e proprio oggetto attraverso la parola chiave *new*:

- textFieldPanel = new TextFieldPanel();
- textArea = new TextArea();

Alla riga successiva viene settato il nome, il colore di background, e le dimensioni sia dei bottoni sia della pedina del Player1 e del Player2. Lo stesso viene fatto anche per i bottoni **save**, **start**, e **loadGame**, con la differenza che non viene settato il colore di background.

A questo punto ai bottoni **save**, **loadGame** e **start** viene aggiunto un ActionListener tramite il metodo *addActionListener()*.

Successivamente vengono create le varie classi "ascoltatrici", quest'ultime, implementando l'interfaccia ActionListener, vengono messe in attesa di un evento creato tramite il metodo *actionPerformed()*, le classi create sono le seguenti:

1. *ExitActionListener*: questa classe fa riferimento al pulsante di uscita dalla schermata, in caso venisse premuto il pulsante **exit** verrà aperta una schermata opzionale con il messaggio "Are you sure you want to quit?", se si clicca su "yes" la schermata verrò chiusa tramite il comando *System.exit()*;

2. *StartActionListener*: quando viene premuto il pulsante **start**, verrà aperta la vera e propria schermata del gioco tramite il comando *new Game(matrix, player1Name, player2Name, turn)*, tramite questo comando, viene creato un oggetto della classe **Game.java**;

3. *SaveActionListener*: nella seguente classe viene fatto in modo che, in caso venisse premuto il pulsante **save**, verranno salvati i nomi presenti all'interno dell'oggetto *textFieldPanel*, in caso di mancato inserimento dei nomi dei due giocatori, verranno considerati i nomi di default, ovvero "PLAYER1" e "PLAYER2";

4. *LoadGameActionListener*: la seguente classe si occupa del caricamento di una partita in caso fosse presente un salvataggio. All'interno dell'actionPerformed viene settata un array di stringhe ottenuto splittando per gli accapo la stringa ottenuta dalla lettura del contenuto del file **Connect4SaveFile.txt**, al cui internp sono presenti la matrice di gioco, i nomi dei due giocatori, e il turno da cui riprenderà la partita. Tramite il *try* e il *catch* e la classe **GetData** viene letto il file di salvataggio e viene riaperto il gioco chiamando la classe **Game** settata con parametri diversi, ad esempio se la matrice ha altri numeri oltre lo zero, nella schermata del gioco saranno presenti alcuni cerchi colorati, proprio a significare il fatto che è stata caricata una partita precedentemente salvata. In caso di mancato salvataggio, verrà lanciata un'eccezione all'interno del *catch*, ovvero verrà aperta una schermata di errore "No save file found".

3.1.2 TextArea.java

Questa classe, che estende JPanel, è la classe da cui deriva l'oggetto *textArea* presente all'interno della classe **StartInterface**. Al suo interno viene creata un'istanza del JTextArea chiamata **textArea**; Viene definito un costruttore al cui interno, attraverso il comando *new JTextArea()*,viene modificata la variabile precedentemente istanziata. JTextArea al suo interno prende un testo, in cui sono presenti le istruzioni di salvataggio:
"1) –Click on Player to change the player name"
"2) –Save your changes before starting"
"3) –Load a previously saved game."
Alla fine, tramite il metodo *add* viene aggiunta la textArea al JPanel.

3.1.3 TextFieldPanel.java

La seguente classe, che estende JPanel, si occupa della creazione dei due riquadri testuali per l'inserimento dei nomi dei giocatori.

Innanzitutto vengono create due istanze, *textField1* e *textField2*, di JTextField, alla riga successiva viene definito un costruttore al cui interno vengono creati i veri e propri riquadri di testo.

Per prima cosa vengono definite le due textArea assegnando un testo di default. Vengono poi settate le dimensioni, il colore di background e il font del testo tramite i metodi *setSize()*, *setBackground()*, *setFont()*.

Di seguito viene definito un Layout e viene settato come *FlowLayout()*. Inoltre vengono aggiunti i due textField(textField1 e textField2) al JPanel tramite il metodo *add()*.

Alla fine viene creato un metodo nominato *getNamePlayer()* per prendere il testo dai textField.

3.1.4 Game.java

Questa classe si occupa di mostrare la schermata in cui viene giocato Forza4, estendendo JFrame, e, attraverso l'implementazione dell'interfaccia ActionListener di poter effettivamente giocare. All'inizio vengono istanziati vari *attributi* necessari sia alla costruzione dell'interfaccia grafica, sia ai controlli relativi alla partita, tutti con modificatore di visibilità *private*:

- **matrix**: di tipo int[][], rappresenta la matrice su cui verranno salvate le mosse, controllate vittoria e pareggio ma anche da cui verrà, mano a mano, aggiornata la griglia disegnando i vari cerchi;
- **turn**: di tipo int, rappresenta il turno del giocatore corrente, all'inizio della partita viene settato ad 1, ma se la partita viene caricata dal salvataggio verrà, invece, caricato con il numero del giocatore che inizierà quella partita;
- **player1Name**: di tipo String, rappresenta il nome scelto dal giocatore 1;
- **player2Name**: di tipo String, rappresenta il nome scelto dal giocatore 2;
- **victory**: di tipo int, settato di base a 0, viene impostato ad 1 o 2, quando uno dei due giocatori raggiunge la vittoria;
- **draw**: di tipo int, settato a 0, viene impostato 1 quando viene raggiunto il pareggio;
- **validity**: di tipo boolean, settato di base a *true*, e serve come controllo per verificare che la mossa fatta del giocatore sia valida o meno, in caso in cui non lo sia viene settata a false;
- **b1,...,b7**: di tipo JButton, sono i sette bottoni che vengono posizionati uno sotto ogni colonna della griglia e gestiscono l'aggiunga in matrice della mossa, controllo vittoria e pareggio per quella specifica colonna;

- **option**: di tipo JButton, bottone che va a mostrare la schermata di opzioni, per salvare la partita oppure uscire senza salvare;
- **turns**: di tipo JTextArea, è l'area dove vengono mostrati i vari messaggi relativi alla partita, come il turno di uno dei due giocatori, oppure un messaggio che avvisa dell'impossibilità di eseguire una mossa;
- **gameMatrix**: oggetto di tipo GameMatrix, che attraverso i sui metodi va ad aggiungere la mossa nella matrice e a controllare vittoria e pareggio.

Successivamente viene istanziato il costruttore **Game(int[][] matrix, String player1Name, String player2Name, int turn)** che riceve in input questi oggetti in modo da poter, sia caricare una partita salvata sia una completamente nuova, e va ad eseguire le varie operazioni di creazione dell'interfaccia grafica.

Viene settato il nome del frame a "CONNECT4: P1 vs P2", con l'opzione *setDefaultCloseOperation(EXIT_ON_CLOSE)* viene fatto in modo che cliccando la "x" in alto a destra si esca dal programma chiudendolo, vengono impostate le dimensioni a 720×800px, viene tolta la possibilità all'utente di modificare le dimensioni del frame e fatto in modo che venga mostrato al centro del desktop, viene reso visibile, e infine, con il comando *setContentPane(new CirclesPanel(this.matrix))* viene chiamata la classe **CirclesPanel()** che prende la matrice e da questa disegna la griglia.

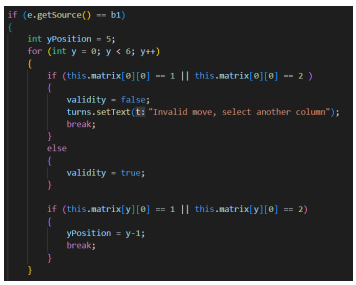
Vengono quindi aggiunti tutti i bottoni, grazie al metodo *setBounds()*, i sette necessari al gioco, sotto ogni colonna della griglia, e l'ottavo in basso a sinistra per le opzioni di salvataggio. Viene quindi aggiunto anche **turns** per mostrare l'andamento dei turni. Ogni bottone viene poi attivato con il comando *nomeBottone.addActionListener(this)* e il layout viene settato a null in modo da poter posizionare i bottoni liberamente.

Le azioni eseguibili sui bottoni sono di due tipi, una comune ai sette bottoni di gioco e l'altra per il bottone di salvataggio.

Bottoni di gioco.

Al momento del click di uno di questi bottoni vengono avviate una serie di calcoli per verificare la validità della mossa scelta e il salvataggio di quest'ultima, l'eventuale vittoria o pareggio, ma soprattutto viene disegnata la grigia con il cerchio appena aggiunto.

Viene inizializzato a priori un intero a 5, in quanto l'ultima riga di una matrice 6×7, e quindi la riga più bassa della griglia, si trova proprio all'indice 5. Con un ciclo **for** si scorrono quindi le righe della singola colonna(quest'ultima è fissa per ogni bottone, il primo è responsabile della colonna 0 della matrice, il secondo della 1 e così via), e si verifica da prima che nella colonna di riferimento sia possibile effettuare una mossa, e quindi che l'elemento nella riga di indice 0 non sia o 1 o 2



assegnando alla variabile **validity** il valore *true*, in caso contrario viene visualizzato nella TextArea il messaggio "Invalid move, select another column" e attraverso un break si esce dal **for**. Se invece la colonna non risulta piena, viene calcolato il primo spazio vuoto andando a trovare l'ultimo spazio pieno della colonna e sottraendo 1 al suo indice e con un break si esce dal **for**.

Se validity è settata a true allora, grazie all'uso dell'oggetto **gameMatrix**, si vanno a compiere alcune operazioni sulla matrice, la prima con il metodo *addMove(int[][][] matrix, int yPosition, int xPosition, int turn)* che aggiunge il numero del giocatore che ha effettuato il turno, 1 o 2, alla matrice, e ritorna la matrice così modificata. Con l'uso del metodo *controllo(int[][][] matrix, int turn)* si va a verifica l'eventuale vittoria, e con *draw(int[][][] matrix)* l'eventuale pareggio.

Dopo questi controlli viene creato un nuovo oggetto **CirclesPanel(int[][][] matrix)** che disegna la nuova matrice con la mossa appena effettuata e grazie al metodo *repaint()* viene ridisegnato il frame in modo che la griglia aggiornata sia visibile.

Vengono quindi fatti gli ultimi controlli, se non vi è stata la vittoria o il pareggio, viene scambiato il turno corrente in modo da andare avanti(se i turno è 1 diventa 2 e viceversa) e aggiornata **turns**, se si è arrivati o al pareggio o uno dei due giocatori ha vinto viene visualizzata la schermata corrispettiva.

Queste operazioni avvengono in maniera uguale al click di ogni bottone, l'unica differenza è la colonna su cui vanno ad operare i vari controlli di validità della mossa e aggiunta della stessa.

Bottone per le opzioni di salvataggio.

Al click di questo bottone viene visualizzata la schermata contenente i bottoni per il salvataggio.

3.1.5 OptionFrame.java

Questa classe si occupa della visualizzazione e della gestione dell'interfaccia necessaria al salvataggio della partita, estendendo la classe JFrame e implementando l'interfaccia ActionListener per i bottoni. Vengono da prima istanziati vari attributi con modificatore di visibilità *private*.

- **matrix**: di tipo int[][][], rappresenta la matrice che verrà salvata nel file per poter essere caricata in un secondo momento;
- **player1Name**: di tipo String, rappresenta il nome del player 1 che verrà salvato;
- **player2Name**: di tipo String, rappresenta il nome del player 2 che verrà salvato;
- **turn**: di tipo int, rappresenta il turno da salvare e quindi il giocatore che eseguirà la prima mossa quando verrà caricata la partita;
- **save**: di tipo JButton, è il bottone che se cliccato salverà la partita;

- **saveExit**: di tipo JButton, è il bottone che se cliccato salverà la partita e terminerà l'esecuzione del programma;
- **exit**: di tipo JButton, da la possibilità di terminare l'esecuzione del programma, e quindi di uscire dalla partita senza salvarla.

Troviamo poi il costruttore **OptionFrame(int[][] matrix, String player1Name, String player2Name, int turn)** che riceve tutte le informazioni relative alla partita da salvare nel file per poterla caricare in un secondo momento.

Vengono impostate la grandezza del frame a 500×430px e settato il *DISPOSE_ON_CLOSE*, in modo che, cliccando la "x" in alto a destra venga chiuso solo il frame e non tutto il programma. Similmente a **Game** vengono poi impostate il blocco della modifica delle grandezze del frame, la sua posizione nel desktop e ovviamente viene reso visibile.

A quel punto vengono aggiunti i tre bottoni, usando *setBounds()*, necessari alle varie operazioni, e vengono attivati in modo da poter eseguire il loro relativo codice al click di uno di essi.

Bottone Save

Cliccando questo bottone si va a salvare la partita allo stato corrente, permettendo però ai due giocatori di poter continuare la partita. La matrice, prima di essere salvata, viene però scritta sotto forma di stringa. Attraverso due **for** che scorrono la matrice ogni elemento viene aggiunto alla stringa, e ogni volta che una riga finisce, tranne se è l'ultima, viene aggiunto uno spazio per separala da quella successiva. A questo punto la matrice così formattata, i due nomi dei giocatori e il turno del giocatore vengono scritti all'interno di un file, di nome "Connect4SaveFile.txt", separati ognuno da un accapo, grazia ad un oggetto di classe FileWriter. A quel punto con *dispose()*, il frame di salvataggio viene chiuso.

Bottone Save and Exit

Il comportamento di questo bottone è del tutto simile a quello del bottone precedente, l'unica differenza è che, invece di dare la possibilità di continuare la partita, questa viene salvata e il programma viene chiuso con un *System.exit(0)*.

Bottone Exit

Questo bottone permette di chiudere il programma senza salvare, al click apparirà un JOptionFrame in cui viene avvisato l'utente che confermando la partita non verrà salvata ma l'esecuzione del programma verrà terminata.

3.1.6 VictoryFrame.java

In caso di vittoria di uno dei due giocatori, verrà visualizzato un apposito frame, gestito proprio da questa classe, che estende JFrame e implementa l'interfaccia ActionListener. Per prima cosa vengono istanziati gli *attributi*.

- **victory**: di tipo int, rappresenta numero del giocatore vincente;
- **player1Name**: di tipo String, il nome del giocatore 1;
- **player2Name**: di tipo String, il nome del giocatore 2;
- **exit**: di tipo JButton, in caso di click sul bottone seguente, verranno terminati tutti i processi del programma;
- **win**: di tipo JTextArea, rappresenta il testo di vittoria all'interno del frame;
- **gold**; di tipo Color, reppresenta il colore di sfondo del frame, con modificatore *final* in modo da non essere modificabile, è settato a 212,175,55 in RGB.

Successivamente viene definito un costruttore **VictoryFrame(int victory, String player1Name, String player2Name)**. Al suo interno vengono inizializzate le variabili definite precedentemente fuori dal costruttore.

Alle righe successive, sempre all'interno del costruttore, vengono settate la dimensione, il titolo, la posizione, il layout, il colore del background e la visibilità della finestra attraverso i metodi: *setSize()*, *setTitle()*, *setResizable()*, *setLocationRelativeTo()*, *setLayout()*, *setBackground()*, *setVisible()*.

Di seguito viene settata la posizione del pulsante di uscita dal programma tramite il metodo *setBounds()*, per poi inserire il pulsante nel pannello tramite il metodo *add()*. Ad **exit** viene anche assegnato un ActionListener in modo da svolgere il processo di chiusura di tutte le finestre tramite il metodo *actionPerformed()* che prende un evento *e* come argomento e, in caso di click sul pulsante **exit** termina il programma.

A questo punto vengono divisi in casi in cui a vincere sia il player1 o player2:

- **victory = 1**: verrà assegnata la vittoria al player1 e si aprirà la finestra con il testo "THE WINNER IS " più il nome da egli scelto;
- **victory = 2**: verrà assegnata la vittoria al player2 sempre con lo stesso testo ma con il nome scelto dal player2.

3.1.7 DrawFrame.java

In caso di pareggio, si aprirà un pannello con un testo che simboleggia che la partita è terminata con questo risultato, gestito da questa classe, che estende JFrame e implementa ActionListener.

Per prima cosa vengono istanziati vari *attributi*, in ordine:

- **exit**: di tipo JButton, in caso di click sul bottone seguente, verranno terminati tutti i processi del programma;
- **drawArea**: rappresenta il testo di pareggio all'interno del pannello;
- **bronze**: di tipo Color, rappresenta il colore di sfondo del frame, con modificatore *final*, è settato a 205,127,50 in RGB.

Successivamente viene definito un costruttore DrawFrame senza parametri. Al suo interno vengono inizializzate le variabili definite precedentemente fuori da quest'utltimo.

Alle righe successive,sempre all'interno del costruttore, vengono settate la dimensione, il titolo, la posizione , il layout ,il colore del background ,la visibilità della finestra e viene negata la possibiltà di ridimensionare la finestra, tutto ciò attraverso i metodi: *setSize()*, *setTitle()*, *setResizable()*, *setLocationRelativeTo()*, *setLayout()*, *setBackground()*, *setVisible()*.

Di seguito viene settata la posizione del pulsante di uscita allo stesso modo in cui è stato fatto per il pulsante **exit** nella classe VictoryFrame.

A questo punto, in caso di pareggio, si aprirà una finestra al cui interno è presente la **drawArea**, a quest'ultima viene inserito il seguente testo:
"THE MATCH ENDED IN A DRAW".

Vengono poi settate la posizione, il colore di background, la possibilità di editatibilità, il colore del testo(foreground) e il font della drawArea, per poi aggiungere quest'ultima all'interno della finestra tramite il metodo *add()*.

3.2 Back-End

In questa sezione ci concentreremo sulla descrizione della classi di back-end, che gestiscono varie operazioni tra cui il disegno della griglia e il conseguente aggiornamento ogni volta che viene eseguita una mossa, il controllo sulla matrice di gioco per vittoria o pareggio, l'apertura del file di salvataggio e l'estrazione dei dati contenuti in esso.

3.2.1 ReadFile.java

Questa classe è quella responsabile della lettura del file, in particolare attraverso il metodo readFileAsString(String fileName). Viene prima creata una stringa vuota e a questa viene aggiunta, leggendo byte per byte tutti i caratteri presenti nel file di salvataggio e ritorna questa stringa.

3.2.2 GetData.java

La classe serve per ottenere, dalla stringa ricavata da ReadFile le informazioni necessarie a caricare una partita salvata.
Attributi:

- **dataList** di tipo String[], rappresenta un array di stringhe ottenuto splittando, con *split("\n")*, per gli accapo la stringa letta dal file di salvataggio.

Metodi:

- int[][] *getMatrix()*:questo metodo va a ritornare la matrice salvata nel file di testo. Per fare questo viene prima creata una matrice vuota e da **dataList** viene preso l'elemento in indice 0, ovvero la stringa dove è salvata la matrice, che viene splittato rispetto agli spazi per avere una lista di stringhe che rappresentano le singole righe. Attraverso duo cicli **for** si scorre su questa lista per leggere i singoli caratteri che vengono convertiti da carattere a valore numerico e vengono inseriti nella matrice vuota nella posizione corrispondente;

- String *getPlayer1Name()*:il metodo inizializza una stringa il cui valore è l'elemento in indice 1 di **dataList** e la ritorna;

- String *getPlayer2Name()*:il comportamento è lo stesso del metodo precedente solo che in questo caso viene preso l'elemento di indice 2;

- int *getTurn()*: in questo caso viene preso l'elemento di dataList all'indice 3, che viene convertito in intero attraverso il metodo *parseInt()* e ritornato, in questo modo è possibile conoscere chi inizierà la partita una volta che sarà ricaricata.

3.2.3 CirclesPanel.java

La seguente classe si occupa del disegno della griglia, estendendo JPanel, sia che questa appartenga ad una nuova partita, sia che venga eseguita una mossa, sia che venga caricata una nuova partita.
Attributi:

- **matrix** di tipo int[][], rappresenta la matrice che verrà usata come base per disegnare la griglia di gioco.

Metodi:

- *drawCircles(Graphics g, int centroX, int centroY, int raggio)*:di tipo void e con modificatore di visibilità *private*, questo metodo, usando un oggetto di classe Graphics *g* e usando i due metodi dell'oggetto *drawOval()* e *fillOval()* va disegnare, e riempire del colore a cui g, un cerchio;

- *paintComponent(Graphics g)*: in questo caso, andando ad attuare l'overriding del metodo *paintComponent()* appartenente a JPanel, ovvero la scrittura, all'interno di una sottoclasse, di un metodo che presenta lo stesso nome di uno della superclasse, per renderlo più specifico, siamo in grado di disegnare la griglia. Viene prima creato un oggetto di classe Color con il colore dello sfondo della griglia, il disegnatore g viene settato a quel colore, e con i metodi *drawRect()* e *fillRect()* viene disegnato e riempito lo sfondo. A quel punto, iterando con due for su **matrix** si vanno a prendere i singoli valori, se il valore risulta 1 viene settato il colore di g a giallo e chiamata *drawCircles()*, se il valore è 2 viene settato a rosso e chiamato *drawCircles()*, infine se nessuno dei due, e quindi 0, viene settato a bianco e chiamato *drawCircles()*.

3.2.4 GameMatrix.java

Infine, questa classe, attraverso i suoi metodi va ad aggiungere le mosse eseguite da ogni giocatore nella matrice e controlla vittoria o pareggio. *Metodi*:

- **addMove(int[][] matrix, int yPosition, int xPosition, int turn)**: questo metodo prende la matrice, l'indice y e x in cui aggiungere la mossa e l'intero che rappresenta il giocatore che l'ha appena eseguita restituendo la matrice così modificata;

- **controllo(int[][] matrix, int turn)**: il compito di questo metodo è quello di effettuare il controllo per la vittoria, che nel forza4 avviene quando 4 cerchi, e quindi 4 numeri nella matrice risultano uguali in linea retta. Viene prima di tutto inizializzata una variabile di tipo int victory che viene restituita a 0 se nessuno ha vinto, 1 o 2 altrimenti, dipendentemente dal vincitore. Per il controllo in se si scorrono con due for prima le righe della matrice e poi le colonne, a questo punto per la vittoria in orizzontale basta controllare le sequenze di 4 numeri che hanno come primo valore un numero in indice colonna compreso tra 0 e 3. Per la vittoria in verticale

si controllano invece le sequenze che iniziano con numeri di indice riga compreso tra 0 e 2. Per le sequenze orizzontali invece, si controllano quelle che partono da indice riga compreso tra 0 e 2 e indice colonna tra 0 e 3, oppure per riga compreso tra 0 e 2 e colonna tra 3 e 6. Ogni volta che uno di questi controlli risulta positivo la variabile victory viene settata uguale a turn in modo da tornare il numero del giocatore vincente;

- **draw(int[][] matrix)**: quest'ultimo metodo serve per controllare il pareggio, questo nel forza4 è possibile solo nel caso in cui la griglia, e quindi la matrice, siano completamente riempite senza che nessuno dei due giocatori abbia raggiunto la vittoria. Il metodo va quindi effettivamente a controllare che solo nella riga della matrice ad indice 0, e quindi quella più in alto, vi siano solo 1 o 2, e in caso affermativo ritorna 1, 0 altrimenti. Il metodo non controlla effettivamente che vi sia un pareggio, potrebbe esserci il caso in cui uno dei due giocatori vinca e la matrice risulti comunque piena, ma visto che in Game.java il controllo sulla vittoria, e la conseguente apertura della schermata di vittoria, precedono quello sul pareggio viene dato peso maggiore alla vittoria non creando problemi.

```
public int controllo(int[][] matrix, int turn){
    int victory = 0;

    for (int y = 0; y < 6; y++){
        for (int x = 0; x < 7; x++){
            if (x < 4){
                if (matrix[y][x] == turn & matrix[y][x+1] == turn & matrix[y][x+2] == turn & matrix[y][x+3] == turn){
                    victory = turn;
                    break;
                }
            }
            if (y < 3){
                if (matrix[y][x] == turn & matrix[y+1][x] == turn & matrix[y+2][x] == turn & matrix[y+3][x] == turn){
                    victory =  turn;
                    break;
                }
            }
            if (x < 4 & y < 3){
                if (matrix[y][x] == turn & matrix[y+1][x+1] == turn & matrix[y+2][x+2] == turn & matrix[y+3][x+3] == turn){
                    victory = turn;
                    break;
                }
            }
            if (x >= 3 & y < 3){
                if (matrix[y][x] == turn & matrix[y+1][x-1] == turn & matrix[y+2][x-2] == turn & matrix[y+3][x-3] == turn)
                {
                    victory = turn;
                    break;
                }
            }
        }
    }
    return victory;
}
```

(a) *Codice controllo vittoria*

```
public int draw(int[][] matrix)
{
    int draw = 0;

    for (int x = 0; x < 7; x++)
    {
        if(matrix[0][x] == 0)
        {
            draw = 0;
            break;
        }
        else
        {
            draw = 1;
        }
    }
    return draw;
}
```

(b) *Codice controllo pareggio*

4 REFERENTI DI SVILUPPO

Per quanto riguarda la divisione dei compiti, come già accennato nell'introduzione, abbiamo deciso che chi avesse scritto il codice di una determinata classe si sarebbe occupato di conseguenza anche della sua descrizione nella documentazione, in particolare:

- **Germano Anselmio** si è occupato della **Schermata di start** e quindi delle classi **StartInterface**, **TextArea**, **TextFieldPanel** e anche delle classi **VictoryFrame** e **DrawFrame**;
- **Mattia Di Marco** si è occupato della **Schermata di gioco** con le classi **Game**, **OptionFrame**, **CirclesPanel**, **GameMatrix** e anche di **ReadFile** e **GetData**.

La scrittura della documentazione è avvenuta insieme grazie all'uso di OverLeaf, mentre per condividere il codice abbiamo usato GitHub (a cui lasciamo il collegamento, cliccando qui).