

# Othello

**Presentato da:**

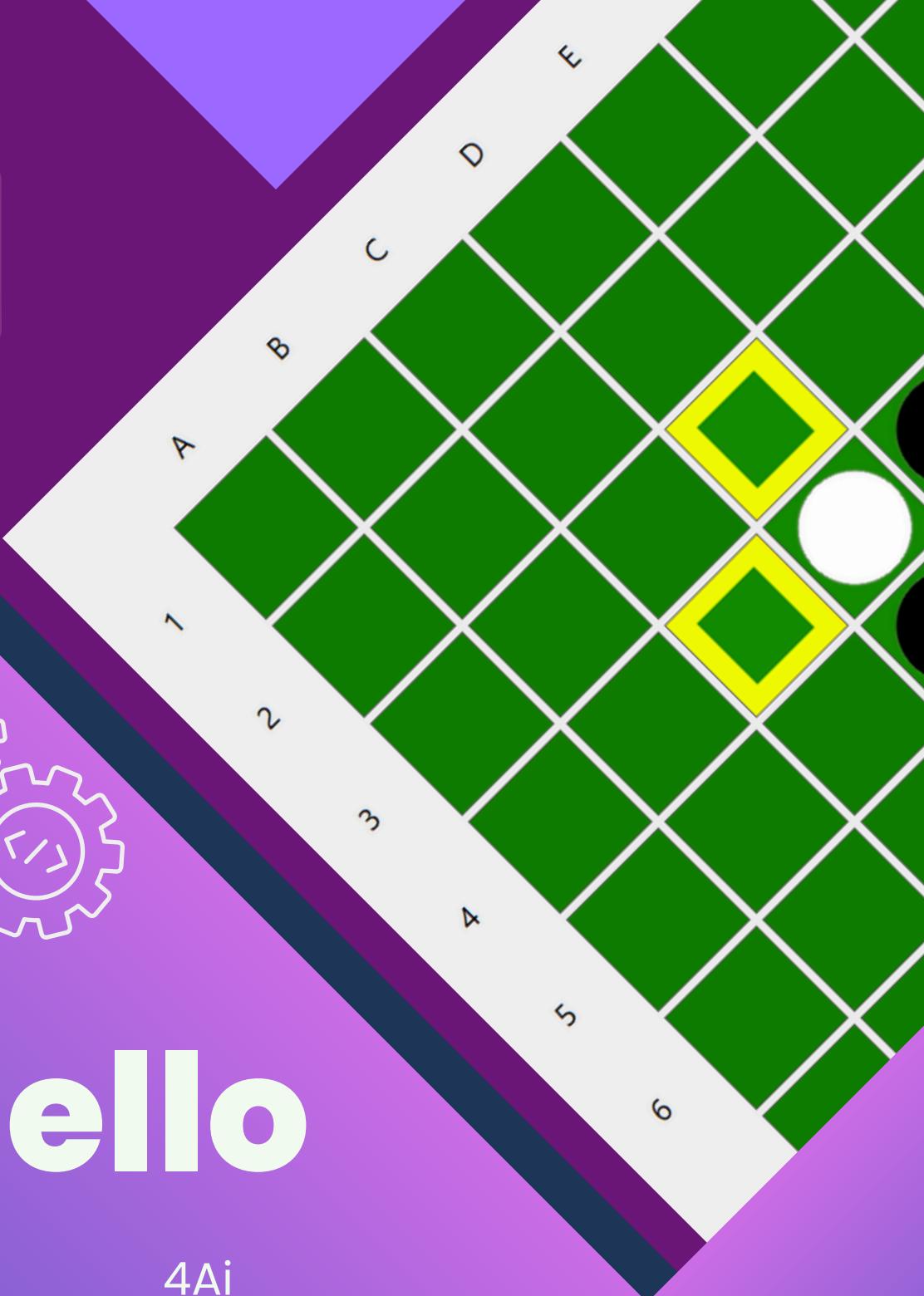
Duhanxhiu Hegi

Mascioli Luigi

Travellini Gabriele

4Ai

A.S. 2024/2025



# Indice Unità

1

**Il problema**

2

**Overview della soluzione**

3

**Test eseguiti**

4

**Il nostro team**

5

**Manuale utente**



# Il problema

- La sfida proposta è quella di sviluppare un programma in C# in grado di simulare una partita del gioco da tavolo Othello.
- L'applicazione deve permettere l'interazione tra due giocatori (umano vs umano o umano vs bot), un'interfaccia grafica e deve rispettare le regole ufficiali del gioco, visionabili al seguente link:  
<https://www.fngo.it/regole.asp>



# Overview della soluzione

## Editor e strumenti utilizzati

- Visual Studio (Framework .NET 8.0)

Per la realizzazione del codice



- GitHub

Per la cooperazione e lo sviluppo del progetto

- Photoshop

Per la realizzazione delle grafiche



- Canva

Per la realizzazione della documentazione

- Microsoft Whiteboard

Per la fase di progettazione e brain storming

# Overview della soluzione

## Linguaggio utilizzato



- C#

## Documenti consultati

- Documentazione fornita dal docente Amato Alessio
- Documentazione della “Federazione Nazionale Gioco Othello”
- Documentazione ufficiale Microsoft C#

## Librerie utilizzate

- Windows Forms
- System IO

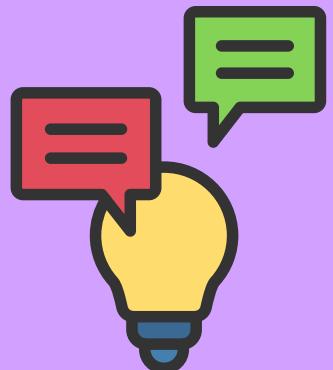


# Overview della soluzione

## strutture dati

### Tabellone: matrice 2D (int[,])

- Facilità di gestione
- Chiarezza nell'utilizzo delle coordinate

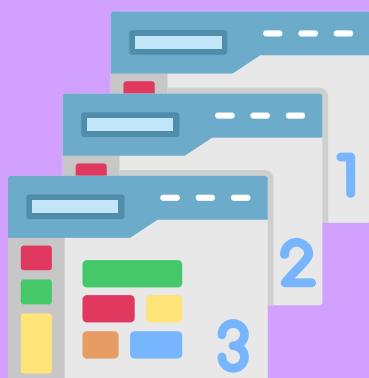


### Movimenti: array monodimensionali (int[] dx e int[] dy)

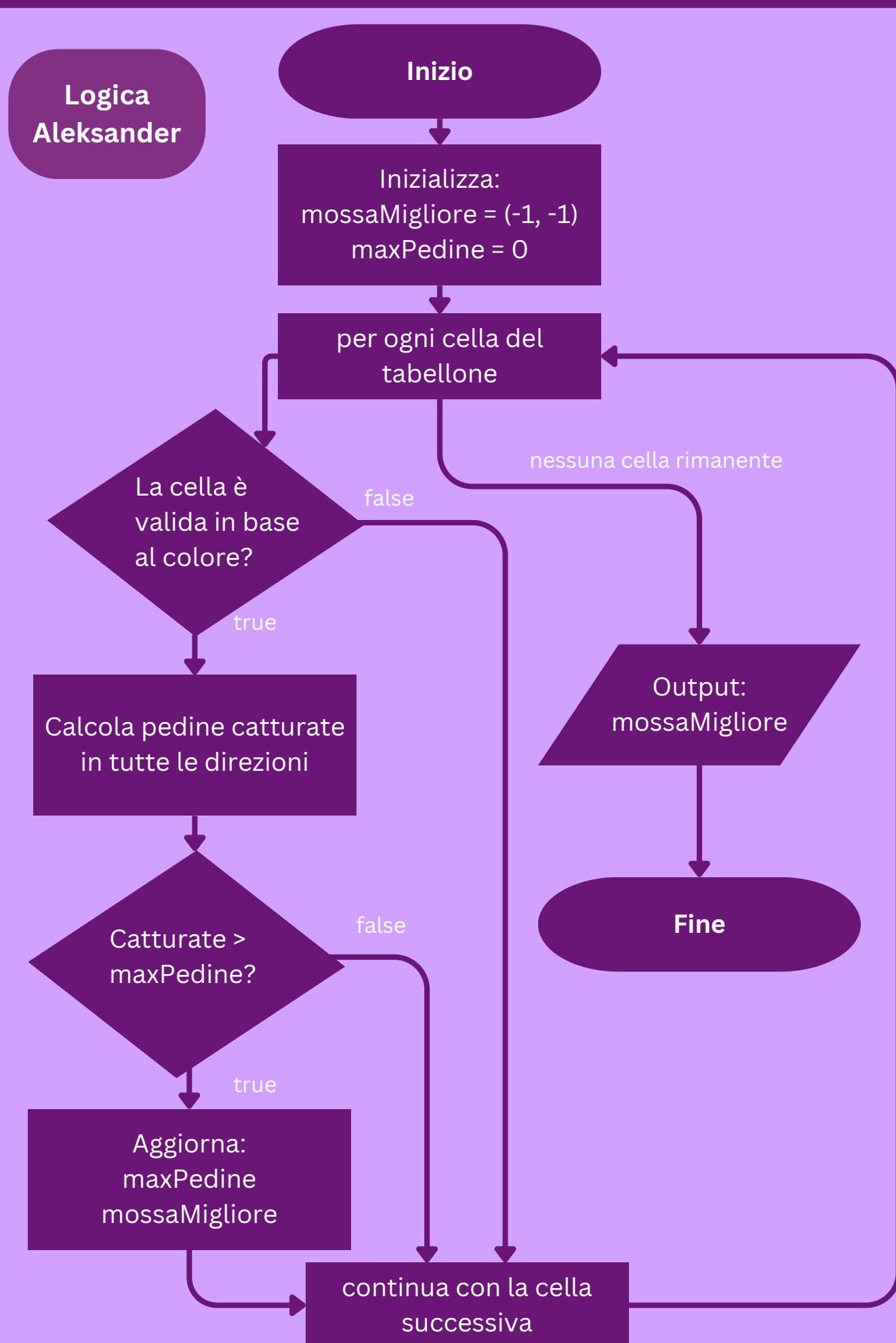
- Compattezza
- Semplicità di utilizzo

### Interazione: dizionario (ChiaviPictureBox, Valori: Tuple<int, int>)

- Semplice correlazione tra GUI e codice
- Chiarezza e intuitività

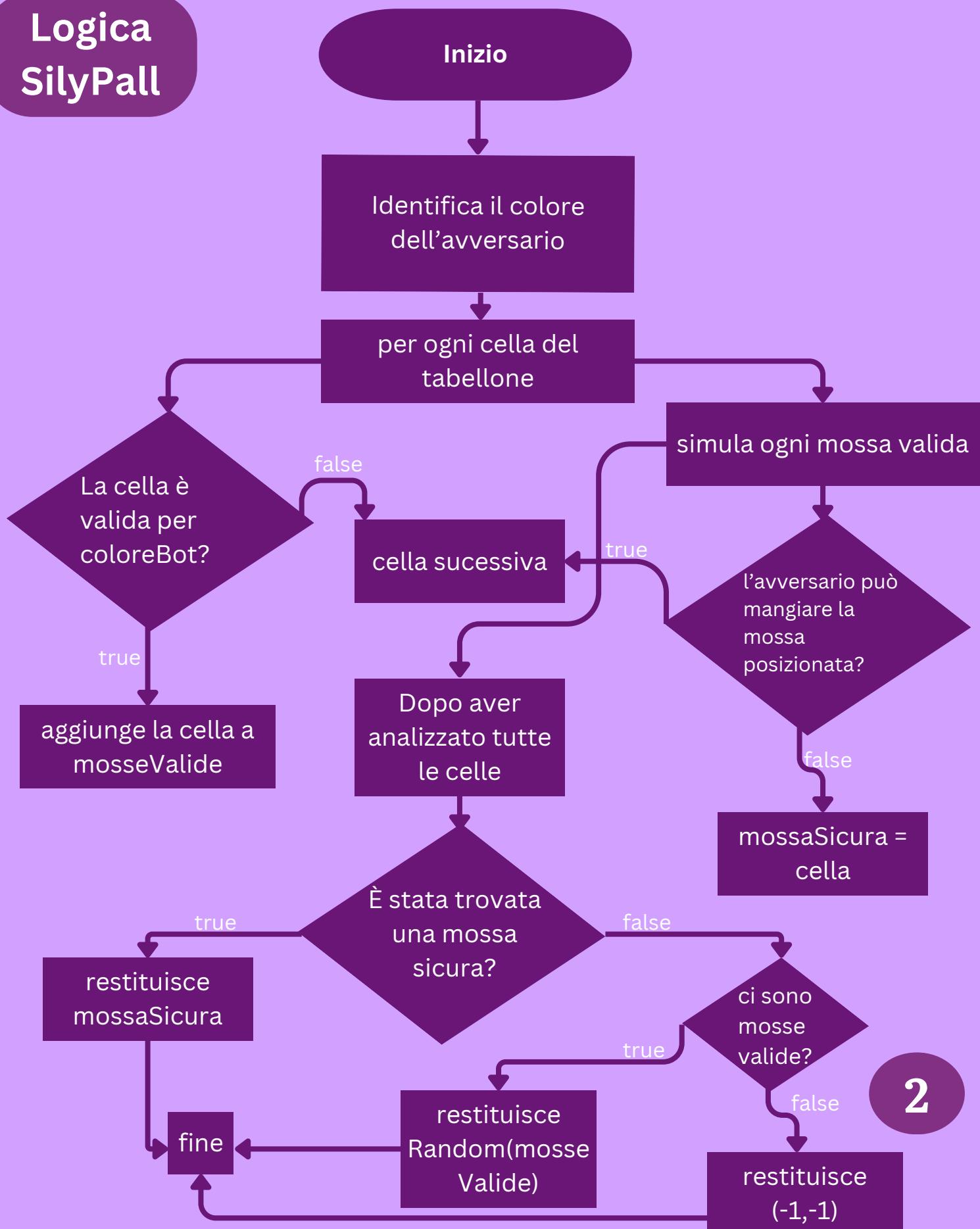


# Overview della soluzione



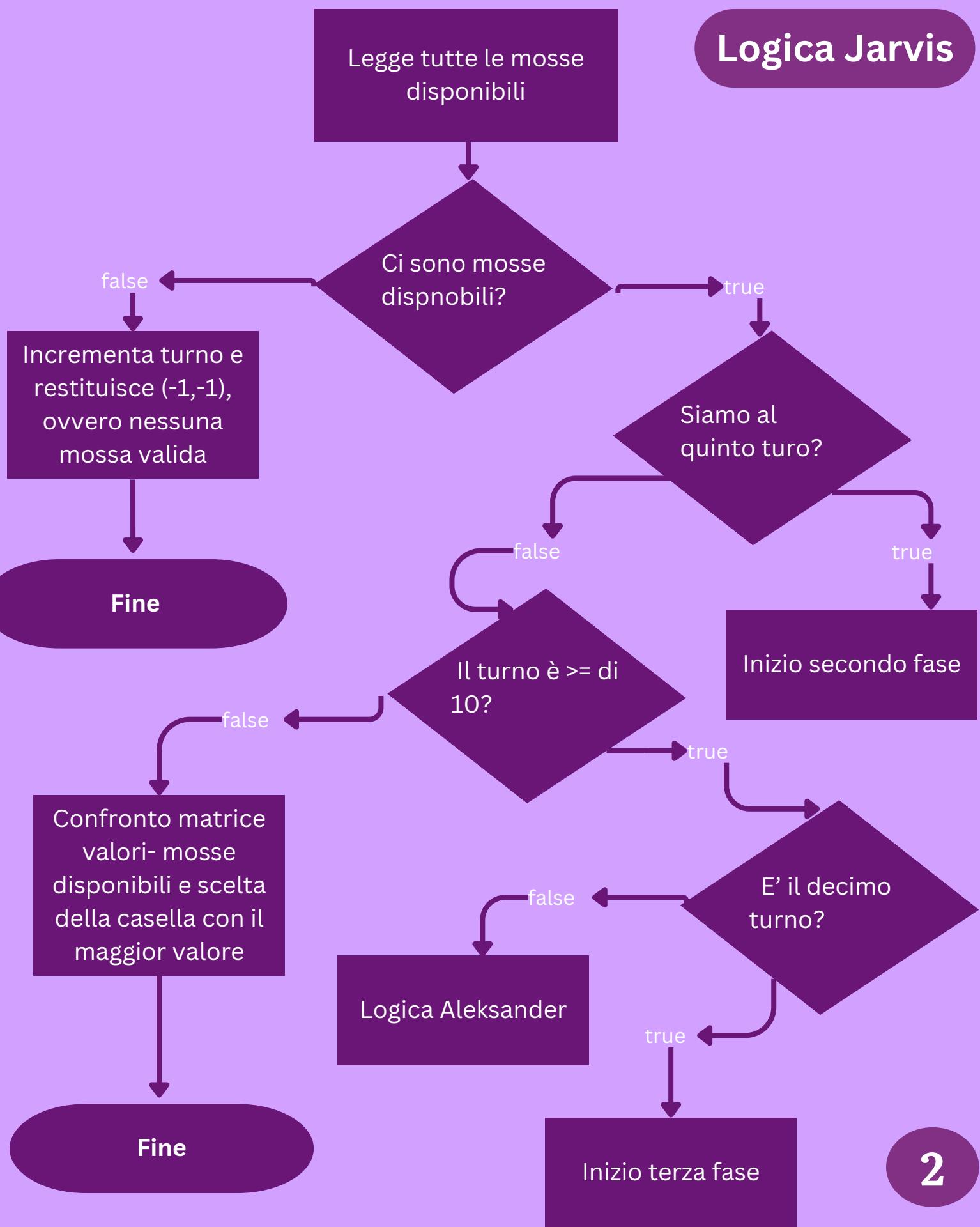
# Overview della soluzione

## Logica SilyPall



# Overview della soluzione

Logica Jarvis



# Logica delle fasi e matrice con valori – Jarvis (1)

## Valori di riferimento

L'inventore di Othello, Goro Hasegawa, ha contrassegnato con lettere particolari le seguenti caselle:

- Le Caselle-B sono al centro del bordo
- Le Caselle-C sono sul bordo più vicino all'angolo
- Le Caselle-A si trovano fra le due appena citate.
- Le Caselle-X sono diagonalmente vicino agli angoli, con X che indica pericolo.

L'unico modo di conquistare l'angolo, che per un gioco come Othello è la posizione migliore visto che non può essere ribaltato, è che l'avversario giochi sulle caselle C o X. Da qui costruiamo la tabella delle priorità.

	a	b	c	d	e	f	g	h
1		C	A	B	B	A	C	
2	C	X					X	C
3	A							A
4	B							B
5	B							B
6	A							A
7	C	X					X	C
8		C	A	B	B	A	C	

```
private int[,] matricePriorita = { //valori default matrice (https://www)  
    { 100, -10, 10, 10, 10, 10, -10, 100 },  
    { -10, -50, 0, 0, 0, 0, -50, -10 },  
    { 10, 0, 5, 5, 5, 5, 0, 10 },  
    { 10, 0, 5, 5, 5, 5, 0, 10 },  
    { 10, 0, 5, 5, 5, 5, 0, 10 },  
    { 10, 0, 5, 5, 5, 5, 0, 10 },  
    { -10, -50, 0, 0, 0, 0, -50, -10 },  
    { 100, -10, 10, 10, 10, 10, -10, 100 }  
};
```

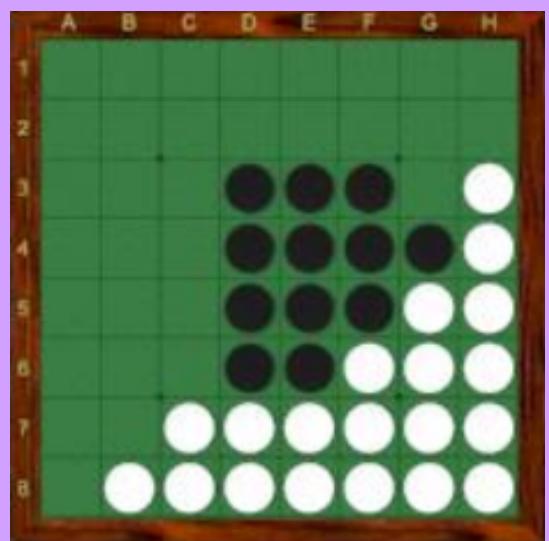
# Logica delle fasi e matrice con valori – Jarvis (2)

## Prima fase (fino a turno 4)

Mantenere i valori di default della matrice e cercare di ottenere almeno un angolo

## Seconda fase (da turno 5 a turno 9)

Jarvis cerca di costruire il suo gioco sulle caselle ai lati per creare un blocco di pedine inattaccabile (vedi figura accanto)



## Terza fase (da turno 10 fino alla fine)

Jarvis passa la palla ad Aleksander, che giocherà aggressivo nella fase finale per assicurarsi una vittoria in volata!



# Diagramma delle classi - Tabellone

## Tabellone

---

- colorePedina: int
  - tabelloneMatrice: Array bidimensione int
- 

- + Tabellone(): void
- + pulisci(): void
- + copiaDa(altro: Tabellone): void
- + getTabellone(): Array bidimensionale int
- + posiziona(colorePedina: int, posizioneX: int, posizioneY: int): bool
- + verificaMossa (colorePedina: int, posizioneX: int, posizioneY: int): bool
- + verificaVittoria(): int
- + getMosseDisponibili (colorePedina: int): Lista (int,int)

# Diagramma delle classi - Aleksander

## Aleksander

---

- + effettuaMossa(tabellone: Tabellone,  
colorePedina: int): (int,int)
  
- calcolaCatture(statoTabellone: int[],  
colorePedina: int, x; int, y: int): int

# Diagramma delle classi - Silypall

## Silypall

---

- random: Random

---

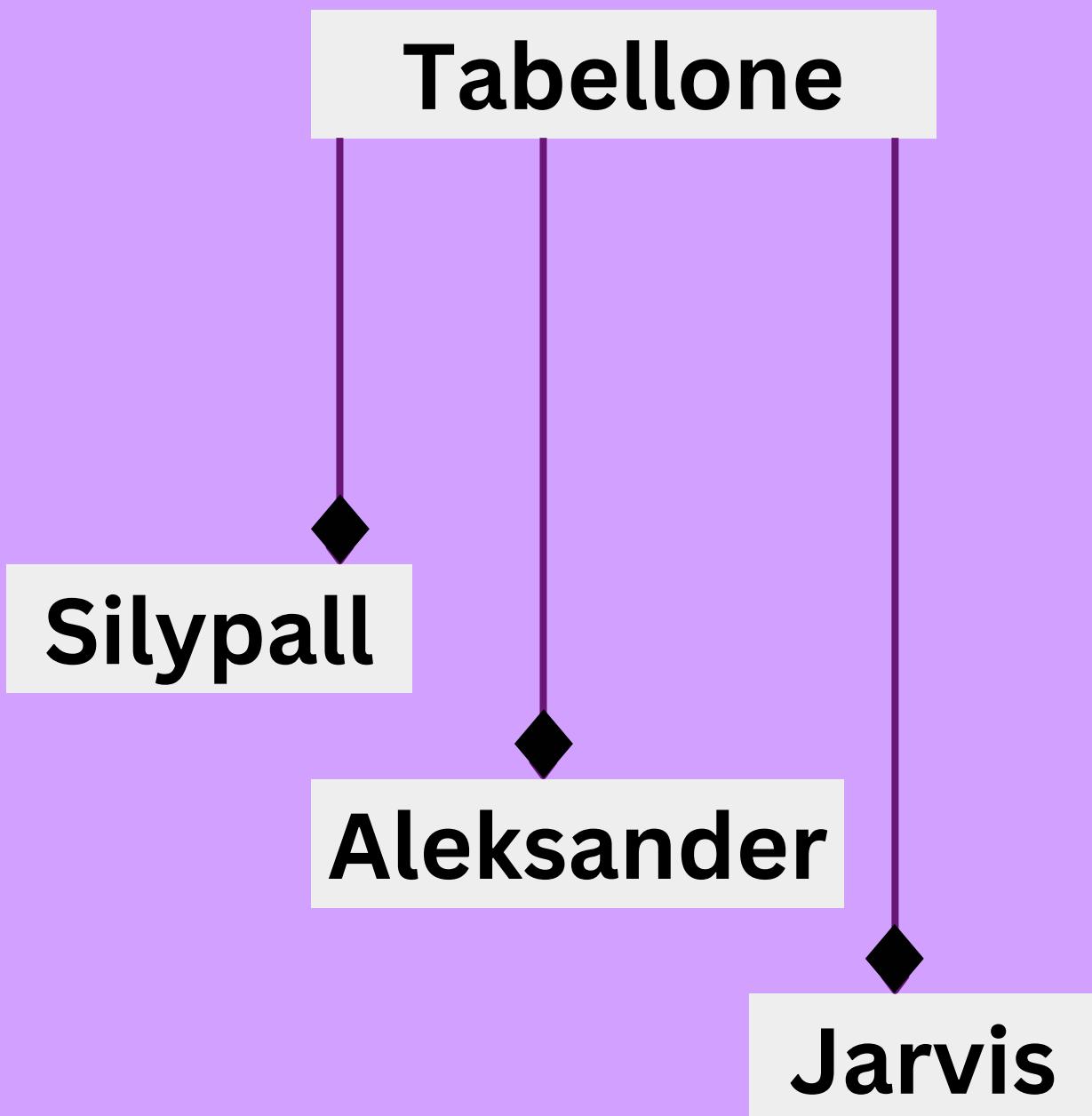
+ mossa(t: Tabellone, coloreBot: int): (int,int)

# Diagramma delle classi - Jarvis

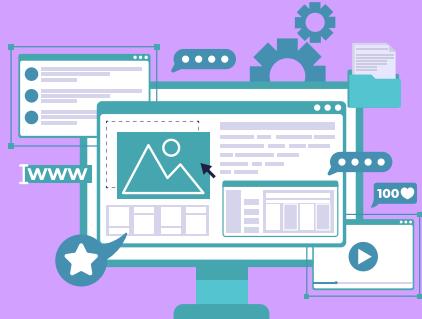
## Jarvis

- colorePedina: int
  - tabelloneMatrice: Tabellone
  - mosseDisponibili: Lista (int,int)
  - contaTurno: int
  - botAggressivo: Bot1
  - matricePriorita: Array bidimensionale int
  - random: Random
- 
- + Jarvis(colorePedina: int, tabellone: Tabellone): void
  - mosseValide()
  - secondaFase(): void
  - terzaFase(): void
  - + effettuaMosssa(): (int,int)

# Diagramma delle classi - composizione (has-a)



# Test eseguiti

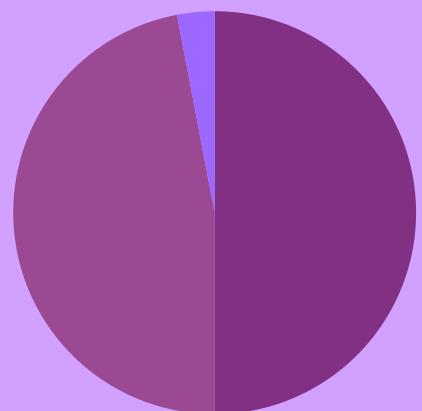


Aleksander  
47%

pareggi

3%

Sillypall  
50%



**Abbiamo fatto eseguire  
1000 Bot contro Bot a  
rotazione per valutarne  
i risultati:**

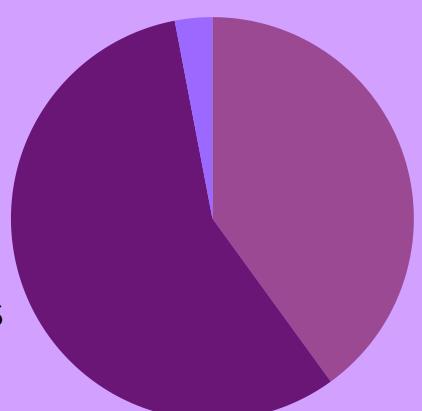


Jarvis  
57%

pareggi

3%

Aleksander  
40%

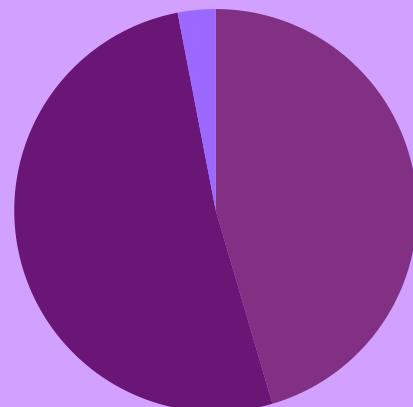


Jarvis  
51.5%

pareggi

3%

Sillypall  
45.5%



# Il nostro team

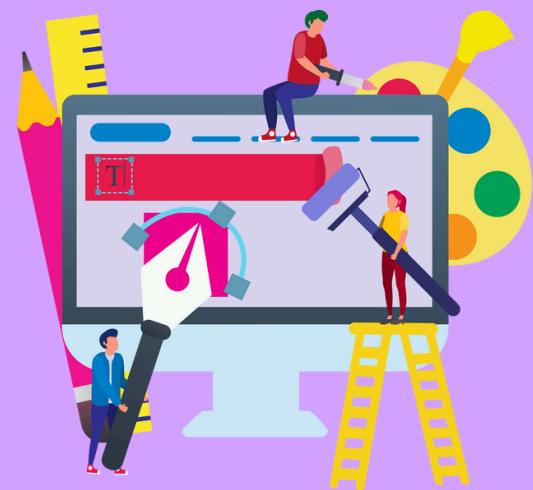
Duhanxhiu:

- Classe Tabellone
- GUI
- Bot Aleksander
- Utente VS Utente
- Grafiche
- Documentazione



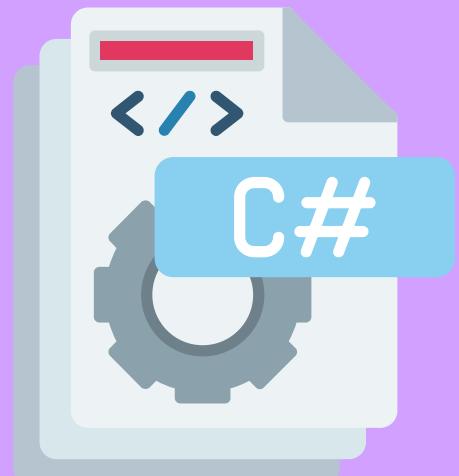
Travellini:

- Classe Tabellone
- GUI
- Bot Sillypall
- Gestione file
- Replay

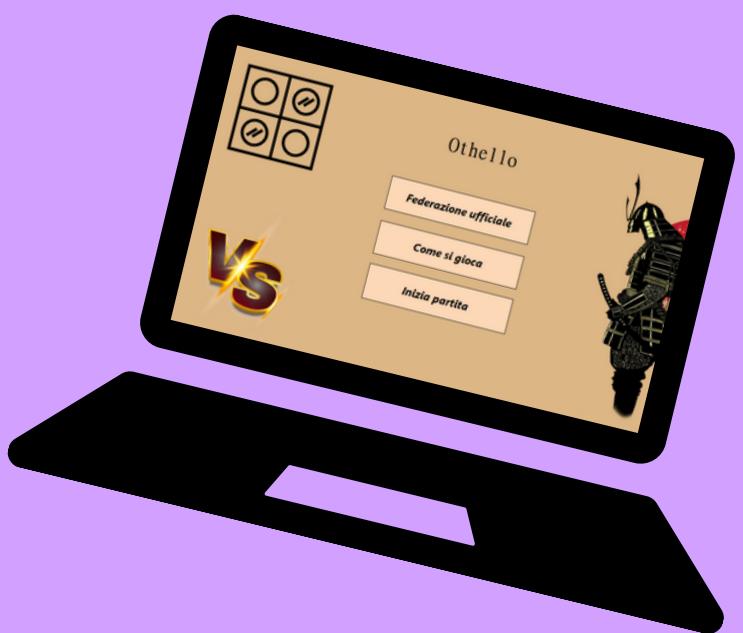


Mascioli:

- Classe Tabellone  
(NO posiziona)
- GUI - schermata iniziale e passo turno
- Bot Jarvis
- Test statistici
- Documentazione



# Manuale utente



# Schermata iniziale

Il sito della FNGO offre un tutorial sulle regole di Othello



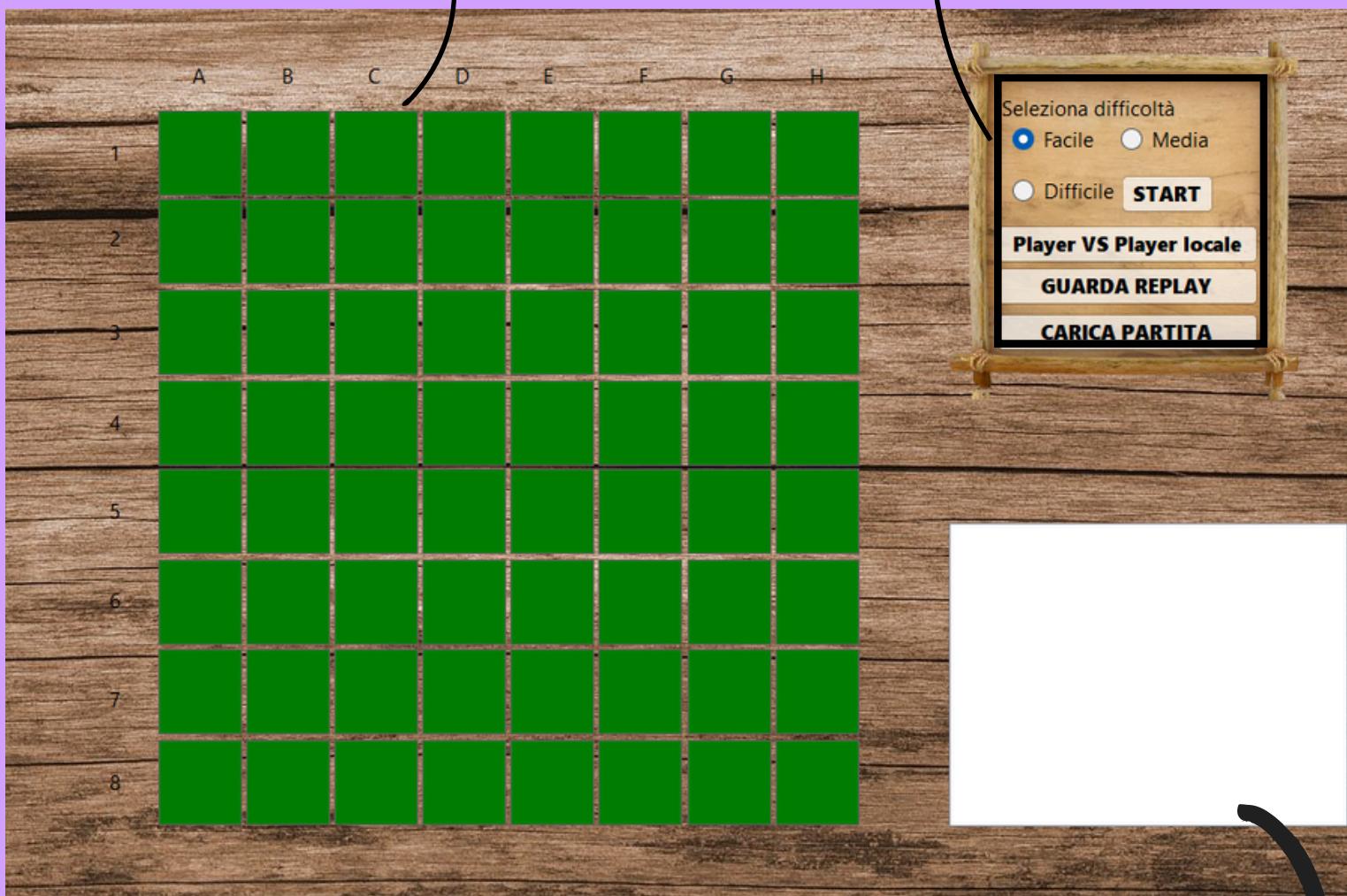
Inizia a  
giocare!

# Schermata di avvio

Scacchiera



Menù di scelta



Listbox in cui vengono mostrare le mosse giocate



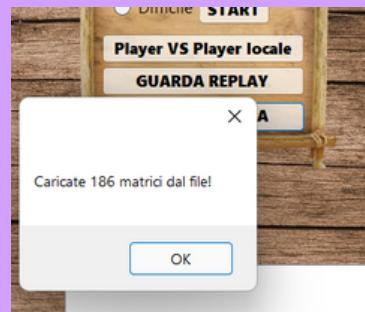


Aleksander - facile  
Silypall - media  
Jarvis - difficile



La difficoltà si basa sui test effettuati tra i bot, ma dipende molto dall'andamento della partita e stile di gioco.

**START** per avviare la partita con il bot



Cliccando su **CARICA PARTITA** è possibile, una volta finita la partita, caricare in memoria la partita appena giocata



La funzionalità del replay (**GUARDA REPLAY**) è possibile se l'utente chiude e riapre l'applicazione caricando successivamente la partita. Il replay verrà interrotto in caso di avvio di partita

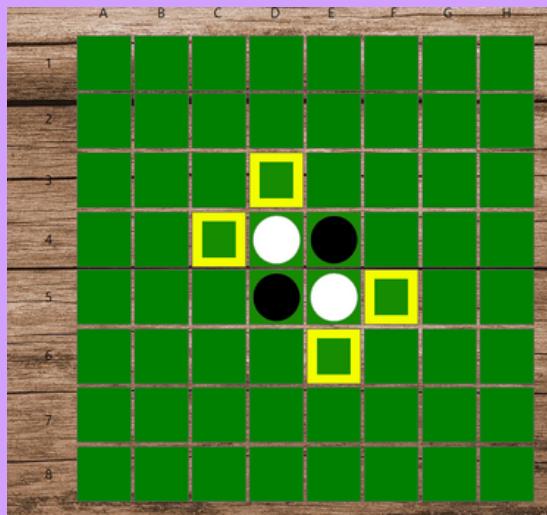
**Player VS Player locale** per giocare 1v1 in locale



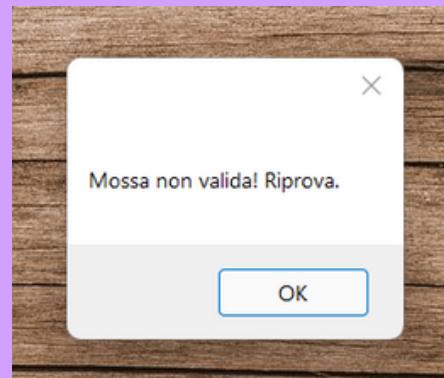
Il progetto contiene “Cartella file” dove all'interno c'è un `gameData.txt`. Lì è salvato il replay, quindi si prega di **NON MODIFICARE** in alcun modo la cartella.

# Inizio e continuazione della partita

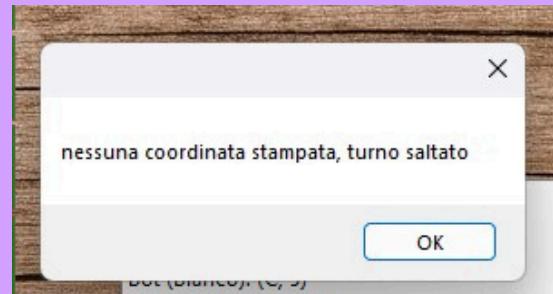
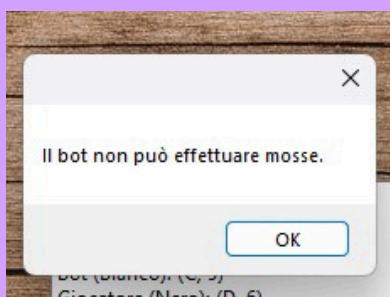
Una volta avviato il programma, inizierà sempre il nero a muovere



Indica le caselle dove è possibile posizionare una pedina. Nel caso di posizionamento non in queste caselle, apparirà un messaggio di errore (fig. sotto) e si dovrà ripetere la mossa



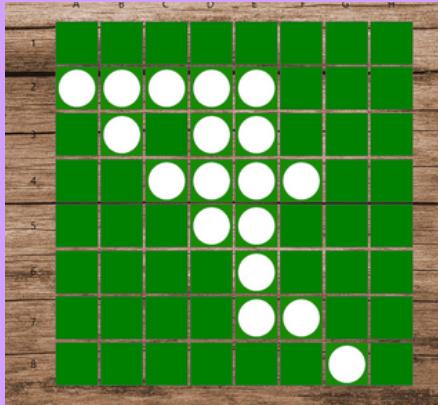
Il programma individua quando non ci sono più mosse disponibili. Il turno viene passato al giocatore successivo, in automatico, mostrando un messaggio di segnalazione



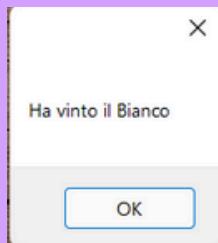
Tramite la listbox in basso a destra, è possibile vedere le mosse effettuate dai giocatori.

Giocatore (Nero): (E, 6)  
Bot (Bianco): (F, 4)  
Giocatore (Nero): (E, 3)  
Bot (Bianco): (D, 6)  
Giocatore (Nero): (C, 5)  
Bot (Bianco): (C, 4)  
Giocatore (Nero): (B, 3)  
Bot (Bianco): (F, 6)  
Giocatore (Nero): (D, 7)  
Bot (Bianco): (C, 8)  
Giocatore (Nero): (D, 8)

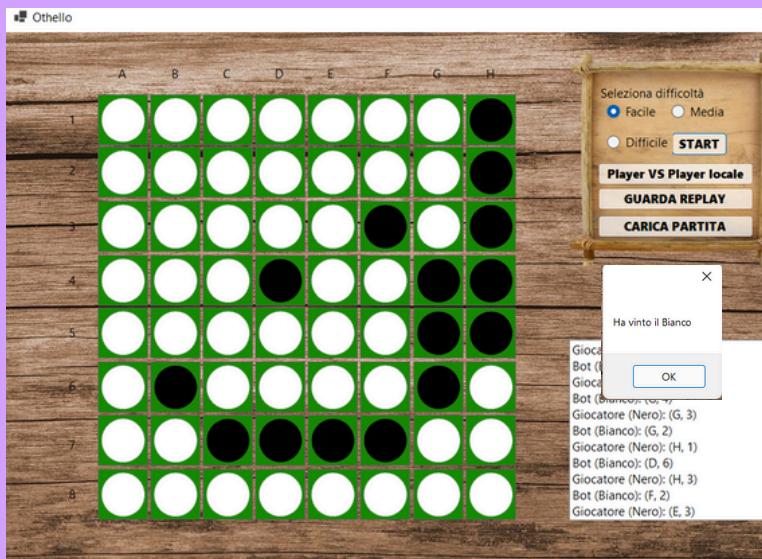
# Fase finale



Se, come in questo caso, le un giocatore mangia tutte le pedine sulla scacchiera dell'avversario, ha vinto automaticamente

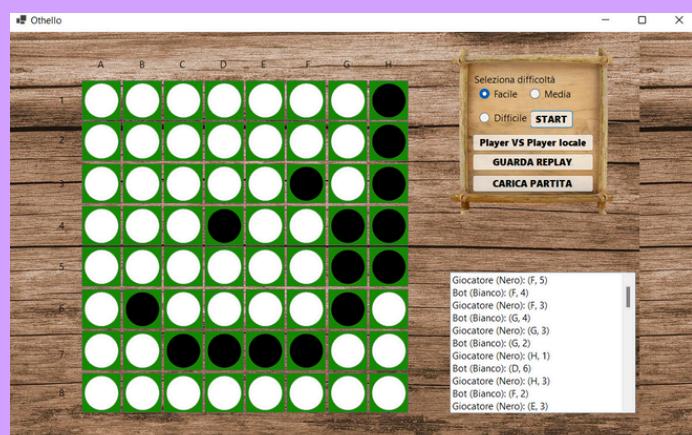
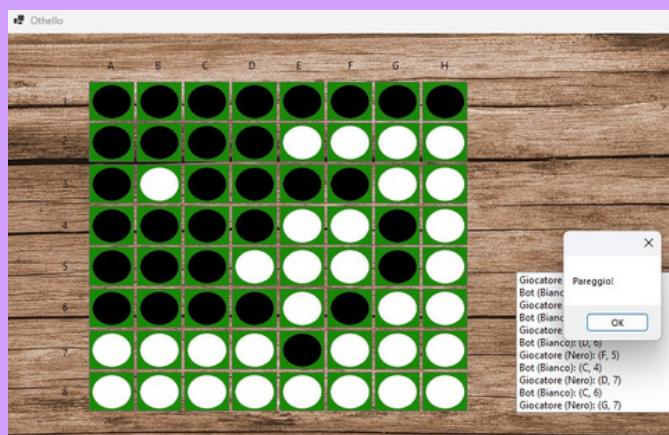


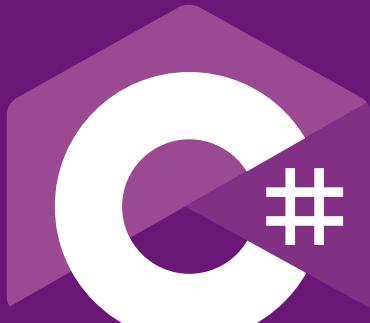
Altrimenti, c'è il conteggio delle pedine una volta completata la scacchiera



Pareggio consentito

Si può ricominciare una nuova partita!





# Grazie e a presto!

