Chess Magic Puzzle

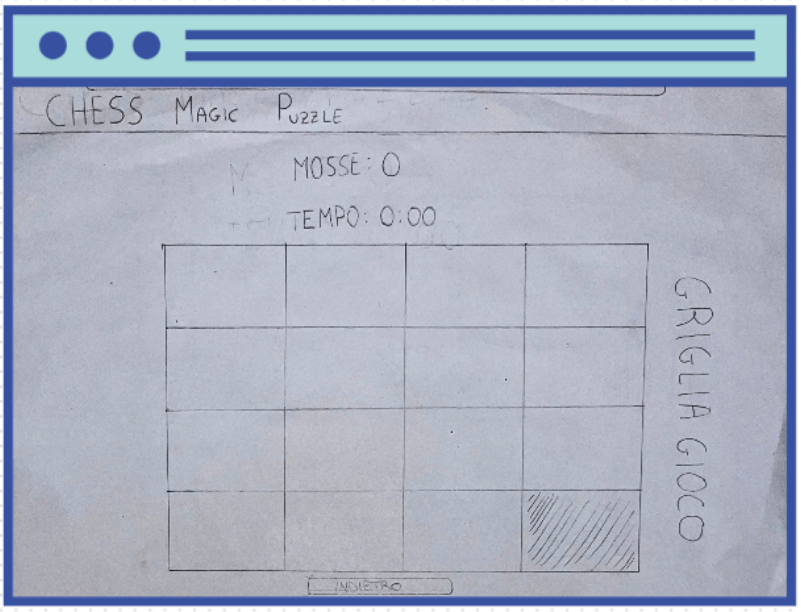
Introduzione

Questo rompicapo è concepito per affinare le capacità cognitive e promuovere il pensiero analitico. Adatto a persone di ogni età, serve come strumento per migliorare la logica e rafforzare la memoria. Ideale per chi desidera mantenere la mente attiva e allenata attraverso un esercizio mentale costante e rigoroso.

Specifica Analisi Progetto Sviluppo

indice

Sketch



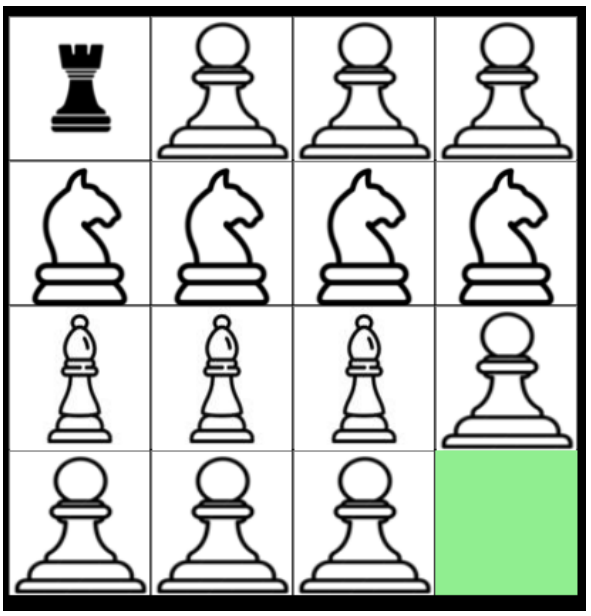
Stile grafico

La griglia di gioco è costituita da una matrice 4x4. Le specifiche della disposizione dei pezzi sono le seguenti:

1. **Configurazione della Griglia**:

   Ogni casella della griglia contiene un pezzo, ad eccezione della casella D4, che rimane libera.

2. **Tipologia dei Pezzi**:
   - Ogni pezzo sulla griglia è un pedone bianco, con l'eccezione della casella A1.
   - La casella A1 contiene una torre nera.
3. **Descrizione Visiva**:
   - Casella A1: Torre Nera
   - Caselle da B1 a D1: Pedoni Bianchi
   - Caselle da A2 a D2: Cavalli Bianchi
   - Caselle da A3 a C3: Alfieri Bianchi
   - Caselle da C3 a C4: Pedoni Bianchi
   - Casella D4: Libera



Requisiti

1)**R1**

il programma deve essere in grado di spostare i pezzi secondo le regole dello scacchi.

R1.1 cavallo si deve muovere ad L

R1.2 Alfiere si deve muovere solo in diagonale

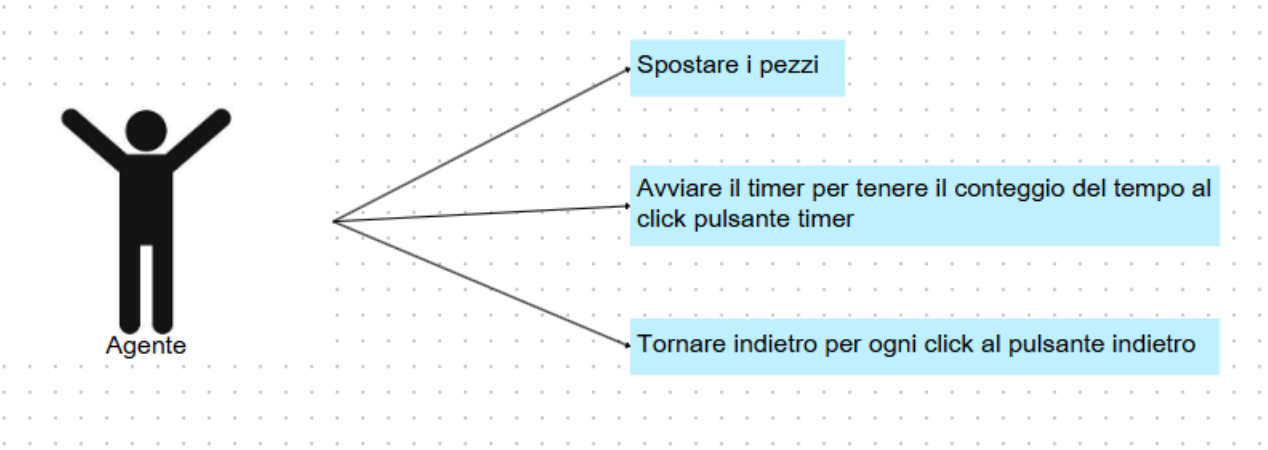
R1.3 Torre solo in verticale o orizzontale

R1.4 Pedone solo in avanti (caso eccezione se l'utente torna indietro con il bottone)

2)**L'obiettivo**

La torre deve arrivare nell'ultima casella della scacchiera la D4

3)**Casi uso (CU)**



CU1: spostamento pezzi

-**Sequenza eventi**

- Il CU1 inizia quando l'agente clicca su un qualsiasi punto all'interno della scacchiera
- Controlla se il pezzo ha delle posizioni libere nei limiti delle mosse consentite

  a) Si, allora lo sposta nella casella libera

  b) No,allora non avviene nessun spostamento

-**PostSequenza**

- Nel caso "a" si effettua uno scambio di posizioni e viene salvata nel programma;

CU2: Avvio timer

-**Sequenza eventi**

- Il CU2 inizia quando agente clicca sul pulsante "Timer"
- Viene fatta partire una funzione che gestisce un contatore per il tempo
- Poi verrà visualizzato a schermo

-**PostSequenza**

- Il "timer" finirà solamente quando il giocatore avrà finito il gioco
- Nel caso "a" si effettua uno scambio di posizioni e viene salvata nel programma;

CU3: Tasto torna indietro

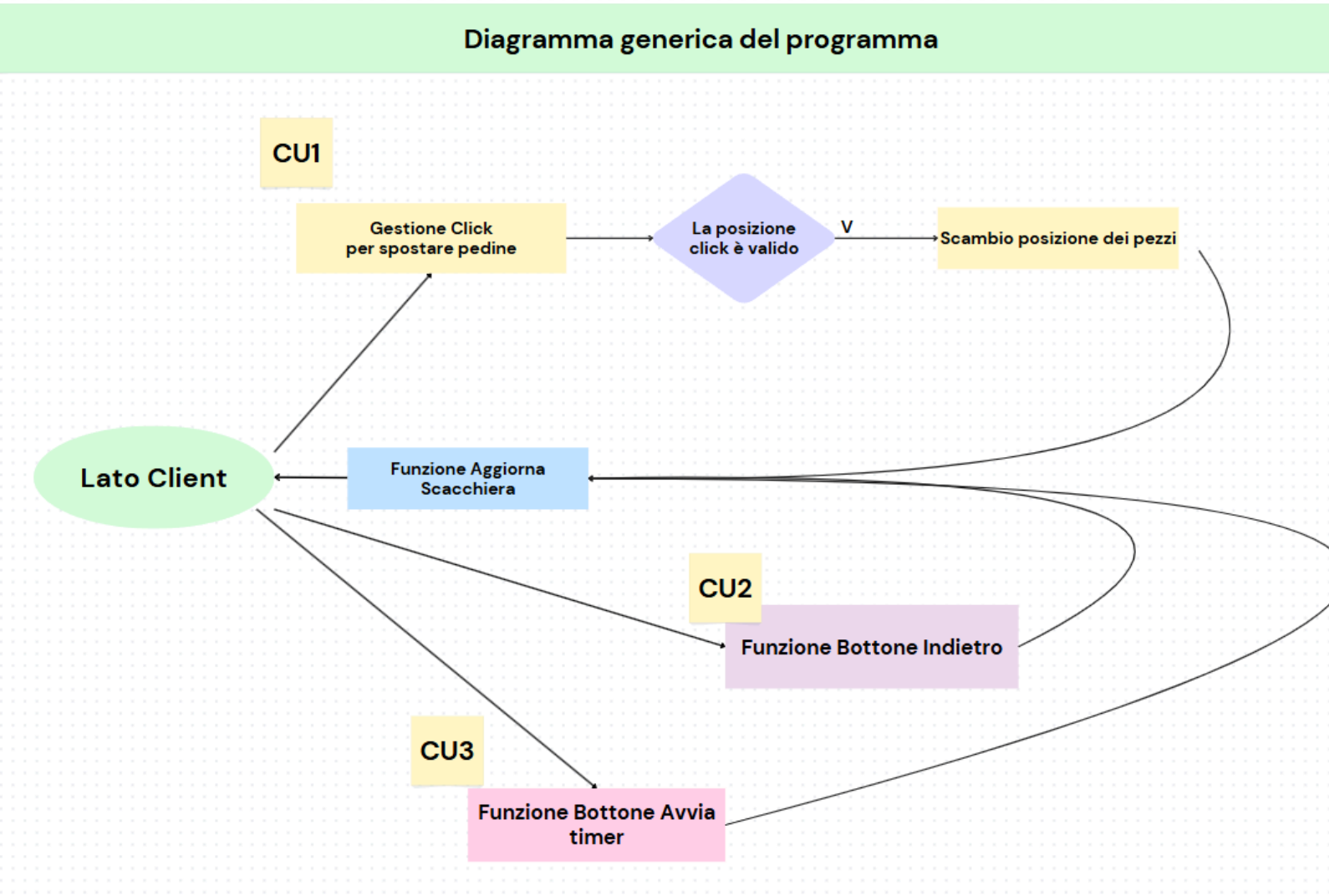
-**Sequenza eventi**

- Il CU3 inizia quando l'agente clicca sul pulsante "Indietro"
- Prende le posizioni salvate precedentemente ad ogni scambio
- Fa uno scambio della posizione

-**PostSequenza**

- La scacchiera torna indietro di uno scambio

4)**Diagramma**

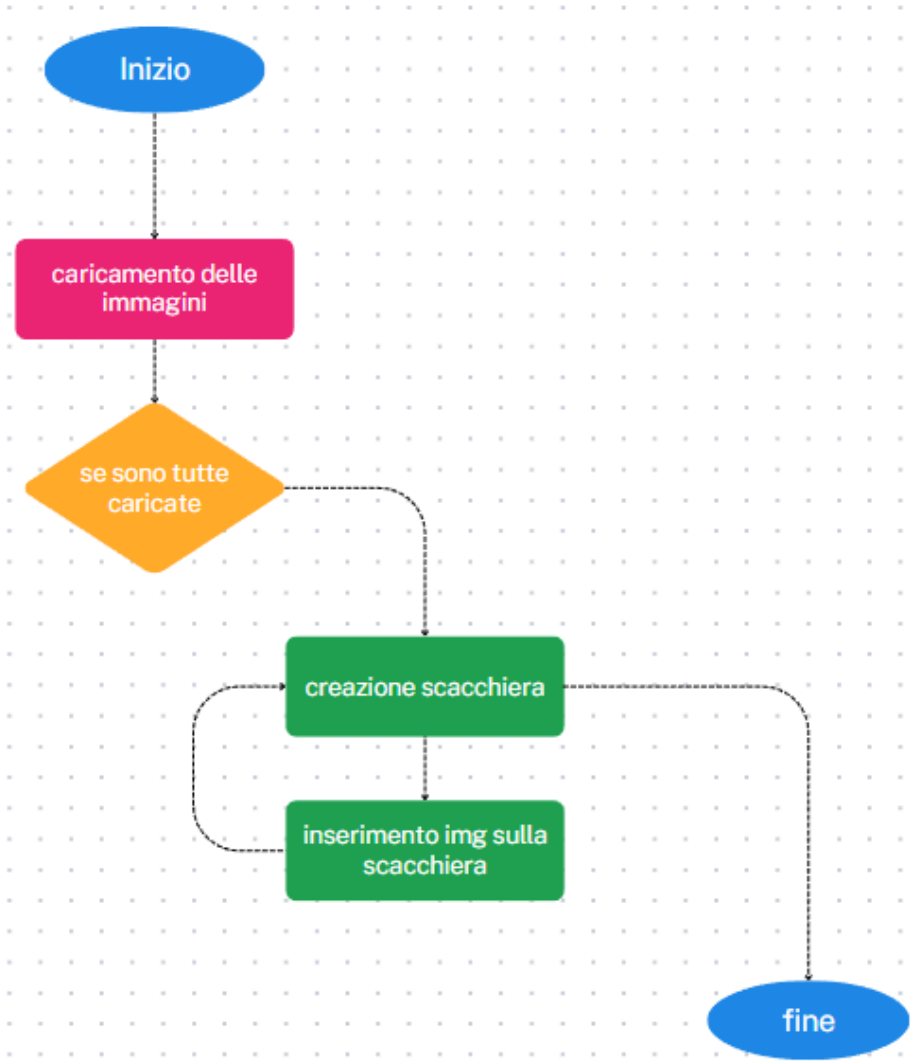


Questo diagramma serve solo come esempio per la realizzazione del gioco, per il codice e per quante parti è suddiviso il programma.

Robustezza

Analizzeremo il diagramma flusso dei vari CU per assicurarci che il programma esegua le sue funzionalità in modo corretto prima di passare al codice

Fase caricamento del gioco



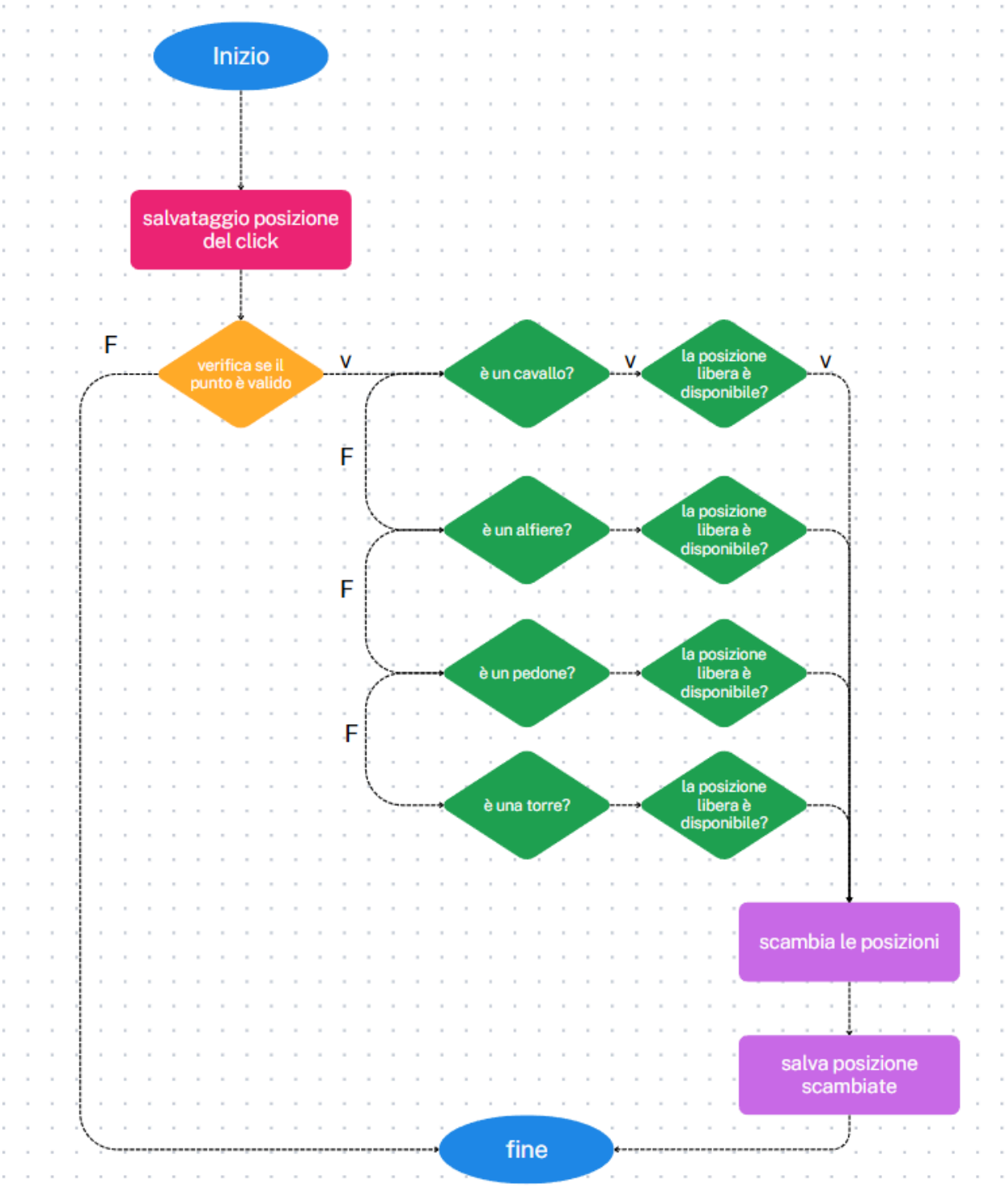
1. **Caricamento delle immagini**: Le immagini vengono caricate
   .
2. **Verifica se tutte le immagini sono caricate**: Si verifica se tutte le immagini sono state caricate correttamente.

- Se non tutte le immagini sono caricate, si ritorna alla fase di caricamento delle immagini.
- Se tutte le immagini sono caricate, si passa al passaggio successivo.

```
let imagesLoaded = 0;
pezzi.forEach(pezzo => {
    if (pezzo.src) {
        const img = new Image();
        img.src = pezzo.src;
        img.onload = () => {
            pezzo.img = img;
            imagesLoaded++;
            if (imagesLoaded === pezzi.length - 1) {
                scacchiera();
            }
        };
    }
});
```

3. **Creazione della scacchiera**: Viene creata la scacchiera.
4. **Inserimento delle immagini sulla scacchiera**: Le immagini vengono inserite sulla scacchiera nelle posizioni appropriate.

CU1



1. **Salvataggio posizione del click**: La posizione del click viene salvata.

```
canvas.addEventListener('click', (event) => {
    const rect = canvas.getBoundingClientRect();
    const mouseX = event.clientX - rect.left;
    const mouseY = event.clientY - rect.top;

    pezzi.forEach((pezzo) => {
        var pezzoSpostato=false;
        if (check(mouseX, mouseY, pezzo.x, pezzo.y)) {
```

2. **Verifica se il punto è valido**: Si verifica se il punto cliccato è valido. Se non è valido (F), il processo termina. Se è valido (V), si passa al passaggio successivo.

```
const check = (mx, my, x, y) => {
    return mx >= x && mx <= x + 100 && my >= y && my <= y + 100 && y > 0;
};
```

3. **Controlla quale pezzo si sta interagendo**: se è un cavallo, torre…
   qua sotto un esempio di come lo abbiamo gestito

```
if (pezzo.nome === 'c') {
    const mosseCavallo = [
        {x:pezzo.x-100, y:pezzo.y-200},
        {x:pezzo.x+100, y:pezzo.y-200},
        {x:pezzo.x-100, y:pezzo.y+200},
        {x:pezzo.x+100, y:pezzo.y+200},
        {x:pezzo.x+200, y:pezzo.y-100},
        {x:pezzo.x+200, y:pezzo.y+100},
        {x:pezzo.x-200, y:pezzo.y-100},
        {x:pezzo.x-200, y:pezzo.y+100}

    ];

    // Trova una casella vuota tra le mosse del cavallo
    for (const posizione of mosseCavallo) {
        // Confronta la posizione libera con la posizione corrente del cavallo
        if (posLibero.x === posizione.x && posLibero.y === posizione.y) {
            // Scambia le coordinate x e y tra i due pezzi
            scambio();
            break; // Interrompe il loop una volta trovato un movimento valido
        }
    }

}
```

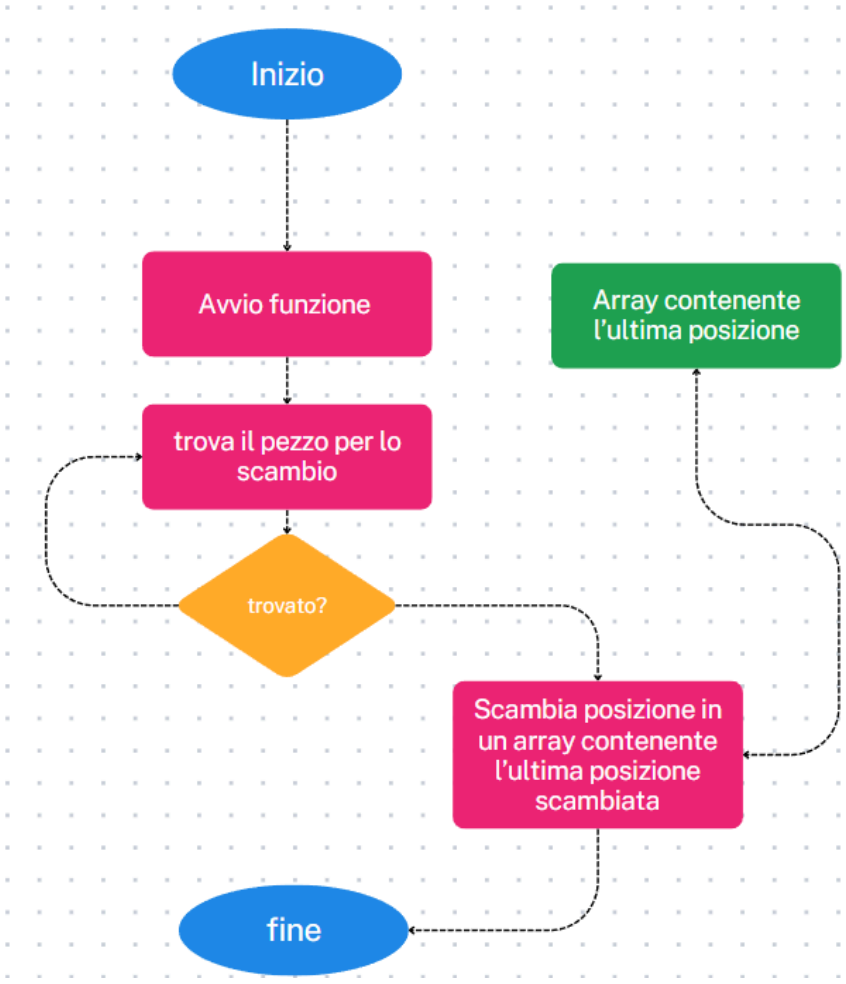
4. **La posizione libera è disponibile?**: Se la posizione libera è disponibile (V) cioè verifica che lo scambio si trova nei limiti per lo spostamento, si scambiano le posizioni e si salvano le posizioni scambiate. Se non è disponibile (F), il processo termina.

   funzione per lo scambio

```
const scambio=()=>{
    const tempX = pezzo.x;
    const tempY = pezzo.y;
    pezzo.x = posLibero.x;
    pezzo.y = posLibero.y;
    posLibero.x = tempX;
    posLibero.y = tempY;
    pezzoSpostato=true;
    cont++;
    contMosse.innerHTML = "Mosse " + cont;

}
```

CU2



1. **Inizio**: il processo ha inizio.
2. **Avvio funzione**: viene avviata una funzione.
3. **Trova il pezzo per lo scambio**: ricerca dell'elemento da scambiare.
4. **Trovato?**: verifica se l'elemento è stato trovato.

- Se sì: scambia la posizione in un array che contiene l'ultima posizione scambiata.
- Se no: fine del processo.

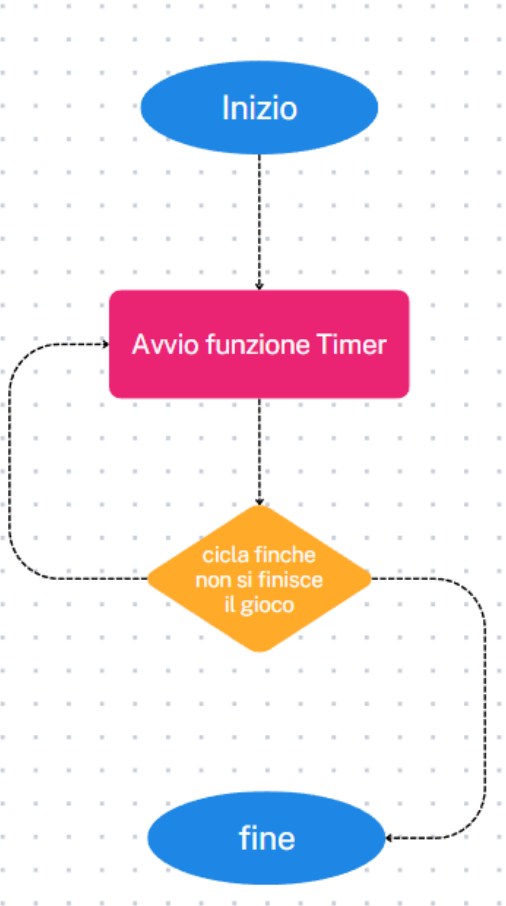
5. **Array contenente l'ultima posizione**: l'array tiene traccia delle posizioni scambiate.

```
const indietro=()=>{
    cont++;
    contMosse.innerHTML = "Mosse " + cont;
    const ultima =indietroMosse[indietroMosse.length-1];
    const [ultimaX, ultimaY] = ultima;
    pezzi.forEach((pezzo)=>{
        if((pezzo.x==ultimaX) && (pezzo.y==ultimaY)){

            const tempX = pezzo.x;
            const tempY = pezzo.y;
            pezzo.x = posLibero.x;
            pezzo.y = posLibero.y;
            posLibero.x = tempX;
            posLibero.y = tempY;
            indietroMosse.pop();
            console.log("ciao")

        }
        scacchiera();
    });
}
```

CU3



1. **Inizio**: il processo comincia con questo punto di partenza.
2. **Avvio funzione Timer**: viene avviata una funzione timer.
3. **Ciclo finché non si finisce il gioco**: continua a ripetere le operazioni finché il gioco non termina.
4. **Fine**: il processo si conclude qui.

```
const timer = () => {
    let tempo = document.getElementById('tempo');
    let secondi = 0;  // inizializzazione della variabile secondi
    let minuti = 0;   // inizializzazione della variabile minuti
    let ore = 0;

    const inizio = setInterval(() => {
        secondi++;

        // Calcolo dei minuti e dei secondi
        if (secondi === 60) {
            secondi = 0;
            minuti++;
        }
        if (minuti === 60) {
            minuti = 0;
            ore++;
        }

        // Visualizzazione del tempo in formato MM:SS
        tempo.innerHTML = "Tempo: " + String(minuti).padStart(2,

        // Condizione per terminare il gioco e aggiornare il temp
        if (pezzi[14].x == 301 && pezzi[14].y == 301) {
            clearInterval(inizio); // Ferma il timer
            tempo.innerHTML = "Hai completato il gioco in Tempo:

        }

    }, 1000);  // Aggiorna ogni secondo (1000 millisecondi)
};
```