



# Console di gioco

21/05/2023

Anno 22/23

---

Alessandro Bettini

Informatica

## Sommario

<b>Descrizione del progetto</b>	<b>1</b>
<b>Giochi della Console</b>	<b>2</b>
Tris	2
Forza 4	2
Battaglia Navale	3
<b>Processo creativo</b>	<b>3</b>
<b>Inizio del codice e main()</b>	<b>3</b>
<b>Tris</b>	<b>9</b>
Tris PvP	9
Tris PvE modalità facile	11
Tris PvE modalità difficile	11
<b>Forza 4</b>	<b>14</b>
Forza 4 PvP	14
Forza 4 PvE facile	16
Forza 4 PvE difficile	16
<b>Battaglia Navale</b>	<b>18</b>
Battaglia Navale PvP	18
Battaglia navale PvE facile	21
Battaglia navale PvE difficile	22
<b>Conclusioni e pensieri finali sul progetto</b>	<b>24</b>

## Descrizione del progetto

Ad inizio secondo quadrimestre ci è stato assegnato il compito di creare una **Console di gioco PvP** in linguaggio C. Il compito consisteva nella creazione di una console da cui si potevano scegliere 3 videogiochi con cui passare il tempo, **Tris**, **Forza 4** e **Battaglia Navale**.

Il nostro lavoro era di programmare questi 3 giochi in modalità Pvp, insieme ad un menù principale ed una classifica che teneva conto dei progressi dei giocatori e dei punteggi.

Col passare del tempo questo compito è stato amplificato dal Professor Fazzone aggiungendo ai 3 giochi in modalità PvP 2 nuove modalità, ovvero la **PvE facile** e **PvE difficile**.

PvP tradotto è ovviamente **Player vs Player** mentre PvE significa **Player vs Environment** (ambiente), dove l'ambiente viene associato al computer. Nei giochi con modalità come queste più vecchi "l'ambiente" viene chiamato proprio Computer o addirittura CPU per sottolineare il fatto che non ci sia un giocatore dietro le azioni che il Player si trova contro in gioco bensì un codice che analizza le partite e si comporta in base alle condizioni presenti avendo come obiettivo la vittoria.

La creazione quindi di veri e propri **BOT** contro cui far giocare i Player mi ha inizialmente esaltato tanto, vista la difficoltà che avrebbe avuto la programmazione che mi aspettava.

## Giochi della Console

Come detto sopra i giochi della console sono Tris, Forza 4 e Battaglia Navale, pur essendo 3 giochi famosissimi perchè molto praticati dai bambini, diamo un'occhiata a come funzionano questi 3 giochi e a cosa sono.

### Tris

Il famoso gioco del Tris (Tic-Tac-Toe in inglese) nasce come gioco su **carta e matita** astratto.

La "mappa" di gioco è una tabella 3X3 (ovviamente nella programmazione viene usata una matrice) dove a turno i 2 giocatori mettono una X o un O con obiettivo metterne 3 di fila in verticale, orizzontale o obliquo.



## Forza 4

Forza 4 (Connect Four in inglese) nasce invece come gioco da tavolo astratto della **Milton Bradley** nel 1974. L'obiettivo del gioco è molto simile a quello del tris, il giocatore per vincere deve mettere **4** pedine del proprio colore (solitamente rosso o giallo) di fila, in verticale, orizzontale o obliquo. Il fattore dominante del gioco però è la presenza della gravità, che impedisce infatti ai giocatori di mettere pedine in tutti gli spazi della tabella (6X7), bensì sempre nello spazio più in fondo alla colonna.



## Battaglia Navale

La battaglia navale nasce come gioco su **carta e matita** per 2 giocatori per poi diventare anche gioco da tavolo per mano della Milton Bradley. Il gioco è composto da due griglie di 10X10 dove i giocatori mettono le proprie navi. Dopodiché si passa alla fase degli attacchi dove i giocatori a turno "sparano" in un punto della tabella tramite le coordinate (A5 ad esempio) e attendono che l'avversario dica se la nave è stata colpita. Quando la nave viene colpita in tutti i suoi punti affonda, vince il gioco chi **affonda tutte le navi avversarie** per primo.



## Processo creativo

Prima di cominciare a programmare i giochi e i bot ho dovuto decidere come organizzare la Console.

La prima idea che ho avuto si è subito rivelata la più efficace per me e difatti quella che ho utilizzato durante tutta la creazione della Console.

Ho deciso di creare un menù nel Main da cui si possono scegliere i giochi e poi inserire **ogni singolo gioco in una diversa funzione** chiamata alla scelta nel menù.

Così facendo mi sono sempre trovato con un codice ordinato e pulito, senza alcuna difficoltà nel trovare errori e pezzi di codice da aggiustare né durante né dopo la programmazione.

Creare il menù e la classifica è stato davvero semplice, niente di quello che ho fatto era nuovo per me e quindi camminavo in territori conosciuti. La **scrittura dei giochi** invece è stata un po' più faticosa, soprattutto una volta arrivati alle modalità PvE difficili.

## Inizio del codice e main()

### Librerie incluse nel codice

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>
#include <time.h>
```

Le librerie usate per il codice sono 5:

- **stdio.h**: la libreria standard C che contiene varie dichiarazioni di costanti, funzioni e tipi usati per input/output;
- **stdlib.h**: standard library per allocazione della memoria, controllo dei processi e funzioni come il random utilizzato per la creazione dei giochi PvE facili;
- **string.h**: libreria standard che contiene le definizioni delle funzione per l'utilizzo di stringhe come strcmp e strcpy;
- **windows.h**: libreria standard C/C++ per le funzioni Windows API come la funzione Sleep();
- **time.h**: libreria standard che fornisce accesso alle funzioni di manipolazione del tempo come srand(time(0)).

### Funzioni del codice

```
//funzioni tris
int tris();
int tris_easybot();
int tris_hardbot();

//funzioni battaglia navale
int battnavale();
int battnavale_easybot();
int battnavale_hardbot();

//funzioni forza4
int forza4();
int forza4_easybot();
int forza4_hardbot();
int forza4_controlli(int c, int r);
int forza4_controlli_bot(int c, int r);
```

Le funzioni che ho scritto e utilizzato per il codice sono in totale **11** e servono per i singoli giochi. Solo dai nomi è molto semplice intuire a cosa servono le varie funzioni ma ecco una descrizione più dettagliata:

- Int tris(): funzione contenente il gioco del **tris PvP**;
- Int tris\_easybot(): funzione contenente il gioco del tris PvE in **modalità facile**;
- Int tris\_hardbot(): funzione contenente il gioco del tris PvE in **modalità difficile**;
  
- int battnavale(): funzione contenente il gioco della **battaglia navale PvP**;
- int battnavale\_easybot(): funzione contenente il gioco della battaglia navale PvE in **modalità facile**;
- int battnavale\_hardbot(): funzione contenente il gioco della battaglia navale PvE in **modalità difficile**;
  
- int forza4: funzione contenente il gioco del **forza 4 PvP**;
- int forza4\_easybot(): funzione contenente il gioco del forza 4 PvE in **modalità facile**;
- Int forza4\_hardbot(): funzione contenente il gioco del forza 4 PvE in **modalità difficile**;
- Int forza4\_controlli(int c, int r): funzione utilizzata dentro le funzioni forza4\_easybot e forza4\_hardbot contenente i **controlli per la vittoria** del giocatore 1 o del bot;
- Int forza4\_controlli\_bot(int c, int r): funzione utilizzata dentro la funzione forza4\_hardbot contenente i **controlli** che il **bot** esegue prima di fare la mossa per decidere dove posizionare il cerchio in modo da fermare il giocatore.

## Main()

Nel main incontriamo subito la dichiarazione delle variabili ovviamente e l'apertura del file **risultati partite.txt** dove viene salvato il risultato di ogni singola partita, il file funge quindi da storico.

Avviando poi la console ci ritroviamo davanti a questo menù:

```
GIOCHI PVP

Menu'
1-Tris
2-Battaglia Navale
3-Forza 4
4-Leaderboard
5- Exit
```

Il menù è costruito con uno **switch** che prende l'input di scelta del gioco/leaderboard.

```
//MENU E LEADERBOARD
while(flag==0)
{
    system("cls");
    printf("GIOCHI PVP\n\n");
    printf("Menu'");
    printf("\n1-Tris\n2-Battaglia Navale\n3-Forza 4\n4-Leaderboard\n5- Exit\n");
    fflush(stdin);

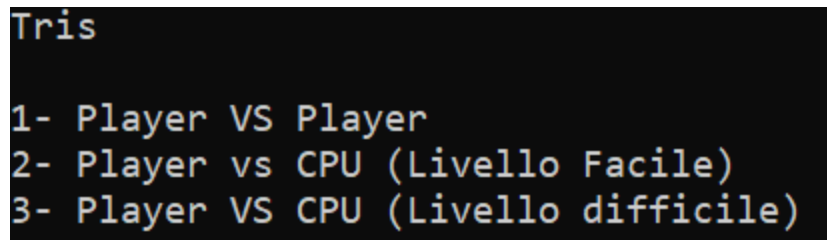
    scanf("%d",&gioco);

    switch(gioco)
```

A questo punto lo switch fa ogni singolo caso.

Dentro i **case 1/2/3** c'è un'altra scelta ovvero quella della **modalità di gioco, PvP o PvE** e della difficoltà in caso di scelta della seconda.

Il menù si presenta così:



```
Tris

1- Player VS Player
2- Player vs CPU (Livello Facile)
3- Player VS CPU (Livello difficile)
```

Ovviamente il menù cambia a discrezione della scelta del gioco.

I case 1/2/3 del menù principale quindi stampano prima il menù singolo e poi hanno un altro switch con 3 case per la scelta della modalità.

```
case 1:

    system("cls");
    printf("Tris\n");
    printf("\n1- Player VS Player\n2- Player vs CPU (Livello Facile)\n3- Player VS CPU (Livello difficile)\n");
    do
    {
        scanf("%d",&scelta_bot);

        if(scelta_bot==1 || scelta_bot==2 || scelta_bot==3){
            flag_bot=1;
        }
        else{
            printf("Scelta non valida, reinserire: ");
        }
    }

    }while(flag_bot==0);

    switch(scelta_bot)
```

Un case invece del secondo menù è composto dalla **chiamata alla funzione** corrispondente e poi da 3 if che controllano il valore restituito dalla funzione una volta completata e scrivono sul file il punteggio aggiornato, inoltre aggiornano anche la **leaderboard**.

```
case 1:
    n=tris();
    fflush(stdin);
    if(n==1){
        tris1++;
        fprintf(op, "Tris: Vince player 1.\n");
    }
    else if(n==0){
        tris2++;
        fprintf(op, "Tris: Vince player 2.\n");
    }
    else if(n==2){
        tris1++;
        tris2++;
        trisCPU++;
        fprintf(op, "Tris: Pareggio.\n");
    }
    else if(n==3){
        trisCPU++;
        fprintf(op, "Tris: Vince CPU.\n");
    }
    break;
```

## Leaderboard

La leaderboard invece si trova al case 4 del primo switch.

Nella console la leaderboard si presenta così:

```
Leaderboard:
           P1           P2           CPU
Tris      1            0            0
Battaglia navale  0            0            0
Forza4     0            0            0

Premere per tornare al menu':
```



Come di nota è composta da 3 colonne e 3 righe. Le colonne indicano **Player 1, Player 2** e **CPU** mentre le righe il gioco.

Nel codice la leaderboard è un insieme di printf che scrivono nomi e punteggi. I punteggi sono costantemente aggiornati nei case sopra. Quando la funzione gioco viene chiamata restituisce un intero che rappresenta la vittoria di uno dei tre possibili vincitori oppure il pareggio.

Quando il programma capisce chi ha vinto la partita appena finita grazie agli IF precedenti **aggiorna un flag** che simboleggia quel giocatore in quel gioco.

Se avviene un **pareggio** vengono aggiornati entrambi i giocatori che hanno preso parte alla partita.

```
case 4:
    system("cls");
    fflush(stdin);
    printf("Leaderboard: ");
    printf("\n          P1      P2      CPU");
    printf("\nTris          %d      %d      %d", tris1, tris2, trisCPU);
    printf("\nBattaglia navale %d      %d      %d", batt1, batt2, battCPU);
    printf("\nForza4          %d      %d      %d\n\n", forz1, forz2, forzCPU);
    printf("Premere per tornare al menu: ");
    scanf("%d",&inutile);

    break;
```

## Tris

### Tris PvP

Il gioco del **tris PvP** è realizzato nella funzione tris().

La funzione riempie una matrice 3X3 di '#' con un ciclo for e la stampa a video.

Dopodiché inizia un ciclo **while(flag==0)** che serve per i turni dei 2 giocatori e che finisce soltanto quando uno dei due player fa Tris.

Il tris a schermo si presenta così:

```
0 1 2
0 # # #
1 # # #
2 # # #
Giocatore 1: inserire coordinate:
```

Dentro il while si inizia con il **turno del primo giocatore** a cui viene chiesto di inserire riga e colonna. Dopo aver inserito le coordinate ci sono i **controlli di validità** di queste.

Con un ciclo while si controlla quindi se le coordinate sono libere e se sono comprese tra 0 e 2 inclusi, in caso non lo siano vengono richieste con un altro scanf.

```
while(flag1==0)
{
    if(griglia[P1rig][P1col]!='#' || P1rig<0 || P1rig>2 || P1col<0 || P1col>2)
    {
        printf("Tabella gia' occupata o non valida, reinserire: ");
        scanf("%d",&P1rig);
        scanf("%d",&P1col);
    }
    else
    {
        flag1=1;
    }
}
```

Dopo aver fatto questo lo schermo viene pulito con **system("cls")** e la griglia viene stampata con la X posizionata dal Player 1.

Poi la funzione effettua i controlli di vincita tramite un if molto lungo che comprende tutti i casi di possibile vittoria nel tris e controlla se uno di essi è presente nella griglia.

In caso un caso sia presente stampa a schermo la frase Giocatore 1 Vince e *restituisce 1 al main*.

```
if(griglia[0][0]=='X' && griglia[1][0]=='X' && griglia[2][0]=='X' || griglia[0][1]=='X' && griglia[1][1]=='X' && griglia[2][1]=='X' || griglia[0][2]=='X' && griglia[1][2]=='X' && griglia[2][2]=='X')
{
    printf("\nGiocatore 1 Vince!!!");
    win=1;
    printf("\n\nPremere per continuare: ");
    scanf("%d",&inutile);
    flag=1;
    system("cls");
    return win;
}
```

La vittoria nel tris si presenta così:

```
  0 1 2
0 0 # X
1 # X #
2 X # 0

Giocatore 1 Vince!!!

Premere per continuare:
```

Finito il controllo di vincita in caso il Player 1 non abbia fatto tris viene effettuato il **controllo del pareggio** che consiste in un ciclo for che controlla se nella griglia sono ancora presenti '#' e aggiorna un contatore quando trova uno spazio diverso da '#'. Se alla fine del for il contatore è a 9 allora la partita finisce in pareggio e viene restituito il valore 2 che significa pareggio al main.

*Lo stesso identico processo si ripete poi per il giocatore 2.*

```
  0 1 2
0 X # 0
1 X 0 #
2 0 # X

Giocatore 2 Vince!!!

Premere per continuare:
```

## Tris PvE modalità facile

La **modalità facile PvE** del tris è stata davvero **facile da realizzare**.

Tutto ciò che è servito fare è infatti stato sostituire con dei **random** le scanf del giocatore 2 nel tris PvP, eliminare alcuni printf e cambiare quelli di vincita e il valore restituito in caso di vittoria della CPU.

```
//turno CPU
```

```
CPUrig=rand()%3;  
CPUcol=rand()%3;
```

Lo schermo si presenta così alla vittoria della **CPU**:

```
  0 1 2  
0 X X O  
1 X O X  
2 O # O  
  
CPU Vince!!!  
  
Premere per continuare:
```

## Tris PvE modalità difficile

Programmare il **tris PvE in modalità difficile** è stato invece molto più **impegnativo** di ciò che immaginavo a differenza di quello precedente.

Ho deciso di partire dal **codice iniziale del tris**. Ovviamente ho lasciato invariata la parte riguardante il giocatore 1 e ho subito eliminato la parte del giocatore 2.

A questo punto ho iniziato a pensare a diverse soluzioni, tutte ancora incerte e non ben chiare nella mia testa. Quindi ho scritto qualche linea di codice utilizzando un **flagturn** per differenziare la mossa in base al turno e quindi in base al numero di X e O presenti nella griglia.

Purtroppo non si è rivelata essere una buona idea visto che in solo un'ora di lavoro mi ero ritrovato con **troppi IF da gestire, troppi flag per troppe possibili combinazioni e un codice disordinato e sbagliato**.

Allora ho iniziato a pensare ad altre soluzioni fino a che sono giunto all'idea migliori di tutte, far agire il bot utilizzando i **controlli di vincita** usati nei bot precedenti.

Ho mantenuto l'idea del flagturn, ma questa volta serviva solo per differenziare il primo turno dagli altri, non tutti quanti.

In caso che fosse il primo turno ho deciso di **randomizzare la posizione della O** con due random in un ciclo while.

```

if(flagturn==1)                                //primo turno
{
    do{
        i=rand()%2;
        j=rand()%2;
    }while(griglia[i][j]!='#');

    griglia[i][j]='O';
}

```

Quando invece il flagturn è diverso da 1 (quindi maggiore) ci sono **due cicli for** uno dentro l'altro che percorrono tutta la matrice partendo dalla posizione[0][0] in alto a sinistra fino alla posizione[2][2] in basso a destra.

```

for(i=0;i<3;i++)
{
    for(j=0;j<3;j++)
    {
        if(griglia[i][j]=='#')
        {
            griglia[i][j]='O';
            if(griglia[0][0]=='O' && griglia[1][1]=='O')
            {
                griglia[i][j]='O';
                nul=0; //controllo attacco
                goto outfor;
            }
            griglia[i][j]='X';
            if(griglia[0][0]=='X' && griglia[1][1]=='X')
            {
                griglia[i][j]='O';
                nul=0; //controllo difesa
                goto outfor;
            }
            else
            {
                nul=1;
                griglia[i][j]=='#';
            }
        }
    }
}

```

Nel mentre che viene fatto questo si controlla se la posizione in cui si trova il for è libera (quindi '#'). Se non lo è va avanti nel for. Se invece la posizione è libera si fanno 2 cose diverse.

Prima si scrive una **O** nella posizione e con un **IF** si fa il controllo di vittoria. Se il controllo di vittoria si avvera significa che con quella O il bot può vincere quindi la posiziona **definitivamente** in quel punto della griglia ed **esce dal for**.

Se invece l'IF non si avvera in quella posizione viene messa una **X** e si fa la stessa cosa. Se l'IF si avvera significa che il giocatore 1 alla prossima mossa può vincere, quindi in quel punto della griglia **viene messa una O** e poi si esce dal for.

Se nessuno dei due casi si avvera c'è un ***else che fa tornare la posizione '#'*** e continua nel for.

Inoltre c'è un ***flag*** chiamato ***nul*** che viene attivato in caso dentro al for non venga messa alcuna O.

Fuori dal for c'è un IF che controlla il valore di questo flag nul, se è uguale a 1 allora ***randomizza*** un punto della griglia e ci inserisce una O.

```
if(nul==1)
{
    nul=0;
    do{
        i=rand()%2;
        j=rand()%2;
    }while(griglia[i][j]!='#');
    griglia[i][j]='O';
    goto outfor;
}
outfor:
```

## Pensieri sul tris

Il gioco del tris e i suoi bot sono probabilmente stati i più ***facili da realizzare*** tra i giochi. La vera e unica difficoltà è stata il ***PvE difficile***, perchè ho voluto cercare di farlo corto e semplice, ma comunque efficiente da non esser facilmente battibile e penso di aver fatto un buon lavoro per questo gioco e questi bot.

## Forza 4

### Forza 4 PvP

Il gioco del forza 4 PvP è realizzato nella funzione *forza4()*.

Inizialmente viene riempita una matrice di **6X7** di '#', proprio come nel tris a parte per le dimensioni e viene stampata.

```

# # # # # # #
# # # # # # #
# # # # # # #
# # # # # # #
# # # # # # #
# # # # # # #
-----
 0 1 2 3 4 5 6
Colonna P1:

```

La prima cosa a cui pensare per forza 4 era il **fattore gravità** perché come sappiamo per mettere le pedine il giocatore deve scegliere la colonna ed essa cadrà nella riga più in fondo.

Per occuparmi quindi dei controlli di validità dell'input ho fatto prima un **while** per controllare se fossero compresi tra 0 e 6, poi un if per controllare se la colonna dove si voleva inserire fosse già piena e in caso far reinserire la pedina.

```

while(r<0 || r>6)
{
    printf("Non valida, inserisci di nuovo: ");
    scanf("%d",&r);
}

if(tabella[0][r]=='X' || tabella[0][r]=='O')
{
    r1=r;
    printf("Non puoi inserire in questa colonna, reinserisci: ");
    scanf("%d",&r);
    while(r1==r)
    {
        printf("Non puoi inserire in questa colonna, reinserisci: ");
        scanf("%d",&r);
    }
}

```

Finito questo e quindi essendosi assicurati che la posizione fosse valida c'è un **ciclo for** che parte dal fondo della colonna a salire e tramite un if controlla la prima posizione valida e mette la pedina.

```
for(i=6;i>=0;i--)
{
    if(tabella[i][r]=='#')
    {
        tabella[i][r]='X';
        c=i;
        i=-1;
    }
    else
    {
    }
}
```

A questo punto lo schermo viene pulito con `system("cls")` e la griglia viene ristampata con la pedina al suo posto.

Vengono poi effettuati i **controlli di vincita** che prendono la posizione della pedina messa sia riga che colonna quindi `tabella[c][r]` e controllano se in tutte le direzioni ci sono altre 3 pedine che possono quindi formare un inseguirsi di 4 pedine che portano alla vittoria.

I controlli sono sia **orizzontali** che **verticali** che **obliqui**.

Se non c'è nessuna vincita ci sono i **controlli di pareggio** tramite due semplici cicli for che controlla tutta la tabella, in caso sia piena significa pareggio.

Il processo si ripete ugualmente per il **giocatore 2**.

```

# # # # # # #
# # # # # # #
# # # # # # #
# # # # # # #
O O O # # # #
X X X X # # #
-----
 0 1 2 3 4 5 6
Vittoria giocatore 1!
Premere per tornare al menu':
```



## Forza 4 PvE facile

Anche il forza 4 PvE in modalità facile (realizzato nella funzione `forza4_easybot()` ) come il tris PvE è stato molto semplice e veloce da realizzare.

Tutto quello che è servito è stato cambiare gli input del secondo giocatore con dei **random che andavano da 0 a 6** e cambiare i valori restituiti alla vittoria della CPU per i controlli del main.

```
//player2 turn  
r=rand()%6;
```

## Forza 4 PvE difficile

Il bot difficile del Forza 4 invece è stato molto più **faticoso e lungo** da programmare rispetto al bot difficile del tris. Ho dovuto fare prove su prove continuando a non ottenere i risultati che volevo. Ho dovuto riscrivere i **controlli** mille volte fino a che sono arrivato ad una soluzione che crea un bot che riesce a tenere una partita buona, anche se non ottima.

Andiamo ad analizzare il codice.

La parte iniziale e la parte di while comprendente il giocatore 1 è rimasta **invariata** dalle altre 2 modalità di Forza 4.

Quando arriva il turno del giocatore 2 invece ho optato per chiamare subito una funzione che è la funzione **`forza4_controlli_bot()`** che restituisce un intero.

```
//CPU turn  
ret=forza4_controlli_bot(c,r);
```

Nella funzione sono contenuti i controlli di **possibile vincita**. Questo significa che la funzione contiene tanti **IF** ed **ELSE IF** che guardano se con 1 o 2 mosse il giocatore 1 può vincere e in caso bloccano quelle mosse.

Prima di questi controlli però c'è un if che controlla se la riga in fondo della colonna dove va messa la pedina è uguale a X (mossa del giocatore 1). In caso lo sia controlla se anche quella sopra è uguale ad X e in caso lo sia mette una O nella posizione ancora più in alto.

Questo serve per bloccare le mosse in alto del giocatore ed è una delle **mosse più importanti** per evitare la vittoria per questo è messa prima degli IF.

```

int i=5;

if(tabella[i][r]=='X')
{
    if(tabella[i-1][r]=='X' && tabella[i-2][r]=='#')
    {
        tabella[i-2][r]='0';
        return 1;
    }
}

```

La funzione **restituisce 1 se entra in un IF**, quindi se fa una mossa, **sennò restituisce 0**.

Nel main dopo la chiamata della funzione c'è un IF che controlla il **valore restituito** dalla funzione. In caso sia 0 capisce che c'è una pedina da mettere e visto che non ci sono pericoli di vittoria da parte del giocatore 1 **randomizza** la posizione.

```

if(ret==0)
{
    r=rand()%6;
    if(tabella[0][r]=='X' || tabella[0][r]=='0')
    {
        r1=r;
        r=rand()%6;
        while(r1==r)
        {
            r=rand()%6;
        }
    }

    for(i=6;i>=0;i--)
    {
        if(tabella[i][r]=='#')
        {
            tabella[i][r]='0';
            c=i;
            i=-1;
        }
    }
}

```

Dopo ci sono ovviamente i **controlli di vittoria** per la CPU.

## Pensieri sul Forza 4

Tutte le modalità del Forza 4 sono state **più difficili** rispetto al tris da realizzare. Il PvP l'ho fatto come primo gioco, addirittura prima del tris e mi ha portato via un bel po' di tempo perché non è stato facile trovare subito la soluzione per il fattore gravità.

Per il bot difficile la parte complessa sono stati i **controlli nella funzione** `forza4_controlli_bot()`.

Comunque ho preferito scrivere il codice del forza 4 rispetto al tris perché mi sono dovuto impegnare di più e ho dovuto fare più ragionamenti.

## Battaglia Navale

### Battaglia Navale PvP

La battaglia navale è realizzata nella funzione **`battnavale()`**.

La battaglia navale è stata fin dall'inizio una sfida perché pur essendo un gioco con un regolamento e un procedimento semplice, realizzarla in C è più difficile di quanto si possa pensare.

Il primo problema è secondo me anche il più grande ed è come far **posizionare le navi** all'utente.

Come in tutti gli altri giochi ho iniziato riempiendo una matrice di 10X10 di '~' (meglio degli # perché son più simili ad onde che simboleggiano il mare) e stampandola.

Dopodiché si entra in un **while** che finisce soltanto quando tutte le navi disponibili sono state posizionate.

Sotto la griglia mare poi con un printf stampo tutte le navi presenti e l'utente sceglie quale posizionare.

```

 0  1  2  3  4  5  6  7  8  9
0 | ~ | ~ | ~ | ~ | ~ | ~ | ~ | ~ | ~ |
1 | ~ | ~ | ~ | ~ | ~ | ~ | ~ | ~ | ~ |
2 | ~ | ~ | ~ | ~ | ~ | ~ | ~ | ~ | ~ |
3 | ~ | ~ | ~ | ~ | ~ | ~ | ~ | ~ | ~ |
4 | ~ | ~ | ~ | ~ | ~ | ~ | ~ | ~ | ~ |
5 | ~ | ~ | ~ | ~ | ~ | ~ | ~ | ~ | ~ |
6 | ~ | ~ | ~ | ~ | ~ | ~ | ~ | ~ | ~ |
7 | ~ | ~ | ~ | ~ | ~ | ~ | ~ | ~ | ~ |
8 | ~ | ~ | ~ | ~ | ~ | ~ | ~ | ~ | ~ |
9 | ~ | ~ | ~ | ~ | ~ | ~ | ~ | ~ | ~ |

Navi presenti:
1- Portaerei (lunghezza 5): 1 (P)
2- Corazzata (lunghezza 4): 2 (C)
3- Sottomarino (lunghezza 3): 3 (S)
4- Incrociatore (lunghezza 2): 4 (I)
```

Dopo l'input c'è uno **switch** della scelta, il modo di inserire una nave è uguale per tutte, lo switch serve solo per le lunghezze della navi, i case sono quindi tutti uguali, cambiano solo i numeri.

Per controllare se il giocatore ha messo il numero massimo di navi di un tipo (ad esempio ha già messo una portaerei e quindi non può più scegliere 1) si usa un **flag** diverso per ogni nave (flagP, flagC ecc...).

Il flag è in verità un **contatore** che va al contrario, quando arriva a zero se la nave viene scelta si ritorna alla scelta grazie ad un if dentro al case che controlla il flag.

```
switch(navi)
{
    case 1:
        if (flagP!=0)
        {
```

Per posizionare la nave quindi ho scelto di far inserire **2 coordinate** e poi far scegliere se mettere la nave in **verticale o orizzontale** partendo da quelle 2 coordinate (verso il basso o verso destra).

```
do
{
    printf("Inserire posizione nave(piu a sinistra o piu in alto)\n");
    scanf("%d",&r);
    scanf("%d",&c);
}while(r>=10||c>=10||r<0||c<0);

do{
    printf("Come vuoi posizionare la nave? Orizzontale (1) o Verticale(2): ");
    scanf("%d",&orient);
}while(orient!=1 && orient!=2);
```

A questo punto c'è un altro switch tra verticale ed orizzontale.

In entrambi i casi viene controllato con un for se le posizioni dove la nave andrà messa sono libere.

```
case 1:
    for(i=0;i<5;i++)
    {
        if(mare[r][c+i]!=126){
            flagv=1;
        }
        if(c>5){
            flagv=1;
        }
    }
}
```

Se fuori dal for *flagv* è uguale a 0 allora la nave viene posizionata.

```

if(flagv==0)
{
    for (i=0;i<5;i++)
    {
        mare[r+i][c]='P';
    }

    flagP--;
}
else{
    goto posizioneP;
}
break;

```

Tutto questo si fa di nuovo fino a che flagv è uguale a 0.

Questo processo è uguale per tutti i tipi di nave.

Alla fine del while si controlla se tutti i flag sono uguali a 0 così che si può passare al turno del giocatore 2 che ovviamente è uguale.

```

//controllo
if (flagP==0 && flagC==0 && flagS==0 && flagI==0)
{
    flagf=1;
}

```

Dopo il turno del giocatore 2 è il momento degli *attacchi*, che vengono fatti in un altro while che finisce quando qualcuno vince.

Inizia il giocatore 1 inserendo le coordinate. Subito dopo c'è un *IF* che controlla se in queste coordinate c'è una nave, in caso ci sia allora trasforma quelle coordinate in '~' e stampa la frase: "Nave colpita!". Inoltre trasforma quelle coordinate in *X*, che simboleggia colpito, in un'altra matrice 10X10 che è la matrice attacco che è vuota e serve per ricordarsi di dove si ha colpito una nave.

```

if(marep2[r][c]=='P' || marep2[r][c]=='C' || marep2[r][c]=='S' || marep2[r][c]=='I'){
    marep2[r][c]=126;
    printf("Nave colpita!\n");
    mareatt1[r][c]='X';
}
else if(marep2[r][c]==126){
    printf("Nave mancata...\n");
}

```

I controlli della vincita sono fatti da due **cicli for** che percorrono la griglia e contano gli spazi uguali a '~'. Se arrivano a **100** significa che non ci sono più navi e la partita finisce.

La stessa identica cosa accade per il giocatore 2.

```
for(i=0;i<10;i++)
{
    for(j=0;j<10;j++)
    {
        if(marep2[i][j]==126){
            contp1++;
        }
    }
}

if(contp1==100){
    printf("\nVince Giocatore 1!!!\n"); win=1; print
}
else{
    contp1=0;
}
```

## Battaglia navale PvE facile

Lo stesso discorso che è stato fatto per Tris e Forza 4 si può fare qua, il codice è uguale e le mosse del giocatore 2 sono sostituite da **random** calibrati in base a cosa devono fare.

```
do
{
    r=rand()%10;
    c=rand()%10;
}while(r>=10 || c>=10 || r<0 || c<0);

do{
    orient=rand()%1+1;
}while(orient!=1 && orient!=2);
```

## Battaglia navale PvE difficile

Per la battaglia navale PvE modalità difficile ho trovato difficile la parte degli attacchi, perchè è importante fare degli **attacchi intelligenti**, che quando trovano una nave provino a colpire le parti vicine per assicurarsi da che parte sia orientata la nave.

L'inserimento della navi è fatto con un **random** e finisce quando tutte le navi sono state posizionate.

```
do
{
    r=rand()%10;
    c=rand()%10;
}while(r>=10||c>=10||r<0||c<0);
```

Gli attacchi invece non sono stati affatto semplici da realizzare.

Tutti gli attacchi avvengono come nelle altre modalità dentro ad un ciclo while e la parte del giocatore 1 rimane invariata.

Per la CPU invece ho utilizzato un flag chiamato **flag\_hit** che se ha valore 0 deve randomizzare se ha valore 1 significa che precedentemente ha colpito qualcosa e deve rimanere lì vicino.

```
if(flag_hit==0)
{
    printf("\nAttacco CPU: ");
    r=rand()%10;
    c=rand()%10;
    //printf("%d %d\n",r,c);
    if(mare[r][c]=='P' || mare[r][c]=='C' || mare[r][c]=='S' || mare[r][c]=='I')
    {
        mare[r][c]=126;
        printf("\nNave colpita dalla CPU!\n");
        mareatt2[r][c]='X';
        flag_hit=1;
        defr=r;
        defc=c;
    }
    else if(mare[r][c]==126){
        printf("Nave mancata...\n");
    }
}
```

Quando invece il flag\_hit è attivo significa che deve fare dei ragionamenti più complicati per continuare a colpire la nave.

Per farlo utilizza degli IF che controllano per **ogni lato** se c'è una nave. Quando la trova continua per quel lato grazie ad un flag che si attiva. Quando non ha più pezzi di nave su quel lato va nel lato contrario nell'eventualità che il primo pezzo di nave colpito non fosse un estremo.

Se non trova altri pezzi di nave capisce che è stata affondata e fa tornare 0 il flag\_hit.

```
if((mare[defr][defc+1]!=126 && defc+1<10) || flagdx==1)
{
    flagdx=1;
    if(mare[defr][defc+1]!=126)
    {
        mare[defr][defc+1]=126;
        mareatt2[defr][defc+1]='X';
        printf("\nNave colpita dalla CPU!");
        var=defc;
        defc=defc+1;
    }
    else if(mare[defr][defc-1]!=126)
    {
        mare[defr][var-1]=126;
        mareatt2[defr][var-1]='X';
        printf("\nNave colpita dalla CPU!");
        var=var-1;
    }
    else{
        flag_hit=0;
    }
}
```

Questo procedimento è uguale per ogni lato e cambiano solo le posizioni dei +1 e -1.

I controlli di vincita sono uguali a quelli della modalità PvP.

## Pensieri sulla battaglia navale

La battaglia navale è stato probabilmente il gioco **più difficile da programmare** in tutte e 3 le modalità, ho avuto tanti problemi sia nella modalità PvP che nel PvE difficile. Creare i controlli del bot non è stato affatto facile e mi ha occupato davvero tanto tempo.



## Conclusioni e pensieri finali sul progetto

Per quanto sia stato difficile, lungo e faticoso creare questa console di gioco mi è piaciuto davvero tanto, non avevo mai fatto un programma così lungo ed elaborato e contenente così tante possibilità diverse, e sono fiero di come sia uscito il programma finito.

Credo che l'idea di darci un progetto così impegnativo sia stata molto buona, tanto che ci ha portato via comunque tutto il secondo quadrimestre, principalmente nelle ore di laboratorio ma negli ultimi giorni e ultimi ritocchi al programma anche ore a casa, soprattutto nel scrivere questa relazione.

---

Alessandro Bettini

3 B Inf

Anno 22/23

Progetto di Informatica di fine anno scolastico