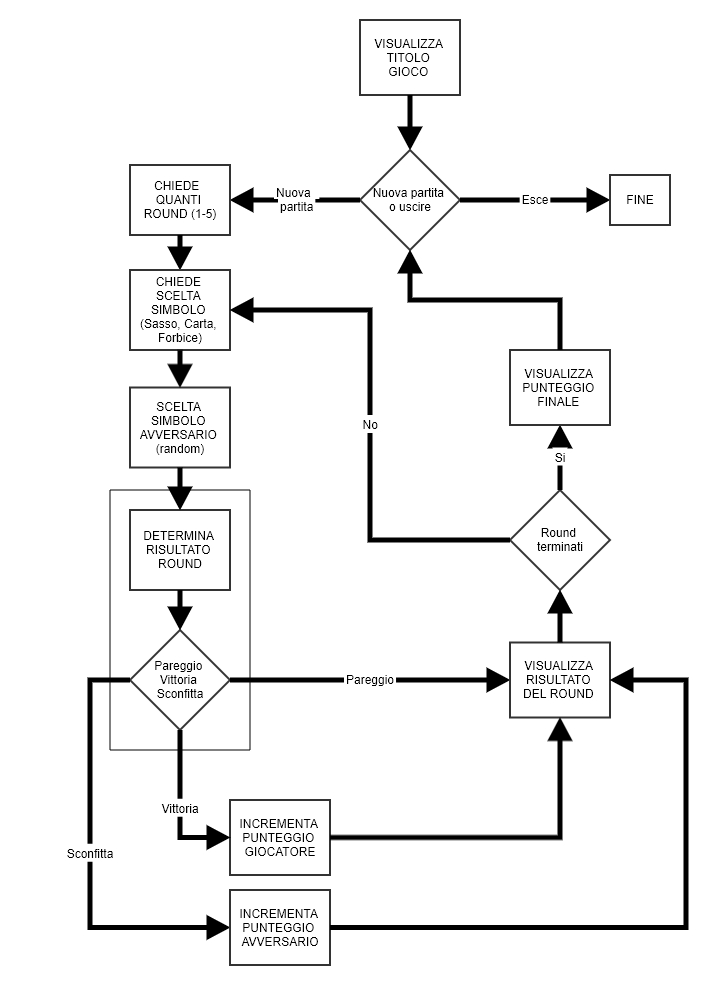
MORRA CINESE

# DIAGRAMMA DI FLUSSO



# SPECIFICHE

Si richiede di implementare il tipico gioco della Morra Cinese. Il giocatore sfiderà un avversario virtuale gestito dal computer.

Ogni partita è composta da round (1-5). Il numero di round va scelto dal giocatore prima di iniziare la partita.

Ad ogni round il giocatore e l’avversario scelgono uno fra i tre simboli della morra: Sasso, Carta, Forbici.

Le comuni relazioni fra simboli sono:

* Sasso batte Forbici
* Carta batte Sasso
* Forbici batte Carta

Ad ogni round, il vincitore guadagna un punto.

Alla fine della partita (fine dei round) viene decretato come vincitore colui che ha ottenuto il maggiore punteggio.

In caso di pareggio (è possibile se i round scelti sono pari), va mostrato comunque il messaggio di pareggio con i relativi punteggi.

Alla fine di ogni partita, l’utente può iniziare una nuova partita oppure uscire dal gioco.

# COMMENTI AL DIAGRAMMA

1. **Visualizza titolo gioco**Nessun commento
2. **Nuova partita o uscire**

Dallo schema di flusso sembra che la risposta non sia il classico Si / NO, ma ovviamente, essendo una domanda, può sempre essere espressa in modo che il risultato (output) sia Si o NO.

1. **Chiede quanti round**

La richiesta accetta un numero fra 1-5 (estremi compresi). Come si vede dallo schema, non c’è una esplicita verifica se l’utente ha inserito qualcosa di non valido (numeri negativi, numeri > 5, oppure testo che non sia un numero).  
La cosa migliore è mantenere tutto dentro la stessa funzione al cui interno all’utente deve essere ripresentata la domanda se non ha inserito qualcosa di valido.  
AIUTO: Usa la funzione data “IsEqualToNumber” che verifica se una stringa è uguale ad un numero specifico.

1. **Chiede scelta simbolo**

Come sopra, sarebbe bene mantenere anche il controllo di validità della scelta dentro la funzione stessa. Quindi anche qui non ne dovrebbe uscire senza aver fatto una scelta valida.  
A questo proposito qui c’è un evidente “compromesso”. La scelta da fare è in realtà un simbolo. Il fatto che si chiami “Carta” per esempio, non significa necessariamente che l’utente deve scegliere la stringa “Carta”.   
In realtà in queste situazioni bisognerebbe lavorare con le classi, ma ovviamente in questo momento va fatta una scelta di compromesso.  
Io nel mio test implementativo, ho deciso che la cosa migliore fosse rappresentare i tre simboli come tre interi 1 (Sasso), 2 (Carta), 3 (Forbici).  
In pratica siamo “costretti” a stabilire un mapping convenzionale fra dei numeri e dei simboli (che dovrebbero essere invece oggetti).  
Dico che siamo “costretti” perché questa scelta convenzionale dovremo seguirla in tutto il gioco (es. quando dovrai stabilire chi ha vinto un round confrontando i simboli).  
NB: la mia scelta di mapping è assolutamente arbitraria. Scegli pure liberamente la corrispondenza e l’ordine dei simboli con gli interi. Hai a disposizione la funzione data GetRandomNumber per generare interi all’interno di un range deciso da te.  
**(\* VEDI VARIANTE FINALE)**

1. **Scelta simbolo avversario**Deve calcola randomicamente un numero fra 1-3, che dovrà rappresentare un simbolo della morra. Sta a te, come detto, decidere che simboli associare ai 3 valori interi.   
   Usa la funzione data GetRandomNumber.  
   **(\* VEDI VARIANTE FINALE)**
2. **Determina risultato round + Pareggio, Vittoria, Sconfitta**

Questo è lo step più delicato del flusso. Idealmente io l’ho diviso in due potenziali funzioni: una che calcola il risultato (confrontando i simboli) e l’altra che determina se si tratta di pareggio, vittoria o sconfitta.  
Qui non c’è in realtà una vera soluzione corretta. Potrebbe essere anche una sola funzione.  
Il punto delicato è questo: il rombo “Pareggio vittoria sconfitta” sottintende una domanda, ma come vedi le risposte sono 3. In realtà però, in caso di Pareggio, non ci sono azioni da fare e si passa subito alla visualizzazione del risultato del round.  
Invece, in caso di Sconfitta o Vittoria vanno aggiornati i punteggi.  
Questo potrebbe suggerire che in realtà quel rombo va sdobbiato: un primo rombo che determina se c’è stato pareggio o no. In caso negativo, si passa al secondo rombo che chiede se il giocatore ha vinto o no. Così si torna alle funzione booleane classiche.

1. **Incrementa punteggio giocatore/avversario**Attenzione che questa operazione ci aveva dato problemi nell’esercizio del gioco a quiz. Ragionando a voce alta eravamo arrivati a definire una funzione DecrementAttemptsLeft() che decrementava i tentativi (attemptsLeft--), ma avevamo visto che all’uscita di quella funzione (corretta dal punto di vista logico), il decremento non aveva effetto. E ti avevo spiegato che era un problema tecnico legato al concetto di passaggio per riferimento o per valore. E quindi avevamo scelto di decrementare esplicitamente i tentativi senza ricorrere ad una funzione ad-hoc.  
   Qui sei in una situazione molto simile (prima o poi vedremo qual è il problema tecnico sottostante, tutt’altro che banale).
2. **Visualizza risultato del turno**

Niente di particolare. Va solo mostrato il confronto dei due simboli scelti, indicando chi ha vinto quel round. Se vuoi puoi mostrare anche il punteggio aggiornato in quel round.

1. **Round terminati**

Classico test per verificare se continuare la partita attuale o meno

1. **Visualizza punteggio finale**

Semplicemente il risultato finale con i punteggi e la dichiarazione del vincitore

# FUNZIONI DATE

1. **IsEqualToNumber:** richiede in input una stringa da verificare e un numero intero come target. Restituisce True se la stringa rappresenta esattamente il numero target. False altrimenti.
2. **GetRandomNumber:** nessun input richiesto. Restituisce in maniera randomica un numero intero fra due estremi “inlcusi”.   
   NB: non definisce esplicitamente la corrispondenza fra un numero e un simbolo della morra!

# VARIANTE

La rappresentazione dei simboli tramite interi è il fulcro “nuovo” di questo esercizio (a parte prendere dimestichezza con il pensiero “a funzioni”).

Come detto, la cosa migliore sarebbe usare le classi (o anche le Enum come vedremo), ma per ora siamo limitati ai tipi base e agli array.

Proprio con gli array potremmo fare una variante del gioco.

Potremmo usare un array di 3 stringhe, con dentro le stringhe “Sasso”, “Carta”, “Forbici” che rappresenta l’array dei simboli disponibili.

A questo punto potremmo sfruttare la funzione data GetRandomNumber per scegliere randomicamente (nel caso dell’avversario) una fra le tre stringhe dell’array dei simboli disponibili.

Il giocatore ovviamente farà la stessa cosa ma non lo farà randomicamente.

La conseguenza in questo caso è che i simboli non saranno più dei valori interi, ma saranno esplicitamente delle stringhe.

Da questo punto di vista, modellare i simboli come interi o come stringhe, nel nostro gioco cambia poco. Al momento del confronto dei simboli per determinare il vincente, non cambierà granchè.

Il punto da “portare a casa” qui è la necessità di rappresentare concetti “complessi” come i simboli della morra cinese, tramite dei tipi di dato base (interi, stringhe…). Questo è il principale motivo per cui servono tipi di dato avanzati come le classi appunto.