

# ALGORITMO PER LA SIMULAZIONE DI BATTAGLIA NAVALE

# INDICE

- 1. SCHEDA TECNICA**
- 2. STRUTTURA**
- 3. IL CAMPO DI GIOCO**
  - 3.1. Creazione**
  - 3.2. Gli stati di una cella**
  - 3.3. Azzeramento**
  - 3.4. Interfaccia grafica**
    - 3.4.1. Visuale assoluta*
    - 3.4.2. Visuale relativa*
- 4. LO SCHIERAMENTO DELLE NAVI**
  - 4.1. Il punto di riferimento**
    - 4.1.1. Il C.E del punto di riferimento*
  - 4.2. Il C.E del punto di riferimento**
  - 4.3. La fase di schieramento**
    - 4.3.1. Manuale*
    - 4.3.2. Casuale*
- 5. L'ATTACCO**
  - 5.1. Gestione dell'attacco**
    - 5.1.1. Il controllo dell'attacco*
    - 5.1.2. L'esito dell'attacco*
  - 5.2. Tipi di attacco**
    - 5.2.1. Attacco manuale*
    - 5.2.2. Attacco casuale*
    - 5.2.3. Attacco forzato*
- 6. LA FASE DI GIOCO**
  - 6.1. Visualizzazione del turno di gioco**
- 7. CONCLUSIONE**

## SCHEDA TECNICA

Le librerie utilizzate sono:

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

Costanti									
nome	K	X	Y	P	O	U	M	L	V
tipo	#define	#define	#define	#define	#define	#define	#define	#define	#define
uso	Dimensione massima del numero del giocatore	Dimensione delle colonne della matrice	Dimensione delle righe della matrice	Dimensione del sottomarino	Dimensione della torpediniera	Dimensione della lancia	Spazio totale occupato dalle navi	Considera solo gli 1 nella somma degli spazi	Considera solo i 4 nella somma degli spazi
valore	10	10	10	3	2	1	10	1	4

Variabili globali				
formato	nome	tipo	C.E	descrizione
char	lett[X]	input	'A''B''C''D''E''F' 'G''H''I''J'	Conversione coordinate numeriche in alfanumeriche
	Nick[K]		Tutti i caratteri supportati	
int	choice		1 o 2	Ognuna è utilizzata per tipi di scelte differenti
	moneta			
	scelta		Da 1 a 4	
	myCampo1[Y][X]	lavoro	Da 1 a 10 sia per le righe che per le colonne	Campo dell'utente
	myCampo2[Y][X]			Campo del bot
	campoIA1[Y][X]			
	campoIA2[Y][X]			
	cNave1		Da 0 a 1	Conta navi schierate
	cNave2		Da 0 a 2	
	cNave3		Da 0 a 3	
	i		Indeterminato	Indici dei cicli
	j			

Le funzioni adoperate sono:

- void azzeraCampo ( int [Y][X] );
- void visualeRelativa( int, int, int [Y][X]
- void visualeAssoluta ( int, int, int [Y][X] );
- void updateCampo ( int [Y][X], int [Y][X], int [Y][X] );
- void copiaCampo ( int [Y][X], int [Y][X] );
- int genCoordManX ( char [] );
- int genCoordManY ( char [] );
- int orientaNave ( int );
- int schieraNaveTop ( int, int, int, int, int [Y][X] );
- int schieraNaveDex ( int, int, int, int, int [Y][X] );
- int schieraNaveBot ( int, int, int, int, int [Y][X] );
- int schieraNaveSin ( int, int, int, int, int [Y][X] );
- void orientaNaveUpMan ( char [], int, int [Y][X] );
- void orientaNaveDexMan ( char [], int, int [Y][X] );
- void orientaNaveBotMan ( char [], int, int [Y][X] );
- void orientaNaveSinMan ( char [], int, int [Y][X] );
- void orientaNaveUp ( int, int [Y][X] );
- void orientaNaveDex ( int, int [Y][X] );
- void orientaNaveBot ( int, int [Y][X] );
- void orientaNaveSin ( int, int [Y][X] );
- int controllo ( int,int [Y][X] );
- void generaNave ( int, int [Y][X]
- int controlloAttacco ( int, int , int [Y][X] );
- void EsitoAttacco ( int, int, char [], int [Y][X] );
- void attacco ( char [], int [Y][X] );
- void autoAttacco ( char [], int [Y][X] );
- void affondaNave ( int [Y][X] );
- int attaccoForzato ( char [], int [Y][X] );

### LO SVILUPPO DELL'ALGORITMO

***L'algoritmo programmato simula battaglia navale sviluppando i seguenti punti:***

1. **Si creano quattro campi di gioco;** due sono quelli originari che prima vengono azzerati e poi riempiti con le navi in modo da essere copiati negli altri due.
2. **Si schierano le navi;** rispetto al nemico, che le dispone obbligatoriamente in modo automatico, il giocatore sceglierà:
  - 2.1. *lo schieramento casuale*, nelle seguenti modalità:
    - 2.1.1. Il programma genera un numero casuale compreso tra 1 e 4 per l'orientamento: 1.destra, 2.sinistra, 3.basso, 4.alto
    - 2.1.2. A seconda della direzione e della grandezza della nave il campo di esistenza delle coordinate del punto di riferimento da cui orientarla sarà differente ed il sistema continuerà a generare nuove coordinate fino a quando non vi appariranno
  - 2.2. *o lo schieramento manuale*,
    - 2.2.1. si decide l'orientamento della nave (1.verso l'alto, 2.verso destra, 3.verso il basso o 4.verso sinistra) rispetto ad un punto di riferimento il cui inserimento è richiesto successivamente
    - 2.2.2. poi si sceglie la nave da schierare e dopo ciò apparirà un avviso sulle navi rimaste in modo che l'utente non si confonda; ma, anche se dovesse accadere, vi è implementato un controllo che non consente lo schieramento di più navi di quelle preimpostate
    - 2.2.3. infine si inseriscono le coordinate del punto di riferimento da cui orientare la nave per lo schieramento (se non valide occorrerà reinserirle fino a quando non risulteranno corrette)
3. **Si copiano i campi con la disposizione delle navi negli altri due fin ora inutilizzati** solo dopo che, essendo già stati generati ed azzerati, vi siano state schierate tutte le navi.
4. **Si implementa la gestione delle visuali di gioco:** avendo quattro campi a disposizione, a coppie di due uguali fra loro, uno di ogni coppia rimarrà tale e quale mentre l'altro verrà aggiornato in base agli attacchi subiti in modo da consentire la corretta gestione delle visuali di gioco ad ogni turno:
  - 4.1. *Visuale di gioco assoluta:* visualizza il campo originario
  - 4.2. *Visuale di gioco relativa:* visualizza, in funzione degli attacchi inflitti o subiti, la copia del campo che verrà aggiornata ad ogni turno in risposta agli attacchi subiti dal campo in questione.

Il giocatore ad ogni turno potrà quindi avere una visuale di gioco completa vedendo tre campi:

  - 4.3. Il primo, che è del giocatore, visualizzato in funzione della visuale assoluta
  - 4.4. Il secondo, che è dello stesso, visualizzato in funzione della visuale relativa, come riferimento per gli attacchi subiti dal nemico sul proprio campo
  - 4.5. Il terzo, del nemico, visualizzato in funzione della visuale relativa come riferimento per gli attacchi inflitti.

5. **Si suddivide la partita in turni**, ognuno è da due mosse; la prima spetta al giocatore che ha vinto il “testa o croce” e l'altra all'avversario.

Le visuali di gioco implementate visualizzano quindi i tre campi, come descritto precedentemente, aggiornati dopo ogni mossa, migliorando l'esperienza di gioco dell'utente.

I turni sono resi “user friendly” dalla possibilità di visualizzare:

5.1. Il numero del turno

5.2. La visuale di gioco

5.3. Il giocatore che sferra la mossa

5.4. Una notifica in caso di attacco forzato

5.5. L'esito dell'attacco, in termini di coordinate e tipologia della cella colpita ( A5 acqua )

5.6. Una notifica sull'abbattimento di una nave che ne specifica il tipo in funzione delle celle da essa occupate, quindi in base alla sua grandezza

6. **Si procede all'attacco solo se la mossa è validata.**

Le coordinate del punto da attaccare vengono assegnate fin che non siano di una cella coperta, sia nel caso dell'attacco manuale che casuale, ma non in quello dell'attacco forzato

Nello specifico:

6.1. **L'attacco manuale:** è programmato in modo tale che l'utente inserisca le coordinate del punto da attaccare finché queste non risultino valide, così da non riattaccare due volte la stessa casella, per, solo dopo ciò, passare il turno al nemico

6.2. **L'attacco forzato:** è programmato in modo tale che vengano implementati all'algoritmo i ragionamenti logici applicati al gioco della battaglia navale. Il software è quindi reso intelligente riuscendo a riconoscere in base allo stato delle celle quelle di stato 3 passando in rassegna la matrice ed analizzando la prima tra queste rilevata da cui studiarne le celle adiacenti per effettuare la mossa:

6.2.1. Se sono di stato 2 allora vengono ignorate

6.2.2. Se al massimo una è di stato 3 allora considerando anche quelle adiacenti si procede attaccando per affondare un sottomarino

6.2.3. Se le celle adiacenti a quella di riferimento sono di stato 0 o 1 vengono considerate per l'attacco

6.3. **L'attacco casuale:** è programmato in modo tale che venga sferrato solo nel caso in cui non è attuabile l'attacco forzato. Fa sì che vengano generate casualmente le coordinate del punto da attaccare fin che non siano valide e quindi finché non vada a segno.

7. **Si effettua il controllo delle celle di stato 4 dopo ogni mossa:** se queste sono pari al numero iniziale di navi schierate allora termina la partita dichiarando come vincitore il giocatore che ha sferrato la mossa

## IL CAMPO DI GIOCO

### PRECISAZIONI

Il campo di gioco è dato da una griglia da dieci colonne e dieci righe ripartita in cento caselle; le coordinate di queste sono ricavabili proiettandole rispettivamente sul margine basso della griglia per quelle alfabetiche e sul margine sinistro della griglia per quelle numeriche

```

1 ? ? ? ? ? ? ? ? ? ?
2 ? ? ? ? ? ? ? ? ? ?
3 ? ? ? ? ? ? ? ? ? ?
4 ? ? ? ? ? ? ? ? ? ?
5 ? ? ? ? ? ? ? ? ? ?
6 ? ? ? ? ? ? ? ? ? ?
7 ? ? ? ? ? ? ? ? ? ?
8 ? ? ? ? ? ? ? ? ? ?
9 ? ? ? ? ? ? ? ? ? ?
10 ? ? ? ? ? ? ? ? ? ?
    A B C D E F G H I J

```

### CREAZIONE DEI CAMPI DI GIOCO

Sono impiegate 4 matrici:

```

int myCampo[Y][X];
int myCampo2[Y][X];
int campoIA[Y][X];
int campoIA2[Y][X];

```

Le matrici sono quindi da 10 righe e da 10 colonne: Y sono le righe e X le colonne;

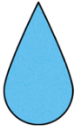




### GLI STATI DI UNA CASELLA

Una cella della matrice può assumere quattro stati:

- 0 corrisponde all'acqua non attaccata
- 1 corrisponde alla nave non attaccata
- 2 equivale all'acqua colpita
- 3 equivale alla nave colpita
- 4 corrisponde alla nave affondata

Ciò così che l'algoritmo identifichi le celle della mappa in base al loro stato per gestirle poi in modi specifici:

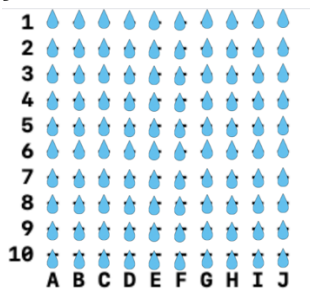
1. Quindi in primis nella creazione dei campi questi vengono azzerati, ovvero ad ogni loro cella viene assegnato uno 0 che la identifica come acqua coperta.
2. Durante lo schieramento delle navi le celle occupate da queste assumono valore 1 venendo quindi identificate come celle di navi coperte; dopodiché la mappa presenterà solo celle di stato 0 e 1.
3. Nella fase di gioco ad ogni turno si può attaccare solo una cella di stato 0 o 1 che muterà il proprio rispettivamente in:
  - a. 2 se si avrà colpito acqua
  - b. 3 se si avrà colpito una nave
  - c. 4 se la si avrà affondata

Acqua non colpita	Nave non colpita	Acqua colpita	Nave colpita	Nave affondata
				
Stato 0	Stato 1	Stato 2	Stato 3	Stato 4

## AZZERAMENTO DEL CAMPO DI GIOCO

Quindi nella creazione dei campi questi vengono azzerati con la funzione `void azzeracampo ( int [Y][X] )`;;, ovvero ad ogni elemento della matrice viene assegnato uno 0 che identifica la cella corrispondente come acqua non attaccata usando un ciclo for che scorre ogni elemento della matrice assegnando a ciascuno valore 0:

```
void azzeracampo ( int campo[Y][X] ){  
    int i;  
    int j;  
    for ( i = 0; i < X; i ++ ) {  
        for ( j = 0; j < Y; j ++ ){  
            campo[i][j] = 0;  
        }  
    }  
}
```



1	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹
2	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹
3	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹
4	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹
5	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹
6	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹
7	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹
8	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹
9	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹
10	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹	🔹
	A	B	C	D	E	F	G	H	I	J



## VISUALI DI GIOCO

*In base allo stato delle celle di un campo questo viene visualizzato diversamente a seconda della visuale usata:*

### 1. Visuale assoluta:

- Se la cella è di *stato 0* allora sarà rappresentata da un trattino (-) che simboleggia appunto le caselle di acqua
- Se la cella è di *stato 1* allora sarà rappresentata da un cerchio (O) che simboleggia appunto le caselle di navi

La procedura che gestisce la visuale assoluta è la seguente:

```
void visualeAssoluta ( int i, int j, int campo[Y][X] ) {  
    if ( campo[i][j] == 0 ) printf ( " -" );           //acqua  
    else if ( campo[i][j] == 1 ) printf ( " O" );       //colpita  
}
```

### 2. Visuale relativa

- Se la cella è di *stato 1 o 0* allora sarà rappresentata da un punto interrogativo (?)
- Se la cella è di *stato 2* allora sarà rappresentata da un trattino (-)
- Se la cella è di *stato 3* allora sarà rappresentata da un cerchio (O)
- Se la cella è di *stato 4* allora sarà rappresentata da una x (X)

La procedura che gestisce la visuale relativa è la seguente:

```
void visualeRelativa ( int i, int j, int campo[Y][X] ) {  
    if ( campo[i][j] == 2 ) printf ( " -" );           //acqua  
    else if ( campo[i][j] == 3 ) printf ( " O" );       //colpita  
    else if ( campo[i][j] == 4 ) printf ( " X" );       //affondata  
    else printf ( " ?" );                               //campo coperto  
}
```

- Il primo che è del giocatore e viene visualizzato in funzione della visuale assoluta
- Il secondo che è dello stesso è visualizzato in funzione della visuale relativa come riferimento per gli attacchi subiti dal nemico sul proprio campo
- Il terzo è del nemico ed è visualizzato in funzione della visuale relativa come riferimento per gli attacchi inflitti al nemico

La procedura che gestisce questa funzionalità è la seguente:

Pag. 10 di 38

DATI DI RIFERIMENTO SULLA CREAZIONE E LA VISUALIZZAZIONE DELLA MAPPA

Procedure e funzioni gestione matrici e grafica								
formato	nome	parametri						descrizione
			Par 1	Par 2	Par 3	Par 4	Par 5	
void	azzeraCampo	tipo	Int [] []					Assegna 0 a tutti gli elementi di una matrice
		nome	campo					
void	visualeAssoluta	tipo	int	int	Int [] []	<pre>1 - - - - - 2 - - - - - 3 0 - - - - 0 - - 4 - - 0 - 0 - 0 - - 5 - - - 0 - 0 - - 6 - - 0 - - - - - 7 - - 0 - - - - - 8 - - - - - 0 - - 9 - - - - - - - 10 - - - - - - -     A B C D E F G H I J</pre>		Visualizzazione della matrice: 0 come “ - “ 1 come “ O “
		nome	i	j	campo			
void	visualeRelativa	tipo	int	int	Int [] []	<pre>1 ? ? - ? ? ? - - 2 - - - ? - - - - 3 X ? - - - X - - 4 - ? X - X ? X ? - ? 5 - - - X - X - - 6 - ? 0 ? - ? - ? - 7 ? - ? - - ? - - 8 ? ? - - - X - ? 9 - - ? - ? - ? - 10 - - ? - - ? ? - ?     A B C D E F G H I J</pre>		Visualizzazione della matrice: 0 e 1 come “ ? “ 2 come “ - “ 3 come “ O “ 4 come “ X “
		nome	i	j	campo			
void	updateCampo	tipo	Int [] []	Int [] []	Int [] []			Visualizzazione in serie delle 3 matrici tale che: campo1 visuale assoluta Campo2 e campo 3 visuale relativa
		nome	campo1	campo2	campo3			
<div><pre>1 - - - - - 2 - - - - - 3 0 - - - - 0 - - 4 - - 0 - 0 - 0 - - 5 - - - 0 - 0 - - 6 - - 0 - - - - - 7 - - 0 - - - - - 8 - - - - - 0 - - 9 - - - - - - - 10 - - - - - - -     A B C D E F G H I J</pre></div> <div><pre>1 ? ? - ? ? ? - - 2 - - - ? - - - - 3 X ? - - - X - - 4 - ? X - X ? X ? - ? 5 - - - X - X - - 6 - ? 0 ? - ? - ? - 7 ? - ? - - ? - - 8 ? ? - - - X - ? 9 - - ? - ? - ? - 10 - - ? - - ? ? - ?     A B C D E F G H I J</pre></div> <div><pre>1 - ? - - ? - - - ? - 2 - ? - X ? ? ? - - 3 - X - - - ? ? ? X 4 - X ? ? - - - ? ? X 5 - - ? - ? - - ? - X 6 ? ? X - - - - ? - 7 - - - - ? ? - - ? 8 - - ? ? - - - ? - 9 ? - - - - ? - - 10 X ? - - - ? - -     A B C D E F G H I J</pre></div>								
void	copiaCampo	tipo	Int [] []	Int [] []				Copia gli elementi di campo1 in campo2
		nome	campo1	Campo2				

## LO SCHIERAMENTO DELLE NAVI

### COMMENTO

Le navi della flotta del giocatore sono personalizzabili sia in numero che in dimensione in quanto per lo spiegamento ci si avvale di due funzioni diverse a seconda del tipo (manuale o casuale):

- Casuale `void generaNave ()`;
- Manuale `void orientaNaveDexMan ()`;

Entrambe sono caratterizzate da due fattori:

- Ripetendo la funzione n volte si schierano n navi
- Agendo sui parametri si stabilisce la dimensione della nave in termini di celle occupate ed il campo in cui schierarla

Esemplificando:

- `generaNave( 3, myCampo )`; schiera casualmente una nave da tre nel campo del giocatore
- `orientaNaveDexMan ( lett, 1, campoIA )`; schiera una nave da uno nel campo nemico dati in input le coordinate del punto di riferimento per orientarla verso destra

**Nonostante sia possibile personalizzarne la quantità e la dimensioni di ognuna, in modo facile e veloce, si possono abbattere solo le navi che appartengano a quelle programmate per essere affondate.**

L'algoritmo della funzione `void affondaNave()`; riesce ad abbattere solo le navi concesse dai limiti dell'algoritmo.

La possibilità che, con sviluppi futuri, si riesca ad affondare qualsiasi nave non sta nel riprogrammare la funzione con un codice specifico per ogni caso, ma nel renderla generica, in grado quindi di affondare qualsiasi nave affrontando qualsiasi casistica.

**Per ovviare a quanto riportato, nonostante sia possibile personalizzare la flotta, ci si deve avvalere solo delle navi preimpostate e programmate ad essere affondate.**

### SPIEGAZIONE PROGRAMMA

**Con schieramento delle navi si intende l'assegnazione dello stato 1 agli elementi della matrice in modo tale che:**

- Gli elementi della matrice di valore 1 inizialmente siano in totale 10
- Gli stessi siano disposti in serie formando gruppi distinti:
  - o 1 gruppo da 3 celle adiacenti
  - o 2 gruppi da 2 celle adiacenti
  - o 3 gruppi da 1 cella adiacente
- Ogni gruppo sia distanziato da almeno una casella

Per schierare correttamente una nave ci sono 3 fattori da considerare:

- La dimensione
- La direzione
- Il punto di riferimento

nome	d	dir	x	y
tipo	input	input	input	input
formato	<b>int</b>	<b>int</b>	<b>int</b>	<b>int</b>
uso	dimensione	direzione	colonna	riga
range	1 a 3	1 a 4	0 a 9	0 a 9

La procedura per schierare una nave e che gestisce queste condizioni è specifica per il tipo di schieramento:

- Se lo schieramento è manualmente si richiama `void orientaNaveDirMan ()`;
- Se lo schieramento è automatico si richiama `void orientaNaveDir ()`;

Ci sono due possibili sviluppi per assegnare la disposizione di un gruppo di celle di stato 1 in un verso:

- Il primo prevede che l'inserimento delle coordinate del punto di riferimento avvenga precedentemente all'inserimento della direzione, implicando che, in base al punto scelto, si effettui un controllo atto a determinare le direzioni possibili verso cui orientare il dispiegamento della nave.
- Il secondo prevede che si prestabilisca prima la direzione verso cui orientare il gruppo di celle di stato 1 per poi stabilire in base alla direzione il range di validità delle coordinate del punto di riferimento in modo che di queste venga richiesto l'inserimento fino alla loro convalida

La metodologia applicata per farlo è la seconda e prevede che:

- Si stabilisca il parametro della dimensione della nave `void orientaNaveDir ( int d, int campo[Y][X] );`
- Si dichiari nella funzione una variabile `int dir;` e gli si assegni un valore corrispondente alla direzione che sia compreso tra 1.top 2.dex 3.bot 4.sin
- si identifichi un punto di riferimento nella matrice `campo` generando le coordinate ad esso associate tali che appartengano al C.E della matrice che è ristretto in funzione della dimensione, della direzione
- Se:
  - o le coordinate del punto di riferimento appartengono al C.E ristretto allora si procede con lo schieramento
  - o le coordinate del punto di riferimento non appartengono al C.E ristretto gli si riassegna un valore fino a quando non siano valide
- Assegnato correttamente il valore delle coordinate di riferimento si procede con l'assegnazione di stato 1 agli elementi considerati

#### IL PUNTO DI RIFERIMENTO ED IL C.E

1. Occorre poi generare le coordinate delle righe e delle colonne tali che siano comprese nei corrispettivi campi di esistenza della matrice:

$C.E_p : 0 \leq x \leq 9 \wedge 0 \leq y \leq 9, P(x;y)$

- o Per l'inserimento manuale


- Genera coordinata x inserendo da tastiera un valore alfanumerico

```
x = genCoordManX(lett);
int genCoordManX ( char lett[] ) {
    char c;
    int i;
    do {
        scanf ("fflush");
        printf ( "\t\tcoordinata
        alfanuemrica\n\t\t\ttx = " );
        scanf ( " %c", &c );
    }while ( ( c < 65 || c > 74 ) && ( c < 97 || c
    > 106 ) );
    if ( c > 97 || c < 106 ) {
        c = toupper(c);
    }
    for ( i = 0 ; i < X ; i ++ ) {
        if ( c == lett[i] ) break;
    }
    return i;
}
```

- ```
y = genCoordManY(lett);
int genCoordManY ( char lett[Y] ) {
    int y;
    do {
        printf ( "\t\tcoordinata
                numerica\n\t\tty = " );
        scanf ( "%d", &y );
    }while ( y < 1 || y > 10 );
    y = y - 1;
    return y;
}
```

- Per l'inserimento casuale:
  - Genera coordinata x  
`x = rand() % 10;`
  - Genera coordinata y  
`y = rand() % 10;`

|    |   |   |   |   |   |   |   |   |   |   |
|----|---|---|---|---|---|---|---|---|---|---|
| 1  | - | - | - | - | - | - | - | - | - | - |
| 2  | - | - | - | - | - | - | - | - | - | - |
| 3  | - | - | - | - | - | - | - | - | - | - |
| 4  | - | - | - | - | - | - | - | - | - | - |
| 5  | - | - | P | - | - | - | - | - | - | - |
| 6  | - | - | - | - | - | - | - | - | - | - |
| 7  | - | - | - | - | - | - | - | - | - | - |
| 8  | - | - | - | - | - | - | - | - | - | - |
| 9  | - | - | - | - | - | - | - | - | - | - |
| 10 | - | - | - | - | - | - | - | - | - | - |
|    | A | B | C | D | E | F | G | H | I | J |

|                                                                                    |
|------------------------------------------------------------------------------------|
| Punto di riferimento                                                               |
|  |
| $P(x; y)$                                                                          |

Il punto di riferimento in sé non è sufficiente allo schieramento delle navi

### VERIFICA DEL C.E RISTRETTO DEL PUNTO DI RIFERIMENTO

Occorre anche la verifica dell'appartenenza al C.E ristretto dai fattori dimensione e direzione delle coordinate che è gestita dalla funzione: `int controlloSchieraDir();`

Questo controllo permette il posizionamento del gruppo di elementi di stato 1 solo se l'elemento associato ad ognuno esiste ed è di stato 0 e ad esso adiacenti vi siano solo celle di stato 0.






La funzione applica le seguenti istruzioni per effettuare il controllo delle coordinate del punto di riferimento da cui assegnare stato 1 agli elementi della matrice considerati:

- Prestabilita la direzione
- Prestabilito il numero degli elementi adiacenti a cui assegnare stato 1
- Prestabilite le coordinate di un elemento della matrice
- Si dichiara una variabile di lavoro: `int errore;`. usata come variabile di ritorno e che restituisca:
  - o `errore = 1;` se la verifica delle coordinate ha dato esito positivo gli elementi considerati assumono stato 1
  - o `errore = 0;` se lo schieramento delle navi ha dato esito negativo si riassegnano i valori alle coordinate

Il ragionamento che permette ciò, dato un gruppo di elementi di stato 1 qualsiasi, è applicato nel seguente modo:

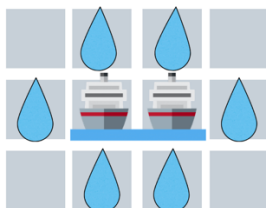
- In base al fatto che i punti adiacenti ad un altro possono essere compresi tra:

$$x_1 = x \pm 1 \wedge y_1 = y \pm 1$$

| rispetto al punto di riferimento                                                  | P1 adiacente al margine alto<br>dir = 1;                                          | P2 adiacente al margine destro<br>dir = 2;                                        | P3 adiacente al margine basso<br>dir = 3;                                           | P4 adiacente al margine sinistro<br>dir = 4;                                        |
|-----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
|  |  |  |  |  |
| $P(x; y)$                                                                         | $P_1(x; y + 1)$                                                                   | $P_2(x + 1; y)$                                                                   | $P_3(x; y - 1)$                                                                     | $P_4(x - 1; y)$                                                                     |

Se la dimensione della nave fosse  $d = 2$ ; ed il punto di riferimento tale che  $\text{campo}[y][x] = 0$ ; e la direzione in cui la nave in cui è orientata rispetto ad esso tale che  $\text{dir} = 2$ ; (destra) allora risulterebbe che gli elementi della matrice ad assumere stato 1 sarebbero  $\text{campo}[y][x] = 1$ ; e  $\text{campo}[y - 1][x]$ ; che dovrebbero esistere ed essere uguali a 0 prima dell'assegnazione. Generalizzando data la grandezza o dimensione della nave, ovvero il numero del gruppo di elementi adiacenti della matrice che devono assumere stato 1, si può automatizzare il meccanismo con un ciclo for come in questo:

```
for ( i = 0; i < 2; i ++ ) {
    campo[y][x] = 1;
    x++;
}
```

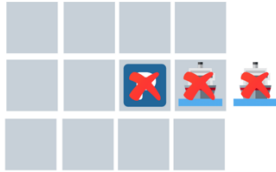


- Quindi le navi possono essere schierate nelle quattro dimensioni spaziali con un ciclo for avente come limite la loro dimensione e incrementando o decrementando una delle due coordinate ad ogni ciclo in base alla direzione:
  - o Nello schieramento verso l'alto:  $y--$ ;
  - o Nello schieramento verso il basso:  $y++$ ;
  - o Nello schieramento verso schieramento verso destra:  $x++$ ;
  - o Nello schieramento avviene verso sinistra:  $x--$ ;

Detto ciò è necessario considerare le celle che dovrebbero assumere stato 1 e quelle ad esse adiacenti che devono essere tutte uguali a 0 per lo schieramento della nave:

Caso 1:

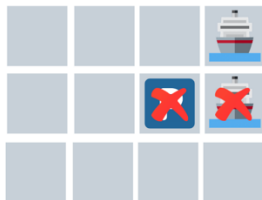
- Stabilito  $d = 3$ ;
- Stabilito  $dir = 2$ ;
- Stabilite le coordinate del punto di riferimento



La nave in questo caso non può essere schierata verso destra in quanto per il punto di riferimento da cui è orientata esce fuori dai bordi

Caso 2:

- Stabilito  $d = 2$ ;
- Stabilito  $dir = 2$ ;
- stabilito il punto di riferimento



La nave in questo caso non può essere schierata verso destra in quanto per la posizione del punto di riferimento sarebbe adiacente ad una nave già disposta

Il controllo atto ad effettuare il corretto schieramento è quindi svolto dalla funzione:

```
int controlloSchieraDir ();
```

Questa è annidata nella funzione **void** orientaNaveDirMan(); e **void** orientaNaveDir(); del relativo verso di schieramento in un do-while che la ha come condizione e in cui come istruzioni vi sono l'assegnazione dei valori delle coordinate del punto di riferimento.

La variabile di ritorno della funzione **int** controlloSchieraDir(); deve essere pari a 0 affinché l'inserimento delle coordinate venga convalidato al fine di interrompere nel do-while

Prima di elencare i controlli in questione occorre spiegare le variabili usate:

| nome    | x          | y          | errore     |
|---------|------------|------------|------------|
| formato | <b>int</b> | <b>int</b> | <b>int</b> |
| tipo    | input      | input      | output     |
| uso     | colonna    | riga       | controllo  |
| Range   | 0 a 9      | 0 a 9      | 0 o 1      |



- Per lo schieramento verso l'alto

```
int controlloSchieraTop ( int d, int x, int y, int yo, int
campo[Y][X] ) {
    int errore = 0;
    int i;
    for( i = 0; i < d; i ++ ) {
        if(campo[y][x + 1] == 0 && campo[y][x - 1] == 0 && campo[y][x]
        == 0 && campo[yo + 1][x - 1] == 0 && campo[yo + 1][x] == 0 &&
        campo[yo + 1][x + 1] == 0 && campo[yo - d][x - 1] == 0 &&
        campo[yo - d][x] == 0 && campo[yo - d][x + 1] == 0 ) campo[y][x]
        = 1;
        else{
            errore = 1;
            break;
        }
        y--;
    }
    return errore;
}
```

- Per lo schieramento verso il basso

```
int controlloSchieraBot ( int d, int x, int y, int yo, int
campo[Y][X] ) {
    int errore = 0;
    int i;
    for ( i = y; i < d + y; i ++ ) {
        if ( campo[i][x + 1] == 0 && campo[i][x] == 0 && campo[i][x -
        1] == 0 && campo[yo - 1][x - 1] == 0 && campo[yo - 1][x] == 0
        && campo[yo - 1][x + 1] == 0 && campo[yo + d][x - 1] == 0 &&
        campo[yo + d][x] == 0 && campo[yo + d][x + 1] == 0 ) campo[i][x]
        = 1;
        else{
            errore = 1;
            break;
        }
    }
    return errore;
}
```

- Per lo schieramento verso destra

```
int controlloSchieraDex ( int d, int x, int y, int xo, int
campo[Y][X] ) {
    int errore = 0;
    int i;
    for ( i = x; i < d + x; i ++ ) {
        if ( campo[y + 1][i] == 0 && campo[y][i] == 0 && campo[y - 1][i]
        == 0 && campo[y - 1][xo - 1] == 0 && campo[y][xo - 1] == 0 &&
        campo[y + 1][xo - 1] == 0 && campo[y - 1][xo + d] == 0 &&
        campo[y][xo + d] == 0 && campo[y + 1][xo + d] == 0 ) campo[y][i]
        =1;
        else{
            errore = 1;
            break;
        }
    }
    return errore;
}
```

- Per lo schieramento verso sinistra

```
int controlloSchieraSin ( int d, int x, int y, int xo, int
campo[Y][X] ) {
    int errore = 0;
    int i;
    for ( i = 0; i < d; i ++ ) {
        errore = 0;
        if ( campo[y + 1][x] == 0 && campo[y][x] == 0 && campo[y - 1][x]
== 0 && campo[y + 1][xo + 1] == 0 && campo[y][xo + 1] == 0 &&
campo[y - 1][xo + 1] == 0 && campo[y - 1][xo - d] == 0 &&
campo[y][xo - d] == 0 && campo[y + 1][xo - d] == 0 ) campo[y][x]
= 1;
        else{
            errore = 1;
            break;
        }
        x--;
    }
    return errore;
}
```

## FASE DI SCHIERAMENTO DELLE NAVI

2. Analizzate queste casistiche è possibile quindi procedere alla programmazione dello schieramento manuale e casuale delle navi, ma prima di ciò, è necessario spiegare le variabili utilizzate:

| Nome    | x        | y     | xo            | yo         | dir       | d          | errore                 |
|---------|----------|-------|---------------|------------|-----------|------------|------------------------|
| formato | int      | int   | int           | int        | int       | int        | int                    |
| Tipo    | di input | input | lavoro        | lavoro     | input     | input      | output                 |
| Uso     | colonna  | riga  | Salva colonna | Salva riga | direzione | Dimensione | Controllo schieramento |
| Range   | 0 a 9    | 0 a 9 | 0 a 9         | 0 a 9      | 1 a 4     | 1 a 3      | 0 a 1                  |

## SCHIERAMENTO MANUALE

3. Rispettivamente per lo schieramento manuale nella direzione inserita dall'utente si applica la relativa funzione di schieramento con uno switch avente come parametro dir ed un caso per ogni direzione in cui vi è richiamata la funzione del relativo tipo di schieramento.

- Dir = 1 Schieramento verso l'alto: C.E(nave):  $y < d - 1$

```
void orientaNaveUpMan ( char lett[], int d, int
campo[Y][X] ) {
    int x;
    int y;
    int yo;
    do{
        x = genCoordManX(lett);
        do {
            y = genCoordManY(lett);
            yo = y;
        }while ( y < d - 1 );
    }while ( controlloSchieraTop ( d, x, y, yo,
campo ) != 0 );
}
```

Ogni diversa funzione di schieramento è distinta dall'altra solo dalla condizione in cui si verifica l'appartenenza delle coordinate al campo di esistenza della nave nel do-while annidato e dalla variabile in esso assegnata (x o y)

- Dir = 2 Schieramento verso destra:  

```
void orientaNaveDexMan ();
do {
    x = genCoordManX(lett);
    xo = x;
}while( x > X - d );
```
- Dir = 3 Schieramento verso il basso:  

```
void orientaNaveBotMan ();
do{
    y = genCoordManY ( lett );
    yo = y;
}while( y > Y - d );
```
- Dir = 4 Schieramento verso sinistra:  

```
void orientaNaveSinMan ();
do{
    x = genCoordManX(lett);
    xo = x;
}while ( x < d - 1 );
```

## SCHIERAMENTO CASUALE

---

4. Per lo schieramento casuale si richiama la funzione `void generaNave ( )`; che con uno switch avente come parametro la direzione ed un caso per ogni verso prima genera casualmente la direzione e poi in base al suo valore esegue le istruzioni del caso relativo da cui si richiama la funzione di schieramento specifica:

```
void generaNave ( int d, int campo[Y][X] ) {
    int dir;
    dir = 1 + rand() % 4;
    if ( dir == 1 ) {
        orientaNaveUp(d,campo);
    }
    else if ( dir == 2 ) {
        orientaNaveDex(d,campo);
    }
    else if ( dir == 3 ) {
        orientaNaveBot(d,campo);
    }
    else{
        orientaNaveSin(d,campo);
    }
};
}
```

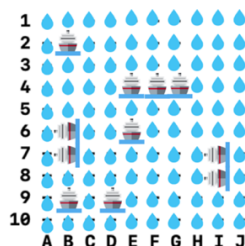
L'unica differenza tra lo schieramento manuale e quello casuale è che nel primo la direzione e le coordinate sono inserite da tastiera mentre nel secondo caso la direzione e le coordinate x e y vengono assegnate casualmente tramite la funzione: `srand ( time ( NULL ) )`;

- `dir = 1 + rand() % 4;`
- `y = rand() % 10;`
- `x = rand() % 10;`

## RIEPILOGO

Riassumendo, per il corretto schieramento delle navi, le coordinate del punto di riferimento continueranno ad esse generate fino a quando non appariranno entrambe al C.E ristretto del punto di riferimento in funzione della direzione e della dimensione.

|               | dir = 1 = alto |             | dir = 2 = destra |             | dir = 3 = basso |       | dir = 4 = sinistra |       |
|---------------|----------------|-------------|------------------|-------------|-----------------|-------|--------------------|-------|
| nome          | x              | y           | x                | y           | x               | y     | x                  | y     |
| C.E ristretto | 0 a 9          | $y < d - 1$ | 0 a 9            | $y > Y - d$ | $x < d - 1$     | 0 a 9 | $x > X - d$        | 0 a 9 |



## CONSIDERAZIONI

Durante la programmazione in questa fase di completamento dello schieramento originariamente si verificava un errore non identificato consistente nell'assegnazione di stato 1 agli elementi della matrice che non dovevano essere considerati navi. Ciò implicava che le caselle di stato 1 nel campo erano più di quelle prestabilite, ovvero più di 10.

Per arginare questo problema senza risolverlo direttamente è stata introdotta la funzione `int controllo ( int h, int campo[Y][X] );` come condizione in un do-while che debba risultare uguale a 1 affinché non ricominci lo schieramento delle navi dall'inizio azzerando nuovamente il campo.

Questa funzione affinché restituisca un valore pari a 0 o 1 effettua un controllo per verificare che le navi schierate e quindi le caselle di stato 1 siano effettivamente quelle che devono essere nelle seguenti modalità:

1. passato come parametro h corrispondente al numero di elementi totali di stato 1 nella matrice dopo lo schieramento
2. dichiarata e azzerata la variabile contatore `int k = 0;`
3. la funzione rassegna la matrice con un for e conta le celle di stato 1 incrementando la variabile contatore ogni volta che ne riscontra una `if ( campo[i][j] == h ) k++;` per poi confrontarla con il numero prestabilito in modo che:
  - a. se dal confronto risultino diverse allora ritorni `errore = 1;`
  - b. Se dal confronto risultino uguali allora ritorni `errore = 0;`

Nello specifico:

```
int controllo ( int h, int campo[Y][X] ) {
    int i;
    int j;
    int k;
    int errore;
    errore=0;
    k=0;
    for ( i = 0; i < X; i ++ ) {
        for ( j = 0; j < Y; j ++ ) {
            if ( campo[i][j] == h ) k++;
        }
    }
    if ( k != M ) errore = 1;
    return errore;
}
```

## DATI DI RIFERIMENTO PER LO SCHIERAMENTO DELLE NAVI

Ricapitolando la fase di schieramento delle navi:

- le funzioni utilizzate sono:
  - Relative all'assegnamento di un valore alle coordinate compreso nel C.E della matrice

```
int genCoordManX ( char [] );
int genCoordManY ( char [] );
```

- Relative al controllo dei valori assunti dalle coordinate in funzione del C.E ristretto

```
int controlloSchieraTop ();
int controlloSchieraDex ();
int controlloSchieraBot ();
int controlloSchieraSin ();
```

- Relative allo schieramento manuale

```
void orientaNaveUpMan ();
void orientaNaveDexMan ();
void orientaNaveBotMan ();
void orientaNaveSinMan ();
```

- Relative allo schieramento casuale

```
void orientaNaveUp ();
void orientaNaveDex ();
void orientaNaveBot ();
void orientaNaveSin ();
void generaNave ();
```

- Relative al controllo di fine schieramento del numero di navi disposte

```
int controllo ();
```

- le variabili utilizzate sono:

| Nome    | x        | y     | xo            | yo         | dir       | d          | errore                 |
|---------|----------|-------|---------------|------------|-----------|------------|------------------------|
| formato | int      | int   | int           | int        | int       | int        | int                    |
| Tipo    | di input | input | lavoro        | lavoro     | input     | input      | output                 |
| Uso     | colonna  | riga  | Salva colonna | Salva riga | direzione | Dimensione | Controllo schieramento |
| Range   | 0 a 9    | 0 a 9 | 0 a 9         | 0 a 9      | 1 a 4     | 0 a 3      | 0 a 1                  |

| Procedure e funzioni schieramento navi |                     |      |           |           |           |      |                                   |                                                                                                                                   |                                               |                     |  |  |  |                                               |
|----------------------------------------|---------------------|------|-----------|-----------|-----------|------|-----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------|---------------------|--|--|--|-----------------------------------------------|
| formato                                | nome                |      | parametri |           |           |      |                                   | descrizione                                                                                                                       |                                               |                     |  |  |  |                                               |
|                                        |                     |      | Par1      | Par2      | Par3      | Par4 | Par5                              |                                                                                                                                   |                                               |                     |  |  |  |                                               |
| int                                    | genCoordManX        | tipo | char []   |           |           |      |                                   |                                                                                                                                   | Assegna ad x un valore tale che $x \in [0,9]$ |                     |  |  |  |                                               |
|                                        |                     | nome | x         |           |           |      |                                   |                                                                                                                                   |                                               |                     |  |  |  |                                               |
| int                                    | genCoordManX        | tipo | char []   |           |           |      |                                   |                                                                                                                                   |                                               |                     |  |  |  | Assegna ad y un valore tale che $y \in [0,9]$ |
|                                        |                     | nome | y         |           |           |      |                                   |                                                                                                                                   |                                               |                     |  |  |  |                                               |
| int                                    | controlloSchieraTop | tipo | int       | int       | int       | int  | Int [] []                         | Verifica se la nave rispetto a $P(x,y)$ può essere schierata Rispettivamente verso:<br>l'alto<br>il basso<br>destra<br>o sinistra |                                               |                     |  |  |  |                                               |
|                                        |                     | nome | d         | x         | y         | Yo   | matrice                           |                                                                                                                                   |                                               |                     |  |  |  |                                               |
| int                                    | controlloSchieraBot |      |           |           |           |      |                                   |                                                                                                                                   | controlloSchieraDex                           | controlloSchieraSin |  |  |  |                                               |
|                                        |                     |      |           |           |           |      |                                   |                                                                                                                                   |                                               |                     |  |  |  |                                               |
|                                        |                     |      |           |           |           |      |                                   |                                                                                                                                   |                                               |                     |  |  |  |                                               |
| void                                   | orientaNaveUpMan    | tipo | char []   | int       | Int [] [] |      |                                   |                                                                                                                                   | Dispone nave solo se: $y < d - 1$             |                     |  |  |  |                                               |
|                                        |                     | nome | lett      | d         | matrice   |      |                                   |                                                                                                                                   |                                               |                     |  |  |  |                                               |
| void                                   | orientaNaveBot      | tipo | int       | Int [] [] |           |      | Dispone nave solo se: $x < d - 1$ |                                                                                                                                   |                                               |                     |  |  |  |                                               |
|                                        |                     | nome | d         | matrice   |           |      |                                   |                                                                                                                                   |                                               |                     |  |  |  |                                               |
| void                                   | orientaNaveBotMan   | tipo | char []   | int       | Int [] [] |      |                                   |                                                                                                                                   | Dispone nave solo se: $y > Y - d$             |                     |  |  |  |                                               |
|                                        |                     | nome | lett      | d         | matrice   |      |                                   |                                                                                                                                   |                                               |                     |  |  |  |                                               |
| void                                   | orientaNaveBot      | tipo | int       | Int [] [] |           |      | Dispone nave solo se: $y > Y - d$ |                                                                                                                                   |                                               |                     |  |  |  |                                               |
|                                        |                     | nome | d         | matrice   |           |      |                                   |                                                                                                                                   |                                               |                     |  |  |  |                                               |
| void                                   | orientaNaveDexMan   | tipo | char []   | int       | Int [] [] |      |                                   |                                                                                                                                   | Dispone nave solo se: $y > Y - d$             |                     |  |  |  |                                               |
|                                        |                     | nome | lett      | d         | matrice   |      |                                   |                                                                                                                                   |                                               |                     |  |  |  |                                               |
| void                                   | orientaNaveDex      | tipo | int       | Int [] [] |           |      | Dispone nave solo se: $y > Y - d$ |                                                                                                                                   |                                               |                     |  |  |  |                                               |
|                                        |                     | nome | d         | matrice   |           |      |                                   |                                                                                                                                   |                                               |                     |  |  |  |                                               |

|      |                   |      |         |           |           |  |                                        |
|------|-------------------|------|---------|-----------|-----------|--|----------------------------------------|
| void | orientaNaveSinMan | tipo | char [] | int       | Int [] [] |  | Dispone nave solo se: x>X-d            |
|      |                   | nome | lett    | d         | matrice   |  |                                        |
| void | orientaNaveSin    | tipo | int     | Int [] [] |           |  |                                        |
|      |                   | nome | d       | matrice   |           |  |                                        |
| void | generaNave        | tipo | int     | Int [] [] |           |  | In base alla direzione schiera la nave |
|      |                   | nome | d       | matrice   |           |  |                                        |

## L'ATTACCO

### *Le regole di tiro di battaglia navale*

Un attacco è conforme e quindi eseguibile solo se attacca una cella coperta

## CONTROLLO DELL'ATTACCO

Questa procedura è utilizzata come condizione in un do-while nelle funzioni di `attacco()`; e `autoAttacco()`;

Dichiarata in essa una variabile di output, se è attaccato un punto già colpito (quindi che non sia di stato 0 o 1), questa assumerà valore uno, implicando che debbano essere reinserite le coordinate del punto fino a quando la cella colpita non risulterà di stato 0 o 1 perché solo se la variabile di ritorno è pari a 0, e quindi solo se il controllo ha dato esito positivo, allora esce dal ciclo.

```
int controlloAttacco ( int x, int y, int campo[Y][X] ) {  
    int errore;  
    errore=0;  
    if ( campo[y][x] != 0 && campo[y][x] != 1 ) errore = 1;  
    return errore;  
}
```

## ESITO DELL'ATTACCO

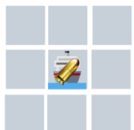

Se `controlloAttacco()` ha dato esito positivo, prima si assegna stato 2 (acqua colpita) o 3 (nave colpita) alla cella attaccata, in base a se era rispettivamente di stato 0 (acqua) o 1 (nave) e poi si procede visualizzando l'esito dell'attacco in funzione delle coordinate e del tipo di cella colpita (A6 nave)

```
void esitoAttacco ( int x, int y, char lett[], int campo[Y][X] ) {  
    if ( campo[y][x] == 0 ) {  
        campo[y][x] = 2;  
        printf ( "\n\n\tL'attacco è andato a segno\n\t\tAcqua nel punto %d%c\n", y + 1,  
lett[x] );  
    }  
    else if ( campo[y][x] == 1 ) {  
        campo[y][x] = 3;  
        printf ( "\n\n\tL'attacco è andato a segno\n\t\tNave colpita nel punto %d%c\n",  
y + 1, lett[x] );  
    }  
}
```



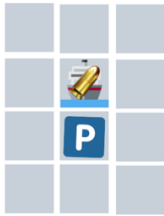
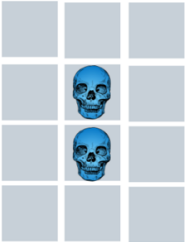
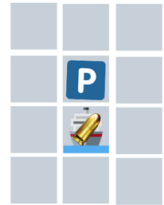
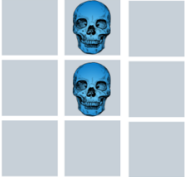


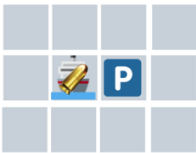

## AFFONDA NAVE

Se `controllaAttacco()`; ha dato esito positivo allora dopo aver richiamato la procedura `esitoAttacco()`; rassegna la matrice con un ciclo for che identifica le celle di stato 3, quindi le navi colpite ma non affondate. Ogni volta che identifica un punto di riferimento corrispondente ad una cella di stato 3 controlla:

| Navi da una casella                                                                                                                                                                                                                                                                                                             |                                                                                      |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|
| <i>Se adiacenti non vi sono né caselle di stato 1 né caselle di stato 3 allora si procede all'abbattimento della nave identificata come da una cella assegnando a questa stato 4</i>                                                                                                                                            |                                                                                      |
| Identifica                                                                                                                                                                                                                                                                                                                      | Affonda                                                                              |
|                                                                                                                                                                                                                                               |  |
| <pre> if ( ( campo[i + 1][j] == 0    campo[i + 1][j] == 2 ) &amp;&amp; ( campo[i - 1][j] == 0    campo[i - 1][j] == 2 ) &amp;&amp; ( campo[i][j - 1] == 0    campo[i][j - 1] == 2 ) &amp;&amp; ( campo[i][j + 1] == 0    campo[i][j + 1] == 2 ) ) {     campo[i][j] = 4;     printf ( "\n\n\tLancia affondata!\n\n" ); } </pre> |                                                                                      |

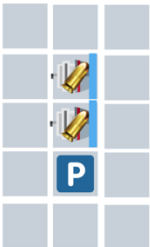

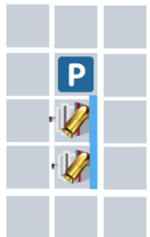

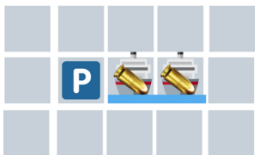

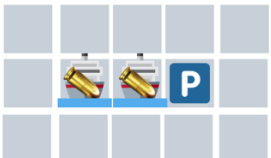

### Navi da due caselle

Se, in base alla direzione, vi è un'altra casella di stato 3 adiacente a quella di riferimento, si procede all'abbattimento della nave identificata come da due celle assegnando a queste stato 4

|                                                                                                                                                                                                                                                                                           | Identifica                                                                            | Affonda                                                                               |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| <b>Alto</b><br><pre> if ( campo[i + 1][j] == 3 &amp;&amp; (campo[i + 2][j] == 0    campo[i + 2][j] == 2 ) &amp;&amp; ( campo[i - 1][j] == 0    campo[i - 1][j] == 2 ) ) {     campo[i][j] = 4;     campo[i + 1][j] = 4;     printf ( "\n\n\tTorpediniere affondato!\n\n" ); } </pre>      |    |   |
| <b>Basso</b><br><pre> if(campo[i - 1][j] == 3 &amp;&amp; (campo[i - 2][j] == 0    campo[i - 2][j] == 2 ) &amp;&amp; ( campo[i + 1][j] == 0    campo[i + 1][j] == 2 ) ) {     campo[i][j] = 4;     campo[i - 1][j] = 4;     printf ( "\n\n\tTorpediniere affondato!\n\n" ); } </pre>       |   |   |
| <b>Destra</b><br><pre> if ( campo[i][j + 1] == 3 &amp;&amp; (campo[i][j + 2] == 0    campo[i][j + 2] == 2 ) &amp;&amp; ( campo[i][j - 1] == 0    campo[i][j - 1] == 2 ) ) {     campo[i][j] = 4;     campo[i][j + 1] = 4;     printf ( "\n\n\tTorpediniere affondato!\n\n" ); } </pre>    |  |  |
| <b>Sinistra</b><br><pre> if ( campo[i][j - 1] == 3 &amp;&amp; ( campo[i][j - 2] == 0    campo[i][j - 2] == 2 ) &amp;&amp; ( campo[i][j + 1] == 0    campo[i][j + 1] == 2 ) ) {     campo[i][j] = 4;     campo[i][j - 1] = 4;     printf ( "\n\n\tTorpediniere affondato!\n\n" ); } </pre> |  |  |

### Navi da tre caselle

Se, in base alla direzione, vi sono due celle di stato 3 adiacenti a quella di riferimento allora si procede all'abbattimento della nave identificata come da tre celle assegnando a queste stato 4

|                                                                                                                                                                                                                                                                                           | Identifica                                                                           | abbatti                                                                               |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| <b>Alto</b><br><pre> if ( campo[i + 1][j] == 3 &amp;&amp; campo[i + 2][j] == 3 ) {     campo[i][j] = 4;     campo[i + 1][j] = 4;     campo[i + 2][j] = 4;     printf ( "\n\n\tSottomarino affondato!\n\n" ); } </pre>                                                                     |   |   |
| <b>Basso</b><br><pre> if ( campo[i - 1][j] == 3 &amp;&amp; campo[i + 2][j] == 3 ) {     campo[i][j] = 4;     campo[i - 1][j] = 4;     campo[i - 2][j] = 4;     printf ( "\n\n\tSottomarino affondato!\n\n" ); } </pre>                                                                    |  |   |
| <b>Destra</b><br><pre> if ( campo[i][j + 1] == 3 &amp;&amp; campo[i][j + 2] == 3 ) {     campo[i][j] = 4;     campo[i][j + 1] = 4;     campo[i][j + 2] = 4;     printf ( "\n\n\tSottomarino affondato!\n\n" ); } </pre>                                                                   |  |  |
| <b>Sinistra</b><br><pre> if ( campo[i][j - 1] == 3 &amp;&amp; ( campo[i][j - 2] == 0    campo[i][j - 2] == 2 ) &amp;&amp; ( campo[i][j + 1] == 0    campo[i][j + 1] == 2 ) ) {     campo[i][j] = 4;     campo[i][j - 1] = 4;     printf ( "\n\n\tTorpediniere affondato!\n\n" ); } </pre> |  |  |

## L'ATTACCO MANUALE

L'attacco, affinché venga sferrato, deve essere possibile e ciò dipende dalle coordinate inserite dall'utente; se l'attacco è conforme alle regole di tiro allora andrà a segno altrimenti farà cilecca e il giocatore in offensiva dovrà ritentare fino a quando non andrà a segno reinserendo nuovi valori delle coordinate.

```
void attacco ( char lett[X], int campo[Y][X] ) {
    int x;
    int y;
    printf ( "\n\n\tInserire coordinate:\n" );
    do{
        x = genCoordManX(lett);
        y = genCoordManY(lett);
        if ( controlloAttacco ( x, y, campo ) != 0 ) printf( "\nCilecca!\n\n\tReinserire
coordinate:\n" );
        else{
            esitoAttacco(x,y,lett,campo);
            break;
        }
    }while ( controlloAttacco(x,y,campo) != 0 );
}
```

## L'ATTACCO CASUALE

L'attacco casuale della funzione `void autoAttacco()`; è l'analogo dell'attacco manuale ma differisce da esso per l'assegnazione di valori casuali alle coordinate con la funzione

```
srand ( time ( NULL ) );
    o y = rand() % 10;
    o x = rand() % 10;
```

## L'ATTACCO FORZATO

Questo tipo di attacco implementa al software i ragionamenti logici applicati al gioco della battaglia navale.

Questa funzione di attacco: `int attaccoForzato()` prevede che venga richiamata prima di ogni attacco casuale che viene effettuato solo se questa restituisce come valore di ritorno uno 0.

Un ciclo `for` passa in rassegna il campo e se identifica una cella di stato 3, quindi una nave colpita ma non affondata, allora esce dal ciclo con un `break` salvandone virtualmente gli indici che corrisponderanno alle coordinate del punto di riferimento.

Se è stato riconosciuto un punto di riferimento allora alla variabile di ritorno si assegna 1 e si procede con lo studio degli stadi delle celle adiacenti assegnando, in base ad essi, un valore, compreso tra 0 e 2, alle variabili direzionali dichiarate internamente alla funzione e usate per identificare quali celle considerare per l'attacco:

| Variabili direzionali rappresentati le caselle adiacenti al punto di riferimento |      |        |                                                                                                    |                                                                                                             |                                                                                                |
|----------------------------------------------------------------------------------|------|--------|----------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|
| formato                                                                          | nome | tipo   | C.E delle variabili direzionali                                                                    |                                                                                                             |                                                                                                |
|                                                                                  |      |        | 0 - bersagliabile                                                                                  | 1 - ignora                                                                                                  | 2 - considera                                                                                  |
| int                                                                              | top  | lavoro | Se la cella adiacente è di stato 0 o 1 allora assegna valore 0 alla relativa variabile direzionale | Se la cella adiacente è di stato 2 o non esiste allora assegna valore 1 alla relativa variabile direzionale | Se la cella adiacente è di stato 3 allora assegna valore 2 alla relativa variabile direzionale |
| int                                                                              | dex  |        |                                                                                                    |                                                                                                             |                                                                                                |
| int                                                                              | bot  |        |                                                                                                    |                                                                                                             |                                                                                                |
| int                                                                              | sin  |        |                                                                                                    |                                                                                                             |                                                                                                |

### Casistiche di attacco con 4 celle di stato bersagliabili

Si attacca casualmente una delle 4 caselle adiacenti per mirare ad affondare la nave:



```
if ( top == 0 && dex == 0 && bot == 0 && sin == 0 )
```

| Direzione<br><code>dir = 1 + rand() % 4;</code> | Condizione                   | Istruzioni                                                                                          |
|-------------------------------------------------|------------------------------|-----------------------------------------------------------------------------------------------------|
| Top<br>                                         | <code>if ( dir == 1 )</code> | <code>i = i - 1;</code><br><code>esitoAttacco(j,i,lett,campo);</code><br><code>return check;</code> |
| Dex<br>                                         | <code>if ( dir == 2 )</code> | <code>J = j + 1;</code><br><code>esitoAttacco(j,i,lett,campo);</code><br><code>return check;</code> |
| Bot<br>                                         | <code>if ( dir == 3 )</code> | <code>i = i + 1;</code><br><code>esitoAttacco(j,i,lett,campo);</code><br><code>return check;</code> |
| Sin<br>                                         | <code>if ( dir == 4 )</code> | <code>j = j - 1;</code><br><code>esitoAttacco(j,i,lett,campo);</code><br><code>return check;</code> |

### Casistiche di attacco con 1 cella bersagliabile







Si attacca solo la cella bersagliabile per mirare ad affondare la nave


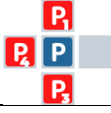




| Direzione                            | Condizione                                                                                    | Istruzione                                                                    |
|--------------------------------------|-----------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------|
| <b>Top</b> è l'unica attaccabile<br> | <code>if ( top == 0 &amp;&amp; dex == 1 &amp;&amp;<br/> bot == 1 &amp;&amp; sin == 1 )</code> | <code>i = i - 1;<br/> esitoAttacco(j,i,lett,campo);<br/> return check;</code> |
| <b>Dex</b> è l'unica attaccabile<br> | <code>if ( top == 1 &amp;&amp; dex == 0 &amp;&amp;<br/> bot == 1 &amp;&amp; sin == 1 )</code> | <code>j = j + 1;<br/> esitoAttacco(j,i,lett,campo);<br/> return check;</code> |
| <b>Bot</b> è l'unica attaccabile<br> | <code>if ( top == 1 &amp;&amp; dex == 1 &amp;&amp;<br/> bot == 0 &amp;&amp; sin == 1 )</code> | <code>i = i + 1;<br/> esitoAttacco(j,i,lett,campo);<br/> return check</code>  |
| <b>Sin</b> è l'unica attaccabile<br> | <code>if ( top == 1 &amp;&amp; dex == 1 &amp;&amp;<br/> bot == 1 &amp;&amp; sin == 0 )</code> | <code>= j - 1;<br/> esitoAttacco(j,i,lett,campo);<br/> return check;</code>   |

### Casistiche di attacco con 2 celle bersagliabili

Si attacca casualmente una delle 3 caselle adiacenti bersagliabili per mirare ad affondare la nave

| Direzione<br>$dir = 1 + rand() \% 2;$                                                                                     | Condizione                                                                               | Istruzioni                                                                                                      |
|---------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| <b>Top e Dex non bersagliabili</b><br>   | <code>if ( top == 1 &amp;&amp; dex == 1 &amp;&amp; bot == 0 &amp;&amp; sin == 0 )</code> | <code>if ( dir == 1 ) i = i + 1;<br/>else j = j - 1;<br/>esitoAttacco(j,i,lett,campo);<br/>return check;</code> |
| <b>Dex e bot non bersagliabili</b><br>   | <code>if ( top == 0 &amp;&amp; dex == 1 &amp;&amp; bot == 1 &amp;&amp; sin == 0 )</code> | <code>if ( dir == 1 ) j = j - 1;<br/>else i = i - 1;<br/>esitoAttacco(j,i,lett,campo);<br/>return check;</code> |
| <b>Bot e sin non bersagliabili</b><br>  | <code>if ( top == 0 &amp;&amp; dex == 0 &amp;&amp; bot == 1 &amp;&amp; sin == 1 )</code> | <code>if ( dir == 1 ) j = j + 1;<br/>else i = i - 1;<br/>esitoAttacco(j,i,lett,campo);<br/>return check;</code> |
| <b>Sin e top non bersagliabili</b><br> | <code>if ( top == 1 &amp;&amp; dex == 0 &amp;&amp; bot == 0 &amp;&amp; sin == 1 )</code> | <code>if ( dir == 1 ) j = j + 1;<br/>else i = i + 1;<br/>esitoAttacco(j,i,lett,campo);<br/>return check;</code> |
| <b>Sin e dex non bersagliabili</b><br> | <code>if ( top == 0 &amp;&amp; dex == 1 &amp;&amp; bot == 0 &amp;&amp; sin == 1 )</code> | <code>if ( dir == 1 ) i = i - 1;<br/>else i = i + 1;<br/>esitoAttacco(j,i,lett,campo);<br/>return check;</code> |
| <b>Top e bot non bersagliabili</b><br> | <code>if ( top == 1 &amp;&amp; dex == 0 &amp;&amp; bot == 1 &amp;&amp; sin == 0 )</code> | <code>if ( dir == 1 ) j = j + 1;<br/>else j = j - 1;<br/>esitoAttacco(j,i,lett,campo);<br/>return check;</code> |

| <b>Casistiche di attacco con 3 celle bersagliabili</b><br>Si attacca casualmente una delle 3 caselle adiacenti bersagliabili per mirare ad affondare la nave |                                                     |                                                                                                                                    |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| Direzione<br>dir = 1 + rand() % 3;                                                                                                                           | Condizione                                          | Istruzioni                                                                                                                         |
| <b>Top non bersagliabile</b><br>                                            | if ( top == 1 && dex == 0 && bot == 0 && sin == 0 ) | if ( dir == 1 ) j = j + 1;<br>else if ( dir == 2 ) i = i + 1;<br>else j = j - 1;<br>esitoAttacco(j,i,lett,campo);<br>return check; |
| <b>Dex non bersagliabile</b><br>                                            | if ( top == 0 && dex == 1 && bot == 0 && sin == 0 ) | if ( dir == 1 ) i = i - 1;<br>else if ( dir == 2 ) j = j - 1;<br>else i = i + 1;<br>esitoAttacco(j,i,lett,campo);<br>return check; |
| <b>Bot non bersagliabile</b><br>                                            | if ( top == 0 && dex == 0 && bot == 1 && sin == 0 ) | if ( dir == 1 ) i = i - 1;<br>else if ( dir == 2 ) j = j + 1;<br>else j = j - 1;<br>esitoAttacco(j,i,lett,campo);<br>return check; |
| <b>Sin non bersagliabile</b><br>                                           | if ( top == 0 && dex == 0 && bot == 0 && sin == 1 ) | if ( dir == 1 ) i = i - 1;<br>else if ( dir == 2 ) j = j + 1;<br>else i = i + 1;<br>esitoAttacco(j,i,lett,campo);<br>return check; |

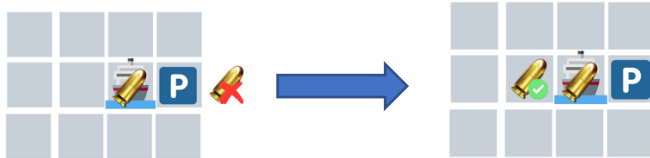


### Casistiche di attacco in cui vi è una nave colpita ma non affondata adiacente al punto di riferimento

Considera, sia la cella adiacente al punto di riferimento di stato 3, sia quella adiacente a quest'ultima nel margine opposto.

Se la cella adiacente al punto di riferimento di stato 3 è bersagliabile allora la attacca altrimenti attacca l'altra.

E' stata implementata una condizione tale che controlli se è possibile attaccare la cella adiacente al punto di riferimento e quella adiacente all'altra nave in modo che l'attacco venga effettuato correttamente: se il punto di riferimento è adiacente ad un bordo allora lo rileva e attua l'attacco alternativo.



| Direzione      | Condizione                                                                           | Istruzione                                                   | Attacco |
|----------------|--------------------------------------------------------------------------------------|--------------------------------------------------------------|---------|
| <b>Top</b><br> | <b>if</b> ( top == 2 && bot == 0 )                                                   | i = i + 1;<br>esitoAttacco(j,i,lett,campo);<br>return check; |         |
|                | <b>if</b> ( top == 2 && ( campo[i - 2][j] == 0    campo[i - 2][j] == 1 ) && i >= 2 ) | i = i - 2;<br>esitoAttacco(j,i,lett,campo);<br>return check; |         |
| <b>Dex</b><br> | <b>if</b> ( dex == 2 && sin == 0 )                                                   | j = j - 1;<br>esitoAttacco(j,i,lett,campo);<br>return check; |         |
|                | <b>if</b> ( dex == 2 && ( campo[i][j + 2] == 0    campo[i][j + 2] == 1 ) && j <= 7 ) | j = j + 2;<br>esitoAttacco(j,i,lett,campo);<br>return check; |         |
| <b>Bot</b><br> | <b>if</b> ( bot == 2 && top == 0 )                                                   | i = i - 1;<br>esitoAttacco(j,i,lett,campo);<br>return check; |         |
|                | <b>if</b> ( bot == 2 && ( campo[i + 2][j] == 0    campo[i + 2][j] == 1 ) && i <= 7 ) | i = i + 2;<br>esitoAttacco(j,i,lett,campo);<br>return check; |         |
| <b>Sin</b><br> | <b>if</b> ( sin == 2 && dex == 0 )                                                   | j = j + 1;<br>esitoAttacco(j,i,lett,campo);<br>return check; |         |
|                | <b>if</b> ( sin == 2 && ( campo[i][j - 2] == 0    campo[i][j - 2] == 1 ) && j >= 2 ) | j = j - 2;<br>esitoAttacco(j,i,lett,campo);<br>return check; |         |

## DATI DI RIFERIMENTO PER LA FASE DI ATTACCO

Le procedure e funzioni utilizzate sono:

- relative al controllo dell'attacco  
`int controlloAttacco ();`
- relative all'esito dell'attacco  
`void EsitoAttacco ();`  
`void affondaNave ();`
- Relative agli attacchi:
  - o attacco manuale  
`void attacco ();`
  - o attacco automatico
    - casuale  
`void autoAttacco ();`
    - forzato  
`int attaccoForzato ();`

| Procedure e funzioni per la fase di attacco |                  |      |              |              |      |              |                                                                                           |
|---------------------------------------------|------------------|------|--------------|--------------|------|--------------|-------------------------------------------------------------------------------------------|
| formato                                     | nome             |      | parametri    |              |      |              | descrizione                                                                               |
|                                             |                  |      | Par1         | Par2         | Par3 | Par4         |                                                                                           |
| int                                         | controlloAttacco | tipo | Int<br>[] [] | Int          | Int  |              | Verifica se tutte<br>l'attacco è possibile                                                |
|                                             |                  | nome | matrice      | x            | y    |              |                                                                                           |
| void                                        | esitoAttacco     | tipo | char []      | Int          | Int  | Int<br>[] [] | Da l'esito dell'attacco<br>(A5 nave)                                                      |
|                                             |                  | nome | lett         | x            | y    | matrice      |                                                                                           |
| void                                        | attacco          | tipo | char []      | int          |      |              | Consiste<br>nell'inserimento da<br>tastiera delle<br>coordinate                           |
| void                                        | autoAttacco      | nome | lett         | matrice      |      |              |                                                                                           |
| void                                        | affondaNave      | tipo | Int[] []     |              |      |              | Identifica un punto di<br>stato 3 e in base al<br>caso affonda la nave                    |
|                                             |                  | nome | matrice      |              |      |              |                                                                                           |
| int                                         | attaccoForzato   | tipo | char []      | Int<br>[] [] |      |              | Identifica un punto di<br>stato 3 e bersaglia<br>uno adiacente in base<br>alle casistiche |
|                                             |                  | nome | lett         | matrice      |      |              |                                                                                           |

## LA FASE DI GIOCO

**Schierate le navi, i giocatori sono pronti ad iniziare la partita**

### REGOLE DEL TURNO DI GIOCO

L'insieme di due mosse, effettuate consequenzialmente prima da un giocatore e solo poi dall'altro, è il turno di gioco; questo si definisce dopo che, i giocatori, schierate tutte le navi, si battono in un "testa o croce"; chi vince da inizio alla partita, i cui turni si succederanno tali che la prima mossa spetti al vincitore e la seconda al nemico.

### CONSIDERAZIONI

#### NUMERO MASSIMO E MINIMO DI TURNI PER VINCERE

##### Numero minimo di turni per vincere

Sapendo che affondando tutte le navi nemiche si vince allora si può solo da quando il numero di mosse è maggiore o uguale a quello minimo e necessario ad affondare tutte le navi nemiche; in quest'ultimo caso, il margine d'errore complessivo del giocatore vincente al turno  $n$  sarà nullo, avendo colpito una nave nemica ad ogni attacco.

Quindi, la condizione per poter vincere al turno minimo, si verifica matematicamente, risultando uno il calcolo della precisione determinabile dal rapporto tra la quantità di celle occupate dalle navi schierate ed il numero di mosse effettuato:

*(Le variabili non hanno riferimenti esterni)*

$x$  = attacchi effettuati

$y$  = celle occupate

$n$  = numero minimo di turni per vincere

$p$  = precisione

$$\frac{x}{y} = 1 = p \rightarrow n = y = x$$

La precisione è quindi calcolata dalla funzione:

```
float precisione ( int d, int turni ){  
    float mira;  
    mira = (float) d / turni;  
    mira = mira * 100;  
    return mira;  
}
```

### Numero massimo di turni per vincere

Noti solo questi fattori si può sapere ciò e, senza considerarne altri, si potrebbe pensare che, nonostante vi sia un numero minimo di attacchi, per vincere non ve ne sarebbe uno massimo; ciò se solo non vi fosse un limite. Nello specifico il numero di turni massimo per vincere è vincolato dal numero di celle contenute dalla griglia risultante dal prodotto tra il numero delle righe e quello delle colonne in quanto ad ogni attacco è associata una cella ed attaccate tutte le caselle non se ne possono sferrare altri.

Sulla base di ciò sono stati programmati due for tali che abbiano come limite il numero totale di celle della matrice e come istruzioni quelle di attacco; si differenziano in quanto cambia l'ordine delle mosse:

- nel primo l'utente fa la prima mossa
- nel secondo l'avversario fa la prima mossa

Andato a segno l'attacco, prima di passare la mossa, controlla se tutte le navi del campo nemico sono state affondate con la funzione `int controllo ( int h, int [Y][X] )`; che passato 4 come al parametro h conta le celle della matrice di stato 4 che devono essere pari a quelle di stato 1 schierate inizialmente: se ciò accade allora termina la partita e visualizza l'esito, altrimenti passa la mossa all'altro giocatore.

### VISUALIZZAZIONE DEL TURNO DI GIOCO

Del turno sono visualizzati:

- Il numero del turno
- Il giocatore che effettua la mossa
- L'avviso "attacco forzato" solo se è stata applicata la logica di attacco implementata
- Il tipo di cella colpita, in base allo stato, e le relative coordinate
- L'abbattimento di una nave specificandone il tipo in base alla grandezza
- L'avviso se si ha vinto o perso in base all'esito della partita

TURNO: 81

Turno tuo

L'attacco è andato a segno  
Acqua nel punto 10G

la tua mappa

|    |   |   |   |   |   |   |   |   |   |   |
|----|---|---|---|---|---|---|---|---|---|---|
| 1  | - | - | - | - | - | - | - | - | - | - |
| 2  | 0 | - | - | - | - | - | - | - | - | - |
| 3  | - | - | - | - | 0 | 0 | - | - | - | - |
| 4  | - | - | - | - | - | - | - | - | - | - |
| 5  | - | - | - | - | - | - | - | - | - | - |
| 6  | - | - | 0 | - | - | - | - | - | - | - |
| 7  | - | - | - | - | - | - | - | - | - | - |
| 8  | 0 | - | - | 0 | 0 | 0 | - | - | 0 | - |
| 9  | 0 | - | - | - | - | - | - | - | - | - |
| 10 | - | - | - | - | - | - | - | - | - | - |
|    | A | B | C | D | E | F | G | H | I | J |

visuale nemica

|    |   |   |   |   |   |   |   |   |   |   |
|----|---|---|---|---|---|---|---|---|---|---|
| 1  | ? | - | - | - | ? | - | - | - | ? | - |
| 2  | X | - | ? | ? | - | - | - | - | - | - |
| 3  | - | - | ? | - | - | - | X | X | - | - |
| 4  | - | - | - | - | ? | ? | - | - | - | - |
| 5  | - | - | ? | - | - | - | ? | ? | - | - |
| 6  | - | - | - | X | - | - | - | - | ? | - |
| 7  | - | - | ? | - | - | - | ? | - | - | - |
| 8  | 0 | - | - | X | X | X | - | - | X | - |
| 9  | - | ? | ? | - | - | - | ? | - | ? | ? |
| 10 | - | - | - | - | - | - | - | - | - | ? |
|    | A | B | C | D | E | F | G | H | I | J |

mappa nemica

|    |   |   |   |   |   |   |   |   |   |   |
|----|---|---|---|---|---|---|---|---|---|---|
| 1  | ? | - | - | - | - | - | - | - | - | ? |
| 2  | ? | - | - | - | - | - | X | X | - | - |
| 3  | - | ? | - | - | - | - | - | - | - | - |
| 4  | - | ? | - | - | - | ? | - | - | X | - |
| 5  | ? | - | - | - | - | - | - | - | X | - |
| 6  | - | - | X | - | - | - | - | - | X | ? |
| 7  | - | ? | X | ? | X | - | - | - | ? | - |
| 8  | - | - | - | - | - | - | - | - | ? | ? |
| 9  | - | - | ? | ? | ? | X | - | - | ? | ? |
| 10 | - | - | - | - | - | - | - | - | - | - |
|    | A | B | C | D | E | F | G | H | I | J |

Turno nemico

Attacco forzato!

L'attacco è andato a segno  
Nave colpita nel punto 9B

Torpediniere affondato!

la tua mappa

|    |   |   |   |   |   |   |   |   |   |   |
|----|---|---|---|---|---|---|---|---|---|---|
| 1  | - | - | - | - | - | - | - | - | - | - |
| 2  | 0 | - | - | - | - | - | - | - | - | - |
| 3  | - | - | - | - | 0 | 0 | - | - | - | - |
| 4  | - | - | - | - | - | - | - | - | - | - |
| 5  | - | - | - | - | - | - | - | - | - | - |
| 6  | - | - | 0 | - | - | - | - | - | - | - |
| 7  | - | - | - | - | - | - | - | - | - | - |
| 8  | 0 | - | - | 0 | 0 | 0 | - | - | 0 | - |
| 9  | 0 | - | - | - | - | - | - | - | - | - |
| 10 | - | - | - | - | - | - | - | - | - | - |
|    | A | B | C | D | E | F | G | H | I | J |

visuale nemica

|    |   |   |   |   |   |   |   |   |   |   |
|----|---|---|---|---|---|---|---|---|---|---|
| 1  | ? | - | - | - | ? | - | - | - | ? | - |
| 2  | X | - | ? | ? | - | - | - | - | - | - |
| 3  | - | - | ? | - | - | - | X | X | - | - |
| 4  | - | - | - | - | ? | ? | - | - | - | - |
| 5  | - | - | ? | - | - | - | ? | ? | - | - |
| 6  | - | - | - | X | - | - | - | - | ? | - |
| 7  | - | - | ? | - | - | - | ? | - | - | - |
| 8  | - | X | - | - | X | X | X | - | - | X |
| 9  | - | X | ? | - | - | - | ? | - | ? | ? |
| 10 | - | - | - | - | - | - | - | - | - | ? |
|    | A | B | C | D | E | F | G | H | I | J |

mappa nemica

|    |   |   |   |   |   |   |   |   |   |   |
|----|---|---|---|---|---|---|---|---|---|---|
| 1  | ? | - | - | - | - | - | - | - | - | ? |
| 2  | ? | - | - | - | - | - | X | X | - | - |
| 3  | - | ? | - | - | - | - | - | - | - | - |
| 4  | - | ? | - | - | - | ? | - | - | X | - |
| 5  | ? | - | - | - | - | - | - | - | X | - |
| 6  | - | - | X | - | - | - | - | - | X | ? |
| 7  | - | ? | X | ? | X | - | - | - | ? | - |
| 8  | - | - | - | - | - | - | - | - | ? | ? |
| 9  | - | - | ? | ? | ? | X | - | - | ? | ? |
| 10 | - | - | - | - | - | - | - | - | - | - |
|    | A | B | C | D | E | F | G | H | I | J |




Hai perso!

Quanto visualizzato ad ogni turno offre tutti gli strumenti necessari per campionare le fasi dell'esecuzione del programma verificandone la correttezza "step by step". Ad esempio, considerando il giocatore che ha effettuato la mossa ed i dati in input dell'attacco, si può verificare se sia stato sferrato correttamente, colpendo la cella giusta del campo nemico; oppure si può controllare, che al termine della partita, ogni giocatore abbia effettuato un solo attacco per turno se le celle rimanenti dovessero risultare pari alla sottrazione tra il numero di quelle totali e quello dei turni effettuati; se il giocatore considerato ha saltato il proprio turno allora il numero di celle coperte alla fine sarà maggiore del risultato della sottrazione o se ha attaccato più volte allora il numero di questo valore sarà minore al risultato della sottrazione. Dalle verifiche del caso, che l'utente può compiere ad occhio, si può riscontrare ad ogni campionamento la correttezza di ogni step.

## CONCLUSIONI

Il test di Turing è un criterio per determinare se una macchina sia in grado di esibire un comportamento intelligente: in una stanza con una macchina e 2 persone, una che fa da osservatore e l'altra che adempie allo scopo della IA, se l'osservatore, senza guardare l'esecuzione del test, non è in grado di distinguere le risposte dell'uomo da quelle della macchina allora questa è una IA.

Se contestualizzato si potrebbe dire che, non differendo da quello umano, il comportamento del bot che simula la battaglia navale, è quindi reso intelligente in quanto applica gli stessi ragionamenti di quelli umani; ciò se solo, durante l'attacco forzato, nel caso in cui vi sia una cella di stato 3 adiacente a quella di riferimento, venisse attaccata casualmente una delle due bersagliabili, se entrambe lo fossero, invece di colpirne prima una e poi alla mossa successiva l'altra.

| Direzione                                                                          | Condizione                                                                                                  | Istruzione                                                        | Attacco                                                                              |
|------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------|--------------------------------------------------------------------------------------|
|  | <pre>if ( top == 2 &amp;&amp; bot == 0 )</pre>                                                              | <pre>i = i + 1; esitoAttacco(j,i,lett,campo); return check;</pre> |   |
|                                                                                    | <pre>if ( top == 2 &amp;&amp; ( campo[i - 2][j] == 0    campo[i - 2][j] == 1 ) &amp;&amp; i &gt;= 2 )</pre> | <pre>i = i - 2; esitoAttacco(j,i,lett,campo); return check;</pre> |  |

In eventuali sviluppi futuri si potrebbe quindi implementare la possibilità che, nel caso in cui siano possibili entrambi gli attacchi, ne venga scelto uno casualmente.

Inoltre, un eventuale sviluppo, potrebbe essere la riprogrammazione della funzione di affondamento della nave in modo che venga generalizzata, distaccandosi dal ragionamento su cui si basa, che affronta le casistiche specifiche singolarmente e che perciò permette l'abbattimento solo di:

- navi da 1 casella
- navi da 2 caselle
- navi da 3 caselle

Ciò consentirebbe di affondare una qualsiasi nave.