UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA DELL'INFORMAZIONE

Gioco di Battaglia Navale sviluppato in linguaggio di programmazione C

Relatore: Prof. Antonio Giunta

Laureando: Nicola Pozzo

ANNO ACCADEMICO 2022 – 2023
Data di laurea: 25/09/2023

SOMMARIO

1
FINALITÀ DI QUESTO LAVORO

1.1 Il gioco della Battaglia Navale e le sue origini

Battaglia navale è un gioco estremamente popolare e diffuso in tutto il mondo. La versione originale “con carta e penna” ha ispirato varie edizioni del gioco in scatola, versioni elettroniche portatili, per computer e persino un film.

Sulle origini del gioco non si hanno notizie certezze: secondo alcuni è stato ideato, intorno al 1900, da un certo Clifford Van Wickler (o Von Wickler) senza però brevettarlo; secondo altri è stato ideato in Francia nel corso della Prima guerra mondiale (si chiamava *L’Attaque*), ma anche Stati Uniti e Russia ne rivendicano la sua paternità nello stesso periodo.

Senza dubbio però il gioco è stato pubblicato per la prima volta dalla casa editrice statunitense di giochi da tavolo “Milton Bradley Company” nel 1931 e importato in Italia dal famoso settimanale "Domenica del Corriere" nel 1932, che lo presentò come una novità proveniente dall'America.

In questo gioco l’obiettivo di ciascun giocatore è quello di affondare tutte le navi del nemico, prima che questi abbia il sopravvento e affondi quelle dell’avversario. Pertanto, fortuna, intuito e strategia sono determinanti per vincere.

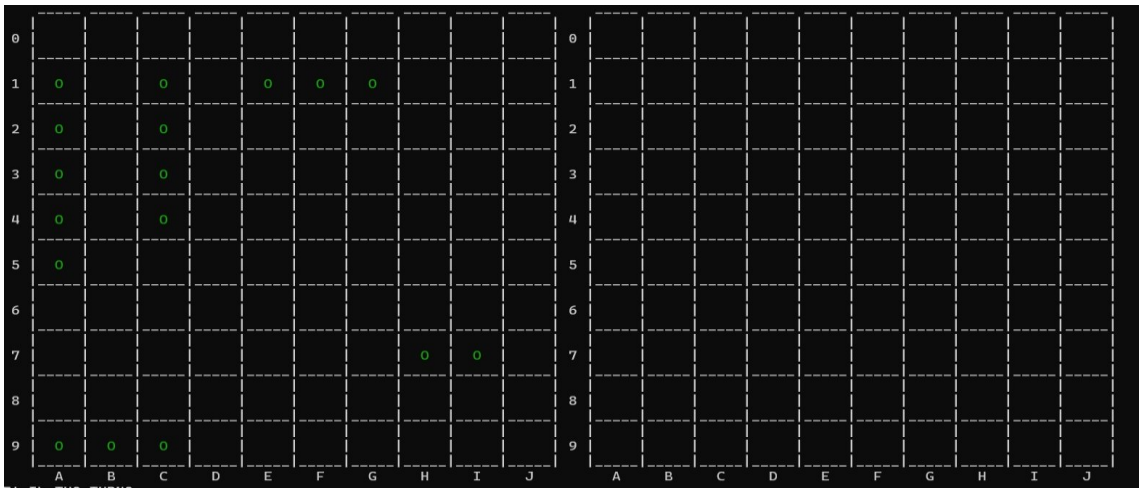
1.2 Svolgimento del gioco

Per giocare a battaglia navale occorrono quattro griglie (due per giocatore), tutte con lo stesso numero di righe e con lo stesso numero di colonne.

La versione classica è 10×10, ma si possono anche concordare dimensioni diverse. I quadretti della tabella sono identificati da coppie di coordinate, corrispondenti a riga e colonna; tradizionalmente si usano lettere per le colonne e numeri per le righe (esempio le celle sono "1-A", "6-B", e così via).

All'inizio, i giocatori devono "posizionare le proprie navi" segnandole su una delle loro due griglie (che terranno nascoste all'avversario per tutta la durata del gioco), inserendole orizzontalmente o verticalmente, non in diagonale,

Un esempio delle griglie che avrete all’inizio della partita è rappresentato nell’immagine qui sotto.



La griglia a sinistra è quella in cui abbiamo inserito le nostre navi e che l’avversario dovrà cercare di colpire, mentre la griglia a destra è quella in cui segneremo le mosse che andiamo a fare per cercare di affondare la flotta nemica.

Ogni battaglia navale può prevedere una formazione diversa delle navi, ma solitamente le navi si dividono:

- Una nave lunga cinque spazi (portaerei)
- Una nave lunga quattro spazi (corazzata)
- Due navi lunghe tre spazi (incrociatore e sottomarino)
- Una nave lunga due spazi (cacciatorpediniere).

Nella maggior parte dei casi, le regole "ufficiali" riportate nelle confezioni del gioco dicono che le navi non si devono toccare e che il giocatore deve annunciare anche quale nave è stata colpita (ad esempio, la portaerei).
Il gioco si svolge a turni alternati e ogni turno è composto da due fasi:

- Il giocatore di turno dichiara quale casella della griglia avversaria vuole colpire
- Il giocatore che subisce l'attacco risponde dichiarando se la mossa fatta ha avuto come risultato ACQUA, oppure COLPITO, oppure COLPITO E AFFONDATO.

Il gioco procede fino a quando uno dei due giocatori non sarà riuscito ad affondare tutte le navi della flotta avversaria.

1.3 Obiettivo di questa tesi

L'obbiettivo di questo progetto è di sviluppare un programma in linguaggio di programmazione C che permetta all'utente che lo utilizza di giocare contro il computer, ma con delle differenze rispetto alla versione classica del gioco sopra riportata:

- 1. le navi possono toccarsi (ma non possono essere nello stesso quadretto).
- 2. nel momento dell'affondamento NON viene dichiarata la tipologia di nave colpita

In questo modo si ampliano moltissimo le possibili combinazioni di navi e pertanto diventano ancor più importanti la logica e la strategia rispetto alla semplice fortuna di trovare a caso le navi.

Potremmo definirla "Battaglia Navale cieca" dove, non avendo la certezza del tipo di navi affondate bisogna ragionare per valutare le possibili combinazioni ed agire di conseguenza. Ecco, quindi, la difficoltà nella programmazione del gioco in quanto il computer, rappresentando il secondo giocatore, dovrà eseguire le mosse imitando quelli che sono i ragionamenti di un giocatore umano.

Come nella versione classica l'utente potrà modificare i parametri della partita, cioè, potrà cambiare le dimensioni della griglia, la lunghezza e il numero delle navi.

2
EVENTUALI LIMITI

2.1 Numero massimo di righe e colonne

Per come è stato sviluppato questo programma la griglia può avere una dimensione massima di 10x26: le righe numerate da 0 a 9 e le colonne con le 26 lettere dell'alfabeto italiano

2.2 Anello chiuso di navi

Essendo le combinazioni di navi infinite, non è stato possibile testare il programma in ogni situazione. Non possiamo quindi escludere che, in presenza di schemi con navi particolarmente incrociate, il computer non riesca a distinguerne le dimensioni e di conseguenza non sappia che mossa fare.

Dopo numerosi test eseguiti, l'unico schema di navi che all'incirca il 2% delle volte causa problemi è quando le navi vengono messe ad anello chiuso. (C'è da precisare che la maggior parte dei test è stata effettuata con una flotta composta da una nave da 5 caselle, una nave da 4 caselle, due navi da 3 caselle e una nave da 2 caselle).

2.3 Colori navi

Il colore delle navi è stato realizzato tramite delle sequenze di escape. Queste ultime non dovrebbero causare problemi con sistemi Unix (anche se non l'ho testato, e quindi non ho la certezza), mentre per quanto riguarda alcuni dispositivi Windows (nel mio computer con Windows 11 non ha avuto problemi) le sequenze di escape potrebbero non essere riconosciute andando a compromettere la visione delle griglie durante il gioco.

3
ALGORITMI E STRUTTURE DI DATI PIÙ SIGNIFICATIVI

3.1
Algoritmi

La parte principale di questo programma è la decisione da parte del computer della prossima mossa da eseguire. Per scegliere, il computer deve innanzitutto capire in che situazione si trova, successivamente fare una serie di ragionamenti (simili a quelli svolti da una persona che gioca a battaglia navale) e per ultimo decidere come agire.

Visto che le navi possono toccarsi e che non viene segnalata la tipologia di nave affondata, il computer inizialmente potrebbe interpretarle in maniera sbagliata (ad esempio, registra di aver affondato una nave da 5, anziché 1 nave da 4 e 1 casella colpita di altra adiacente). Sono state quindi schematizzate innanzitutto le tipologie di mosse utilizzate e successivamente le possibili eccezioni.

Vediamo ora tutti i casi e i ragionamenti inseriti nel programma.

3.1.1 Mossa_random

Casi in cui viene eseguita:

- non è stata ancora trovata nessuna nave
- le navi trovate sono state affondate.

Il computer in questo caso colpisce casualmente una casella della griglia.

3.1.2 Mossa_ragionata

È eseguita quando il computer:

- trova una nuova nave
- deve gestire una nave che è stata affondata ma che non è coerente con il tipo di navi che rimangono da affondare

Riesce a gestire, con più mosse ragionate, anche i casi particolari in cui troviamo più navi.

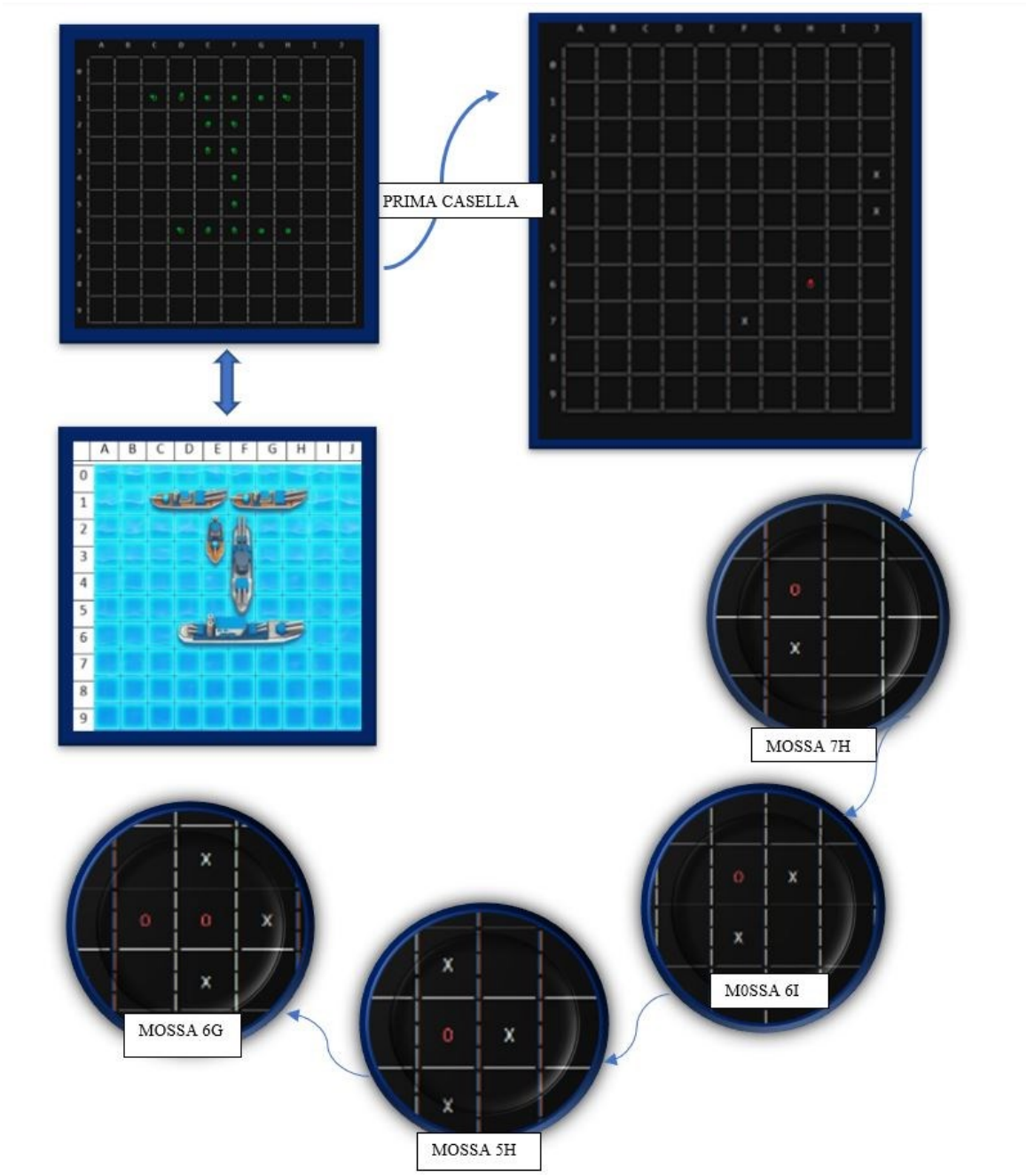
In base alla situazione invoca le seguenti funzioni che saranno dettagliate nelle pagine successive:

- MossaUnaCasella
- MossaPiuCaselle
- MossaNaviDoppie
- MossaIncrociata

3.1.2.1 MossaUnaCasella

Quando abbiamo colpito una casella di una nave, inizialmente il computer controlla se è possibile colpire le quattro caselle adiacenti. Se almeno una mossa risulta fattibile, sceglie casualmente quale fare. Se invece non ci sono mosse idonee, si presentano due possibilità:

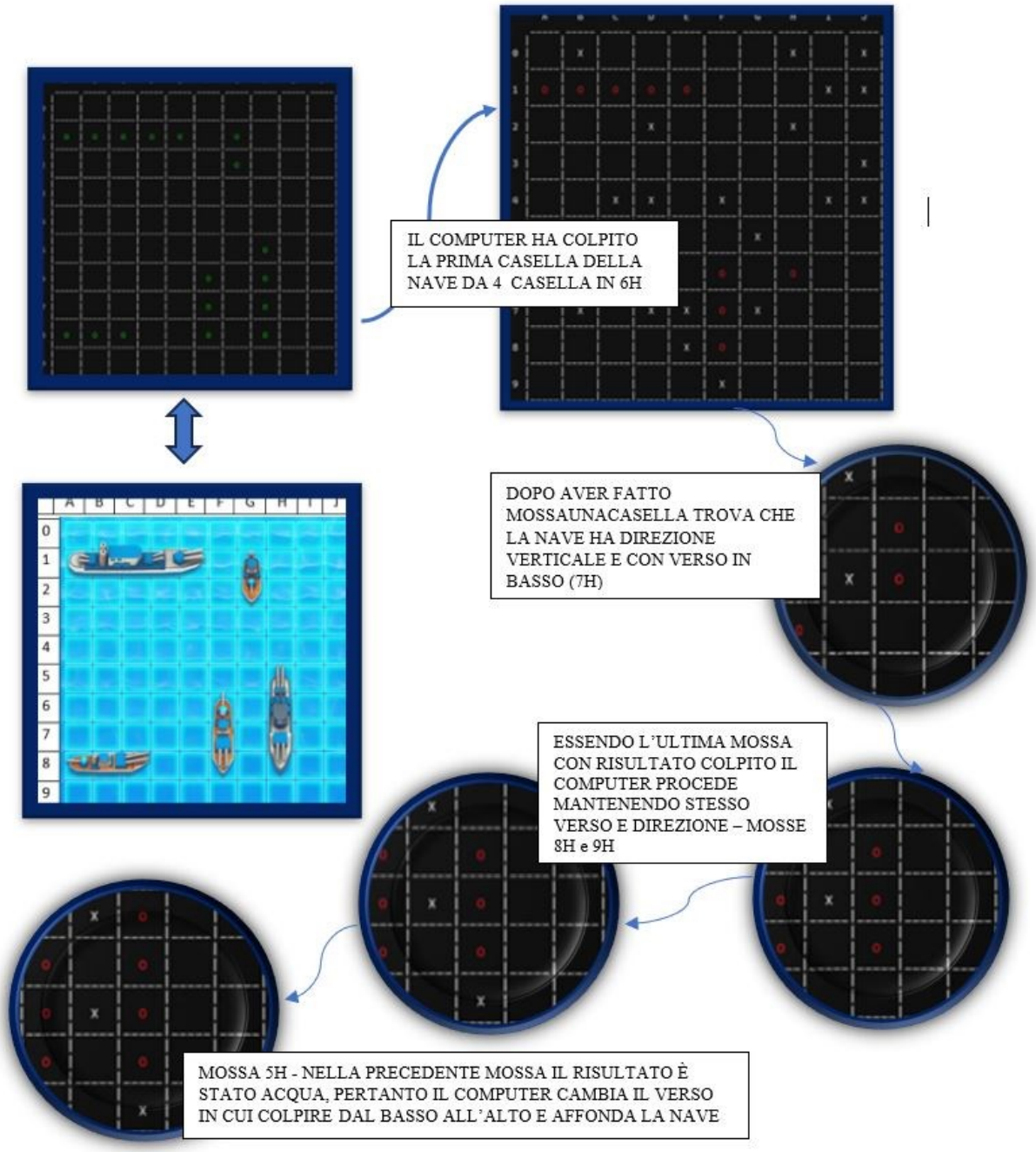
- Una casella di un'altra nave appartiene in realtà a quella che ho appena colpito. In questo caso verrà invocata la funzione MossaIncrociata (di cui parlerò in seguito)
- Non ci sono possibilità di mosse ulteriori. Di conseguenza, questa casella deve appartenere ad una nave che era stata mal interpretata precedentemente. In questo caso il computer aggiungerà questa casella a quella nave e ricomincerà i ragionamenti per fare la prossima mossa.



3.1.2.2 MossaPiuCaselle

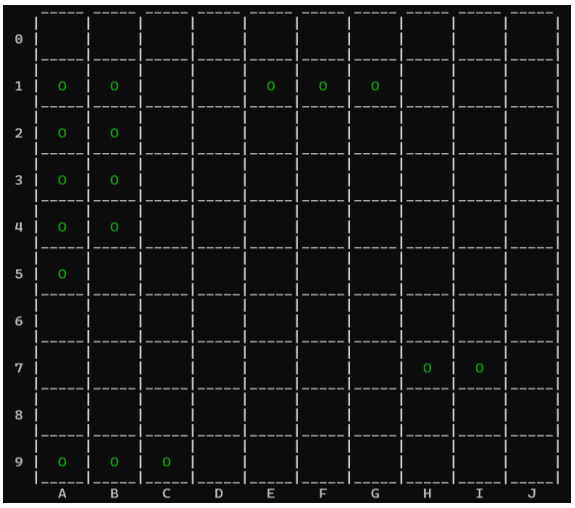
Quando abbiamo colpito due o più caselle (e la nave non risulta affondata) il computer, in base al risultato ottenuto nella precedente mossa, ragiona in uno di questi due modi:

- se la mossa precedente ha avuto come risultato colpito, allora prosegue a colpire nella stessa direzione e nello stesso verso
- se la mossa precedente ha avuto come risultato acqua, allora prosegue a colpire nella stessa direzione ma nel verso opposto



Nel caso in cui da ambo i lati non possiamo più colpire e non abbiamo ancora ricevuto come risultato affondato, allora potremmo avere uno di questi due casi:

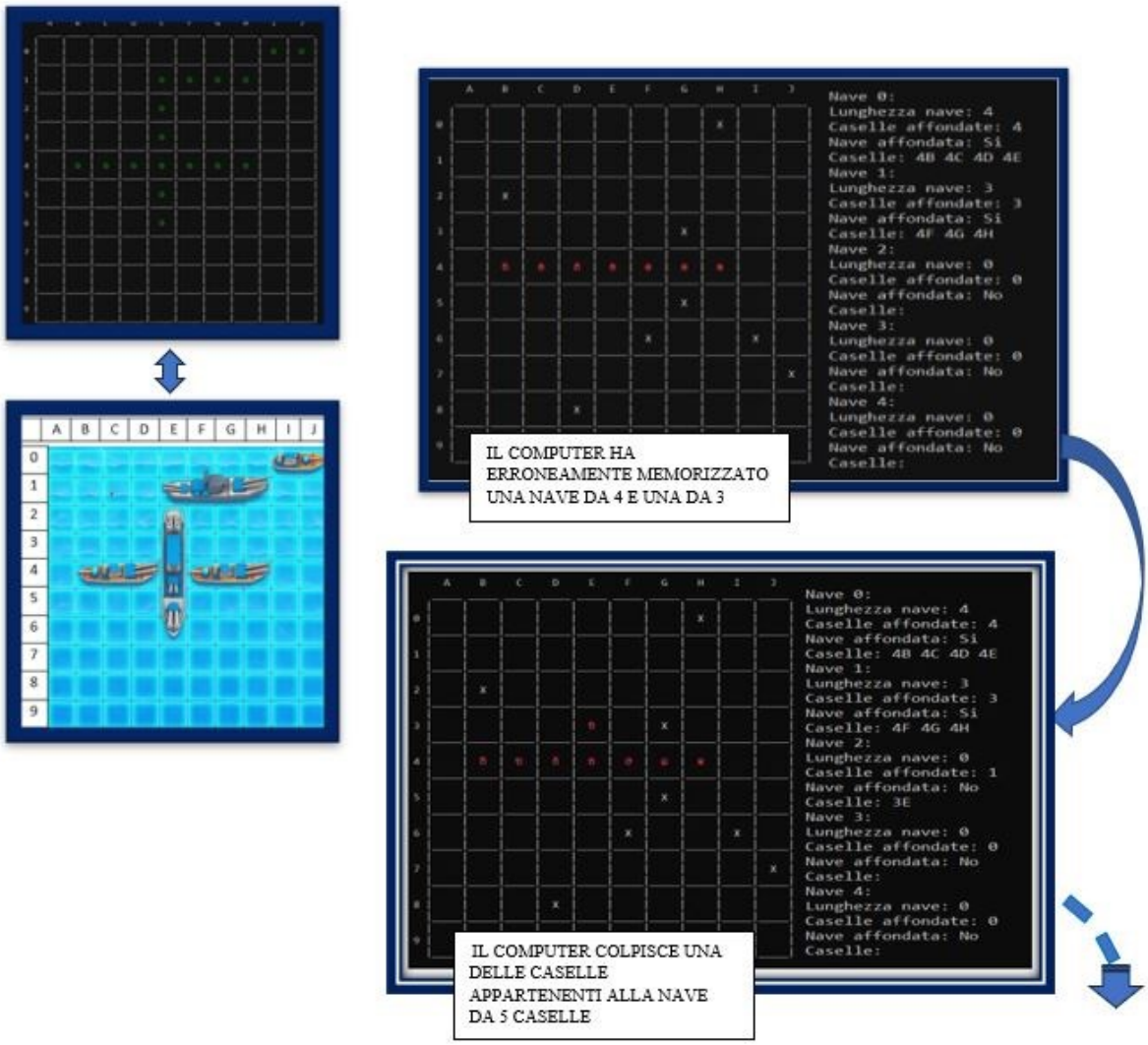
- Una casella di un'altra nave appartiene in realtà a quella che ho appena colpito; in questo caso verrà invocata la funzione MossaIncrociata (di cui parlerò in seguito)
- Se non c'è la possibilità di mossa incrociata, allora entriamo nel caso di navi affiancate. Un esempio è nell'immagine seguente dove le due navi, quella da quattro e quella da cinque caselle, si trovano una a fianco all'altra. In questo caso se il computer riceve un colpito in 2A, in 2B e successivamente trova acqua in 2C, allora riconosce che nella direzione in cui stava colpendo non ha trovato una nave affondata, e questo significa che ci troviamo nel caso di navi affiancate nella direzione opposta. Prima di procedere con le mosse successive, il computer controlla, per ogni casella colpita, se è presente, in una delle due caselle adiacenti nella direzione opposta, una nave; in questo caso, la casella viene assegnata all'altra nave. (Nell'esempio fatto qui a lato non trova alcuna nave; quindi entrambe le caselle verranno trattate come navi separate.) Dopo aver fatto questi spostamenti per le caselle in cui era possibile, il computer riconosce ogni casella rimasta come una nave diversa, e con le mosse successive torna ai casi base e va a colpire una nave alla volta.

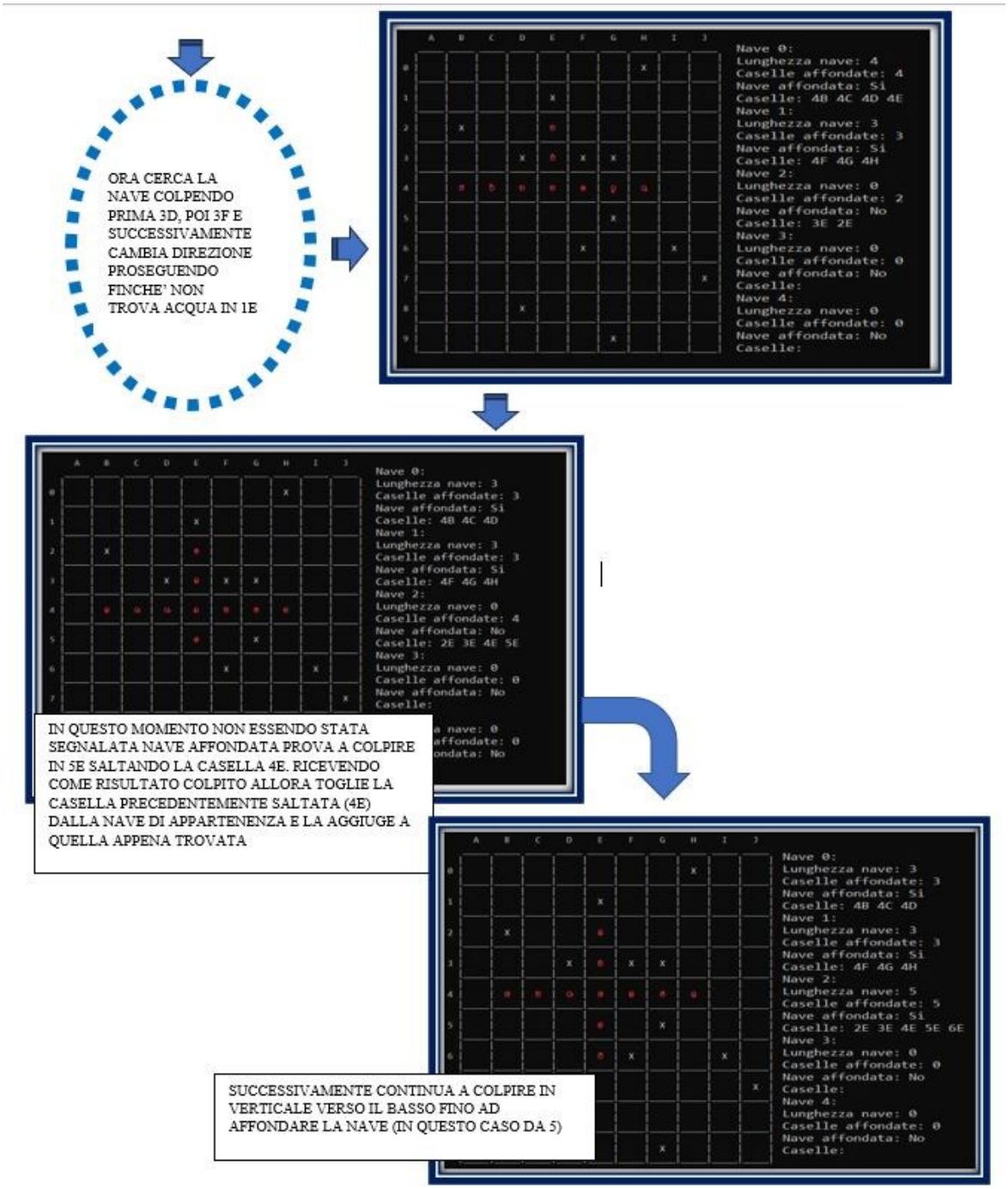


3.1.2.3 MossaNaviIncrociate

Come indicato precedentemente, questa mossa viene chiamata nel momento in cui il computer non rileva possibili mosse in quanto le caselle adiacenti sono state già colpite e almeno una di esse risulta appartenere ad altra nave già identificata. In questo caso andrà a colpire, se possibile, la prima casella successiva all'altra o altra/e nave/i,

Se il risultato della mossa è colpito, il computer riconosce la o le caselle saltate come appartenenti alla nave appena colpita, e le toglie dalla/e precedenti navi di appartenenza che vengono automaticamente aggiornate.





Nell'eventualità che queste nuove navi non siano previste dalla flotta, vengono temporaneamente salvate nell'array e successivamente trattate con apposite funzioni. Per rendere più chiaro il ragionamento propongo un esempio in cui il computer ha eseguito questa azione.

3.1.2.4 MossaNaviDoppie

Questo caso si presenta quando affondiamo una nave con:

- un numero di caselle che non è previsto
- affondiamo un numero di navi di una certa lunghezza maggiore di quelle previste.

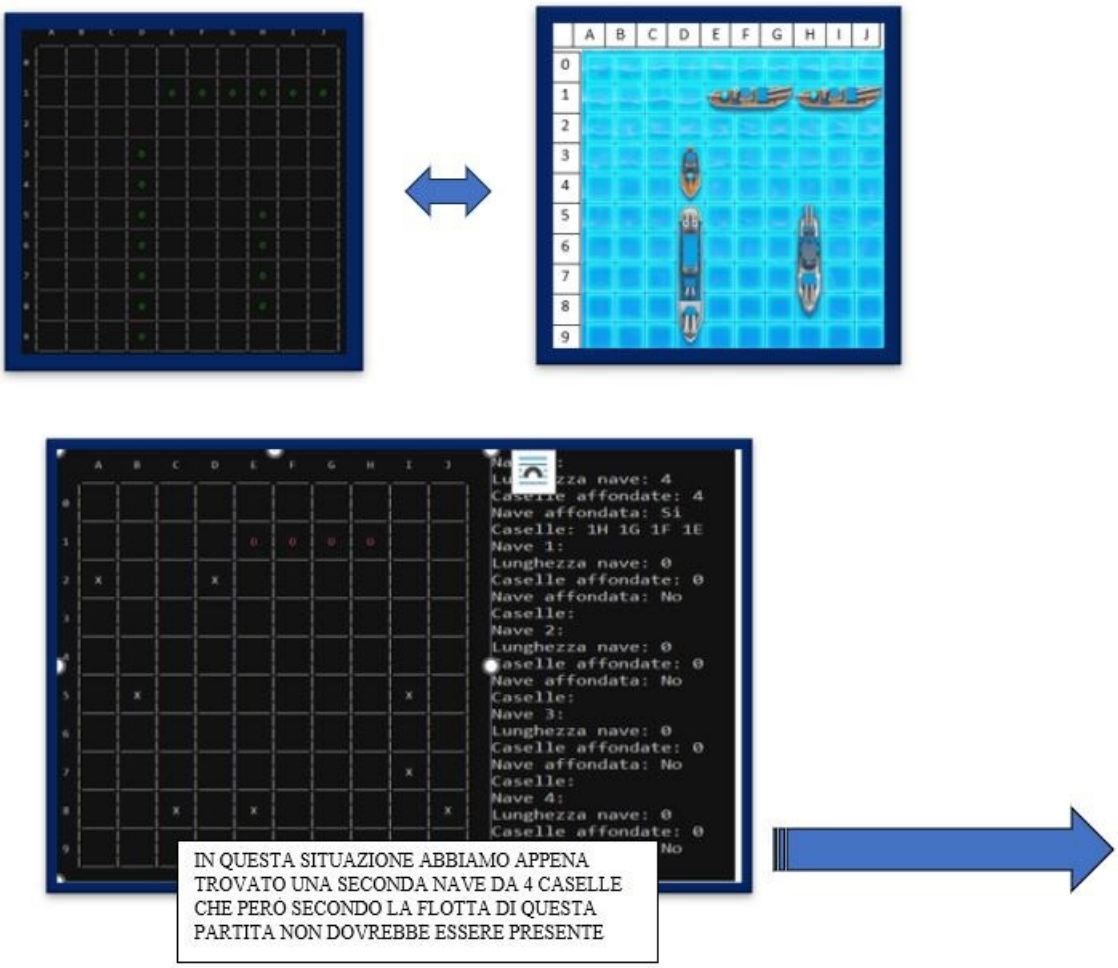
In tal caso il computer controlla immediatamente se la nave affondata ha una lunghezza minore della più piccola della flotta.

Se è così, allora le altre caselle di questa nave sono già state colpite e sono state erroneamente assegnate ad altra nave; il computer, tramite le eccezioni di cui parlerò in seguito, andrà a sistemare e aggiornare le navi.

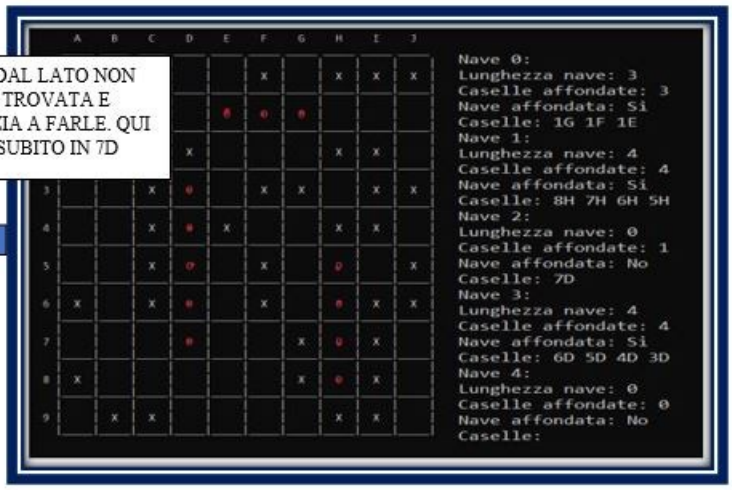
Altrimenti, il computer controlla se nella casella opposta a quella in cui è stata affondata la nave ci sono possibili mosse:

- In caso affermativo prova a fare una di quelle mosse
- In caso negativo, cerca di capire se ci sono delle navi vicine, in maniera da sistemare, tramite le eccezioni di cui parlerò in seguito, il numero di caselle affondate rispetto al numero di navi trovate.

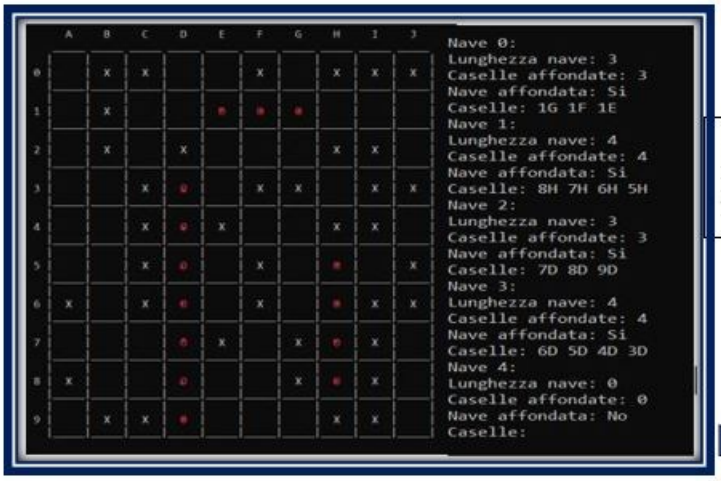
Se non è ancora stata fatta una mossa, il computer ricomincia i ragionamenti della mossa.



IL COMPUTER, QUINDI, GUARDA DAL LATO NON AFFONDATO DELL'ULTIMA NAVE TROVATA E TROVANDO POSSIBILI MOSSE INIZIA A FARLE. QUI CON LA PRIMA MOSSA COLPISCE SUBITO IN 7D

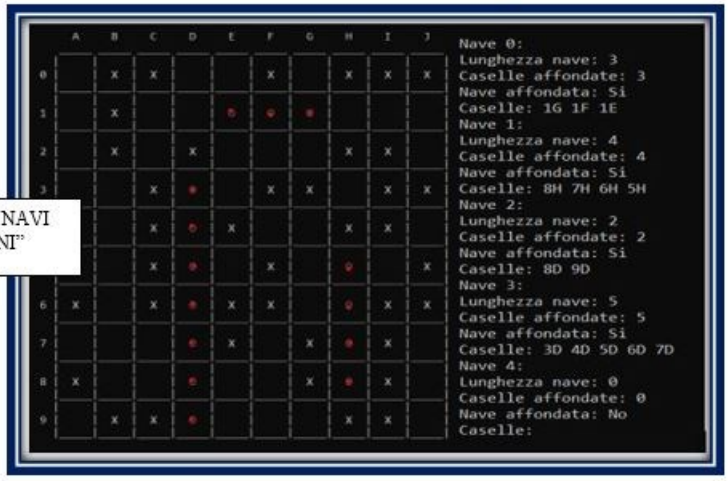


POI PROCEDE SULLA STESSA DIREZIONE



FINO A QUANDO NON OTTIENE UN ALTRO AFFONDATO IN 9D REGISTRANDO PROVVISORIAMENTE 2 NAVI DA 4 E 2 DA 3

INFINE SISTEMERA' LE NAVI TRAMITE LE "ECCEZIONI"



3.1.3 Eccezioni

Finora abbiamo analizzato le mosse che il computer esegue per colpire le navi; ma, come abbiamo visto, in certi casi è necessario andare a modificare delle navi già affondate in precedenza. Sono state quindi predisposte delle eccezioni che gestiscono gli scambi di caselle tra una nave e un'altra.

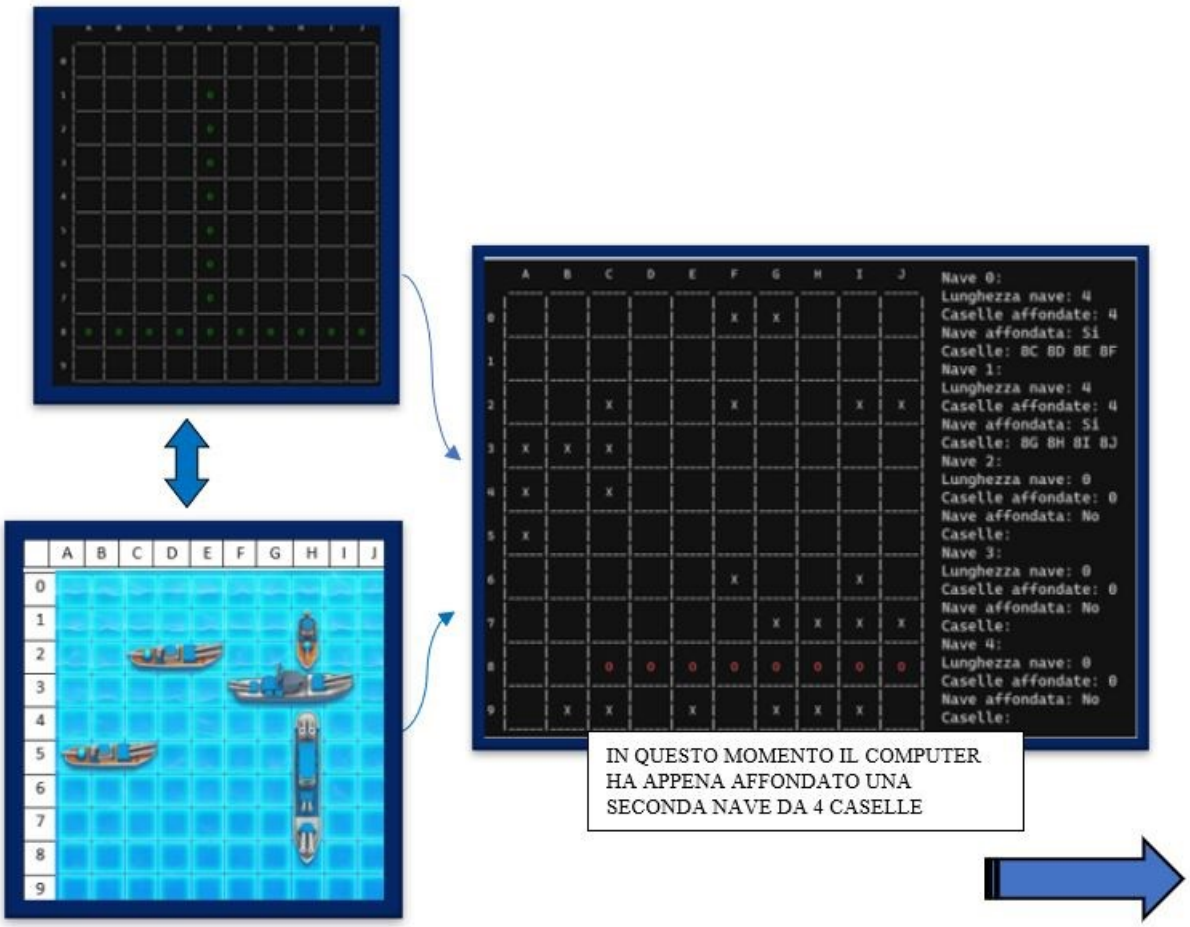
I tipi di eccezione sono principalmente tre:

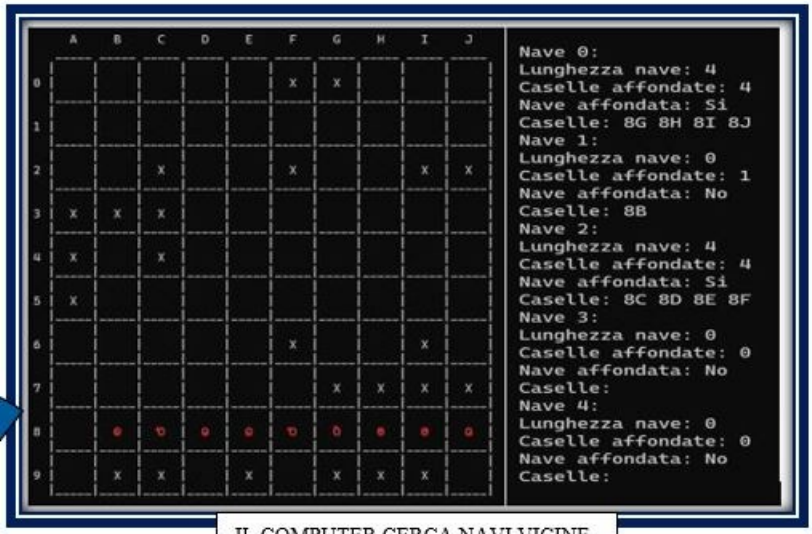
- EccezioneNaviInFila
- EccezioneNaviIncrociate
- SeparaNave

3.1.3.1 EccezioneNaviInFila

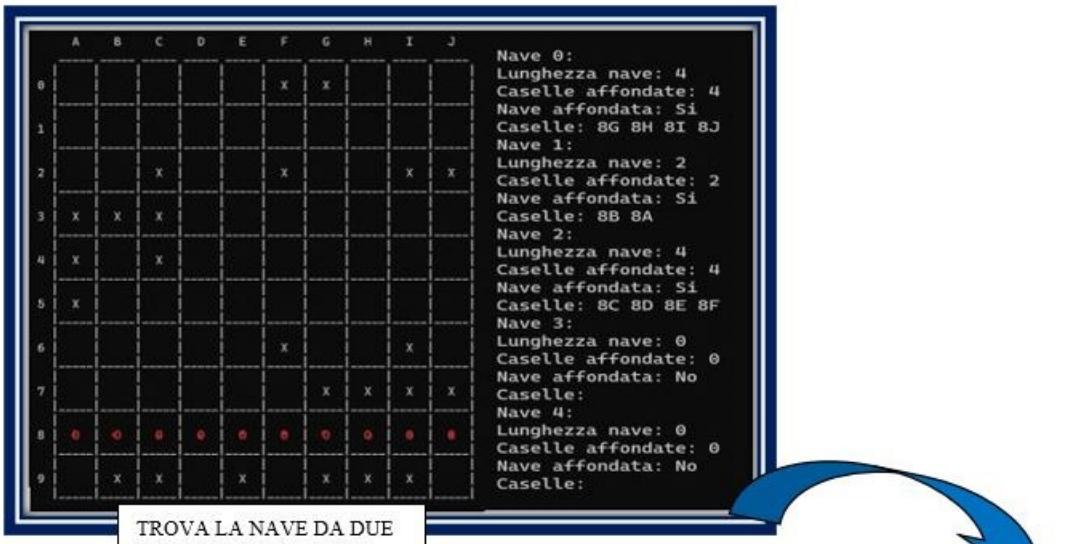
Questa eccezione viene lanciata nel momento in cui il computer riconosce di avere due navi vicine una di seguito all'altra. In questo caso, calcola il numero di caselle che queste due navi occupano in totale e cerca delle possibili combinazioni tra le navi ancora da trovare.

Se ne trova una possibile, sistema le due navi con il nuovo numero di caselle; altrimenti restituisce un valore booleano false per dire che questo tentativo di sistemare le navi non è andato a buon fine.

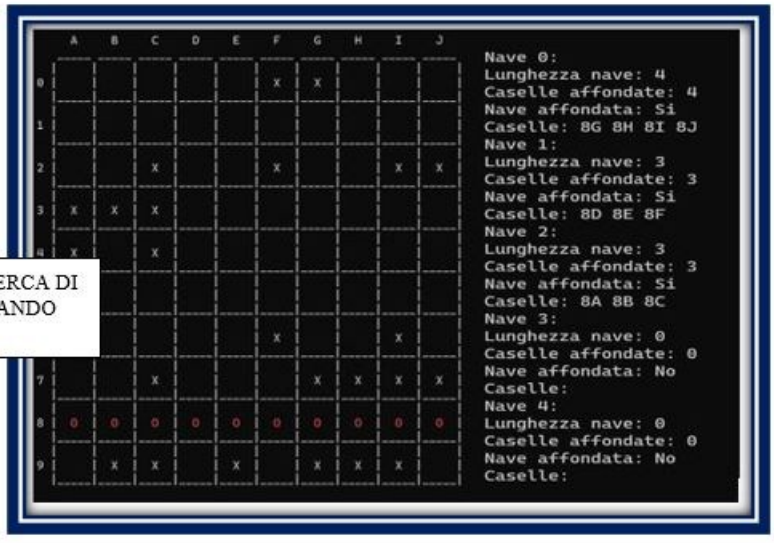




IL COMPUTER CERCA NAVI VICINE



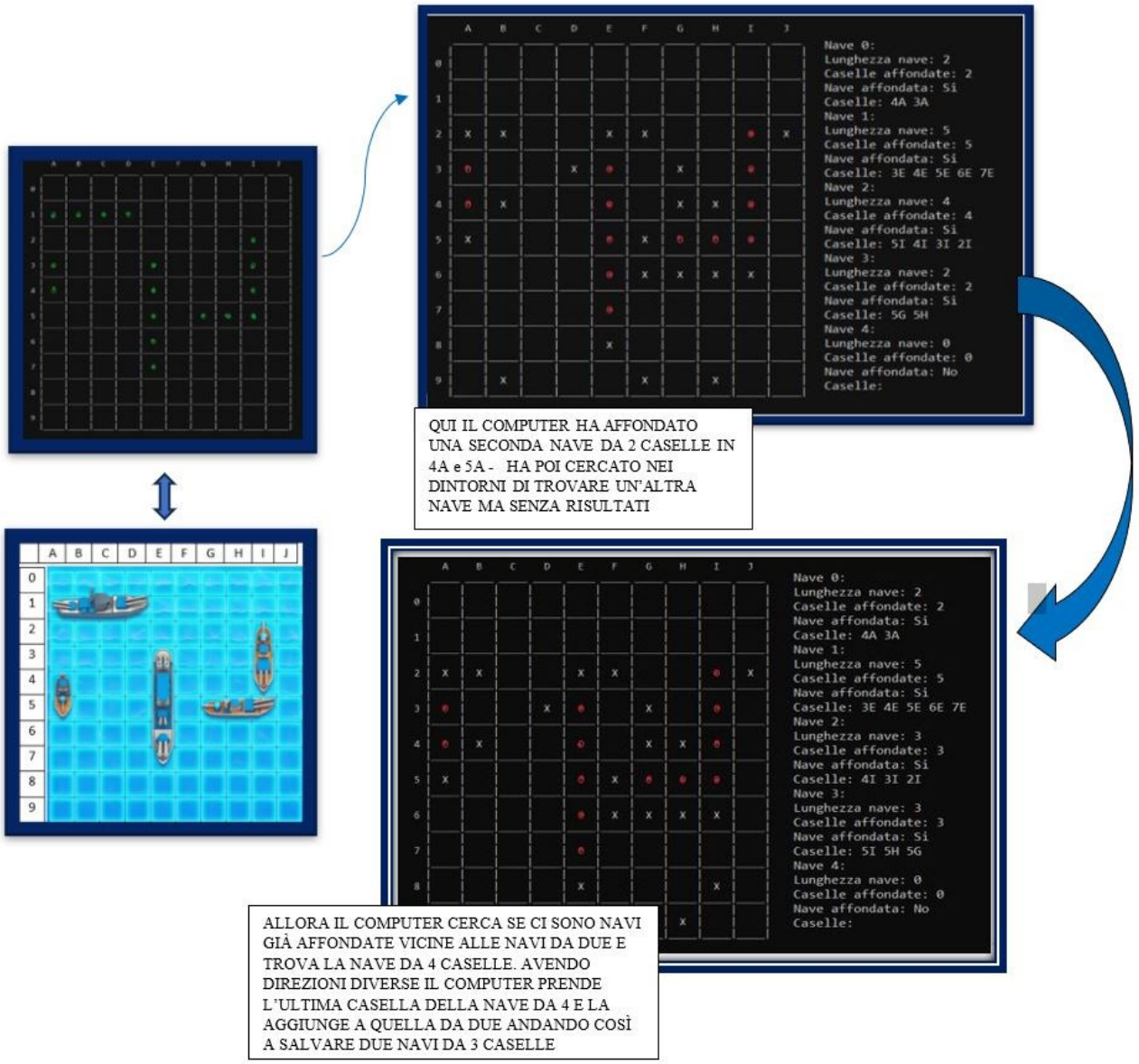
TROVA LA NAVE DA DUE
E LANCIA "ECCEZIONE
NAVI IN FILA"



SOMMA LE CASELLE E CERCA DI
COMBINARLE IDENTIFICANDO
LE 2 NAVI DA 3 CASELLE

3.1.3.2 EccezioneNaviIncrociate/EccezioneNaviIncrociateReverse

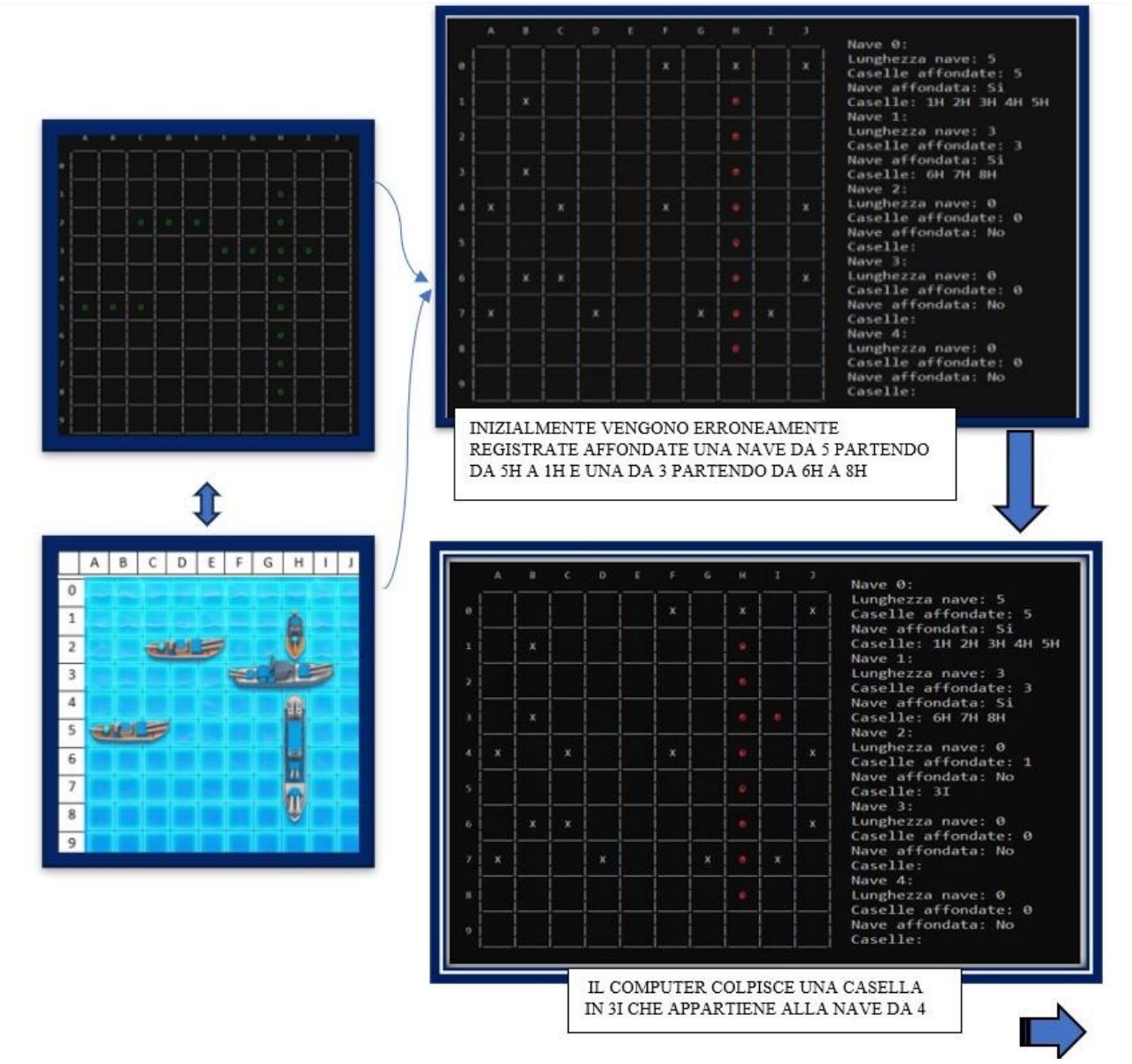
Questa eccezione viene lanciata nel momento in cui il computer riconosce di avere due navi vicine ma in direzione diverse. Quello che viene fatto in questo caso è prendere una delle due caselle che sono a contatto tra di loro e spostarla da una nave all'altra.

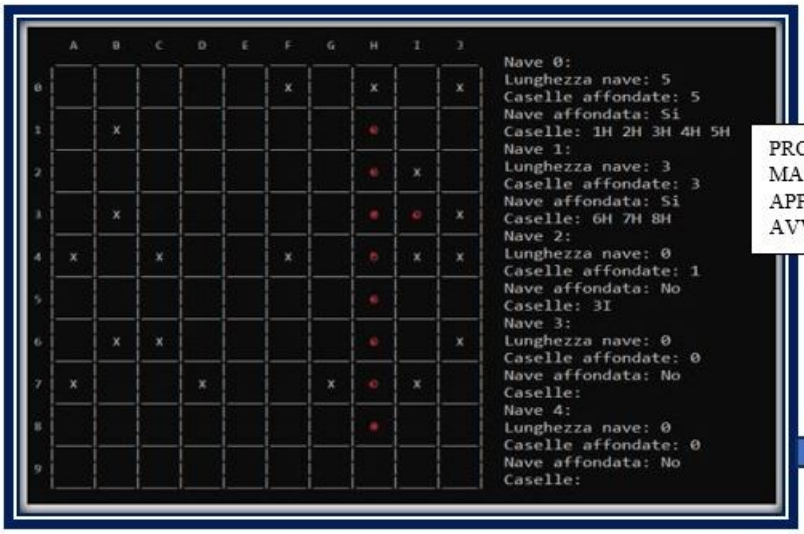


3.1.3.3 SeparaNave

Questa funzione viene lanciata quando, con una mossa incrociata, viene tolta da una nave una casella che non si trova agli estremi. Quello che il computer fa è controllare se ci sono accanto delle navi:

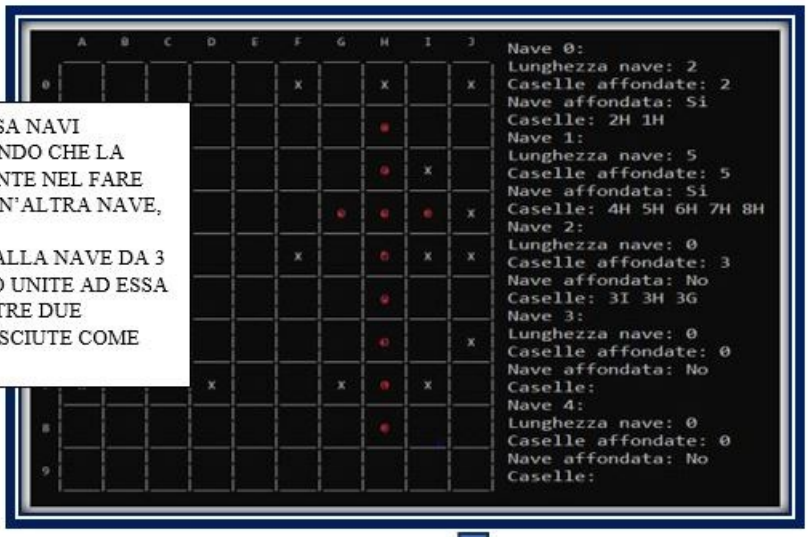
1. se presenti, accorpa le caselle separate a quest'altra nave, mentre quelle dal lato opposto vengono salvate come nave affondata aggiornandone la lunghezza.
2. In alternativa, salva le caselle rimaste dalla parte dove era stata affondata come una nave affondata aggiornandone la lunghezza, mentre le caselle dalla parte opposta vengono segnate come una nuova nave ancora da affondare.

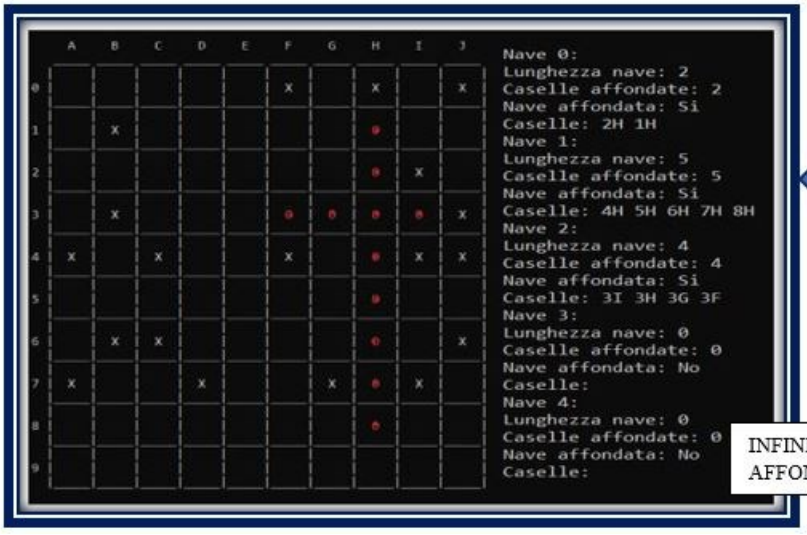




PROVA A COLPIRE NEI DINTORNI MA NON TROVA ALTRE CASELLE APPARTENENTI A NAVI AVVERSARIE

A QUESTO PUNTO COMPIE UNA MOSSA NAVI INCROCIATE E COLPISCE IN 3G. ESSENDO CHE LA CASELLA SALTATA PRECEDENTEMENTE NEL FARE LA MOSSA (3H) ERA NEL MEZZO DI UN'ALTRA NAVE, QUEST'ULTIMA VIENE SEPARATA.
LE DUE CASELLE CHE SONO VICINE ALLA NAVE DA 3 CASELLE GIÀ AFFONDATA VENGONO UNITE AD ESSA E SALVATE COME NAVE DA 5. LE ALTRE DUE CASELLE INVECE VENGONO RICONOSCIUTE COME UNA NAVE E QUINDI SALVATE.





INFINE PROCEDE CON LE MOSSE E AFFONDA ANCHE LA NAVE DA 4 CASELLE

3.2
Strutture di dati

3.2.1 Tipi strutturati

3.2.1.1 Coordinate

Serve per individuare una casella all'interno della griglia

```
typedef struct{
      int riga;/* riga della griglia */
      int colonna;/* colonna ella griglia */
}Coordinate;
```

3.2.1.2 Casella

Rappresenta una casella della griglia

```
typedef struct{
      Nave* pN;/*Puntatore alla nave contenuta*/
      bool contenuto;/*True se è presente un pezzo di una nave, false al-
      trimenti*/
      bool colpito;/*True se è stata colpita, false altrimenti*/
}Casella;
```

3.2.1.3 Nave

Rappresenta una nave

```
typedef struct{
      short lunghezza;/* lunghezza della nave */
      int caselleAff;/* numero di caselle affondate */
      bool affondata;/* valore booleano se true la nave è affondata */
      Coordinate* posizione;/* array contenente le coordinate di tutte le
      caselle della nave*/
}Nave;
```

3.2.1.4 InfoNavi

Contiene le informazioni riguardanti le navi della partita

```
typedef struct{
      int tipo; /*Contiene il tipo della nave, cioè la sua lunghezza*/
      int nNav; /*Contiene il numero di navi di questo tipo*/
}InfoNavi;
```

3.2.2 Array bidimensionale

Viene utilizzato esclusivamente per gestire le griglie: ognuna delle quattro griglie è formata da un array bidimensionale composto da oggetti di tipo casella con capienza variabile in quanto, come detto all'inizio, l'utente può liberamente scegliere il numero di righe e di colonne della griglia (restando nei limiti imposti).

3.2.3 Array monodimensionale

Viene utilizzato per gestire:

- Navi dei giocatori: sia le navi inserite dall'utente, sia quelle inserite dal computer vengono

salvate in un array monodimensionale composta da oggetti di tipo nave.

- Navi trovate dal computer: la necessità di salvare le navi trovate dal computer deriva dal fatto che potrebbe doverle modificare in base a quelle successive. Vengono pertanto salvati in un array come quello precedente.
- Informazioni sulle navi della partita: le informazioni contenenti la lunghezza e il numero di ogni tipo di nave viene salvato in un array monodimensionale composto da oggetti di tipo InfoNavi. Ci serve sia per inserire le navi all'inizio della partita sia poi per il computer che tiene traccia di che navi ha trovato e quali deve ancora trovare.
- Posizione delle navi: ogni nave contiene un array di oggetti di tipo coordinate che salva le caselle in cui è posizionata

4
INPUT E OUTPUT DEL PROBLEMA

4.1
Input

4.1.1 INPUT DA FILE: informazioni generali sulla partita

Il programma richiede in input da riga di comando un file contenente i dati della partita:

numero di righe

numero di colonne

numero di tipi diversi di navi (cioè navi di lunghezza diversa)

tipo di nave e numero di navi di quel tipo

4.1.2 INPUT DA TASTIERA: inserimento navi

Una volta che il programma inizia sarà richiesto all'utente di inserire le navi nella propria griglia. Per ogni nave sarà richiesto:

a. Prima casella della nave
b. Direzione della nave (orizzontale o verticale)
c. Numero di caselle della nave da posizionare da un lato (quelle rimanenti verranno posizionate automaticamente dal lato opposto)

Il programma effettua comunque un controllo sulle navi per verificare che non siano state inserite in maniera errata; in tal caso chiede di reinserirle.

4.2
Output

Una volta concluso l'inserimento di tutti i dati necessari alla partita, il programma inizierà il gioco alternando i turni tra l'utente e il computer. Durante il gioco saranno visualizzate a schermo le due griglie dell'utente insieme alle istruzioni su come procedere.

Alla fine della partita verranno stampate tutte le griglie e le posizioni delle navi del computer così da poterli controllare (se non ci si fida 😊).

5
CODICI SORGENTI

Il file BattagliaNavale.c contiene le funzioni per la preparazione della partita e lo svolgimento dei turni

```
#include "Mossa.h"
#include <stdio.h>
#include <unistd.h>
#include <time.h>
#include <assert.h>
#include <string.h>
#include <stdlib.h>


/*Funzione che ordina in modo decrescente l'array contenente le informazioni
sulle navi della partita

IP+OP naviPartita: puntatore all'array contenente le informazioni sulle navi
della partita
IP nTipi: dimensione del'array

*/
void ordinaInfoNavi(InfoNavi* naviPartita, int nTipi){
    int i, j;
    InfoNavi key;
    for (i = 1; i < nTipi; i++) {
        key = naviPartita[i];
        j = i - 1;
        while (j >= 0 && naviPartita[j].tipo < key.tipo) {
            naviPartita[j + 1] = naviPartita[j];
            j = j - 1;
        }
        naviPartita[j + 1] = key;
    }
}/* ordinaInfoNavi */


/*Funzione che libera lo spazio occupato da una griglia

IP griglia: array di cui liberare lo spazio
IP righe: capienza 1 dell'array

*/
void puliziaGriglia(Casella** griglia, int righe){
    int i;
    for(i = 0; i < righe; i++){
        free(griglia[i]);
    }
    free(griglia);
}/* puliziaGriglia */


/*Funzione che libera lo spazio in memoria occupato dalle griglie

IP my_griglia: griglia del giocatore
IP my_grigliaAvv: griglia avversaria del giocatore
IP c_griglia: griglia del computer
IP c_grigliaAvv: griglia avversaria del computer
IP righe: capienza 1 degli array
```

```
*/
void puliziaGriglie(Casella** my_griglia, Casella** my_grigliaAvv, Casella**
c_griglia, Casella** c_grigliaAvv, int righe){
    puliziaGriglia(my_griglia, righe);
    puliziaGriglia(my_grigliaAvv, righe);
    puliziaGriglia(c_griglia, righe);
    puliziaGriglia(c_grigliaAvv, righe);
}/* puliziaGriglie */


/*Funzione che libera lo spazio occupato da una nave e il suo array di coordina-
te

IP nave: array di cui liberare lo spazio
IP num_navi: capienza dell'array

*/
void puliziaNave(Nave* navi, int num_navi){
    int i;
    for(i = 0; i < num_navi; i++){
        free(navi[i].posizione);
    }
    free(navi);
}/* puliziaNave */


/*Funzione che libera lo spazio in memoria occupato dalle navi

IP my_navi: array delle navi del giocatore
IP c_navi: array delle navi del computer
IP c_navi_trovate: array delle navi ttrovate dal computer
IP num_navi: numero di navi degli array

*/
void puliziaNavi(Nave* my_navi, Nave* c_navi, Nave* c_navi_trovate, int
num_navi){
    puliziaNave(my_navi, num_navi);
    puliziaNave(c_navi, num_navi);
    puliziaNave(c_navi_trovate, num_navi);
}/* puliziaNavi */


/*Viene inserito da riga di comando il nome del file che contiene le specifiche
per la partita
    -numero linee e colonne della tabella
    -numero di tipi di navi
    -numero di navi di ciascun tipo e lunghezza di qeuste navi
*/
int main(int argc, char* argv[]){

    /*Dichiarazione variabili*/

    FILE* fIn; /* file da cui prendere le informazioni iniziali */
    Casella **my_griglia, **my_grigliaAvv;/* griglie giocatore */
    Casella **c_grigliaAvv, **c_griglia;/* griglie computer */
    int num_navi = 0, i, righe, colonne, nTipi; /* numero di navi della partita,
righe della griglia, colonne della griglia, diversi tipi di navi*/
    int numNaviC, numNaviG; /* numero di navi da colpire di giocatore e computer
*/
    int max_dimensione; /* contiene la dimensione massima tra righe e colonne */
    Nave *my_navi, *c_navi; /* array contenenti le navi del giocatore e del com-
puter */
```

```
    InfoNavi* naviPartita; /* array contenente le informazioni delle navi della
partita */
    Nave* comp_navi_trovate; /* array contenente le navi trovate dal computer
*/
    char pausa; /* variabile che serve per far fermare il programma */
    bool inserimento = false; /* indica se l'inserimento delle navi da parte del
giocatore Ã¨ avvenuto correttamente */
    bool turno = 0; /*0 --> turno giocatore | 1 --> turno computer*/

    /*Preparazione gioco*/

    if((fIn = fopen(argv[1], "r")) == NULL)
        return -1;
    fscanf(fIn,"%d", &righe);
    fscanf(fIn, "%d", &colonne);
    fscanf(fIn, "%d", &nTipi);
    naviPartita = malloc(sizeof(InfoNavi) * nTipi);
    assert(naviPartita != NULL);
    for(i = 0; i < nTipi; i++){
        fscanf(fIn, "%d", &naviPartita[i].tipo);
        fscanf(fIn, "%d", &naviPartita[i].nNav);
        num_navi += naviPartita[i].nNav;
    }
    fclose(fIn);

    ordinaInfoNavi(naviPartita, nTipi);
    stampaInfoPartita(naviPartita,righe,colonne,num_navi,nTipi);

    /* Preparazione griglie */
    while(!inserimento){
        my_griglia = allocaGriglia(righe, colonne);
        inizializzaGriglia(my_griglia,righe,colonne);
        my_navi = creaFlotta(naviPartita, nTipi, num_navi, righe, colonne);
        if(!inserisciNaviGriglia(my_griglia, my_navi, num_navi)){
            printf("INSERIMENTO NAVI ERRATO REINSERIRLE\n");
            free(my_griglia);
            free(my_navi);
        }else{
            inserimento = true;
            stampaGriglia(my_griglia, righe, colonne);
            stampaNavi(my_navi, num_navi);
        }
    }
    c_griglia = allocaGriglia(righe,colonne);
    inizializzaGriglia(c_griglia, righe, colonne);
    c_navi  = creaFlotta_computer(c_griglia, naviPartita, nTipi, num_navi, ri-
ghe, colonne);
    c_grigliaAvv = allocaGriglia(righe,colonne);
    inizializzaGriglia(c_grigliaAvv, righe, colonne);
    my_grigliaAvv = allocaGriglia(righe,colonne);
    inizializzaGriglia(my_grigliaAvv, righe, colonne);



    comp_navi_trovate = malloc(sizeof(Nave) * num_navi);
    assert(comp_navi_trovate != NULL);

    max_dimensione = dimensioneMax(righe, colonne);

    for(i = 0; i < num_navi; i++){
        inizializzaNaveIncognita(&comp_navi_trovate[i], max_dimensione);
    }
```

```
    numNaviC = numNaviG = num_navi;

    /*Inizio gioco*/
    printf("INIZIO GIOCO\n");
    scanf("%c", &pausa);
    stampaDoppiaGriglia(my_griglia,my_grigliaAvv,righe,colonne);

    while(numNaviG && numNaviC){
        /* Turno Computer */
        if(turno){
            scanf("%c", &pausa);
            printf("TURNO DEL COMPUTER\nPremi per continuare");
            scanf("%c", &pausa);
            if(eseguiMossa(c_grigliaAvv, my_griglia, comp_navi_trovate, naviPar-
tita, nTipi, righe, colonne, num_navi) == 2)
                numNaviG--;
            printf("\nNavi rimaste: %d\n", numNaviG);
            stampaDoppiaGriglia(my_griglia,my_grigliaAvv,righe,colonne);
        }else{
            /* Turno Giocatore */
            printf("E' IL TUO TURNO\nPremi per continuare");
            scanf("%c", &pausa);
            stampaGriglia(my_grigliaAvv, righe, colonne);
            if(mossaGiocatore(c_griglia, my_grigliaAvv, righe, colonne) == 2)
                numNaviC--;
            printf("\nNavi rimaste: %d\n", numNaviC);
            stampaDoppiaGriglia(my_griglia,my_grigliaAvv,righe,colonne);
        }
        turno = !turno;
    }

    /* Griglie finali */
    printf("\n\nMIE GRIGLIE\n\n");
    stampaDoppiaGriglia(my_griglia,my_grigliaAvv,righe,colonne);
    printf("\n\nCOMPUTER GRIGLIE\n\n");
    stampaDoppiaGriglia(c_griglia,c_grigliaAvv,righe,colonne);
    printf("Navi del computer:\n");
    stampaNavi(c_navi, num_navi);

    /*Pulizia*/
    free(naviPartita);
    puliziaGriglie(my_griglia, my_grigliaAvv, c_griglia, c_grigliaAvv, righe);
    puliziaNavi(my_navi, c_navi, comp_navi_trovate, num_navi);

    return 0;
}
```

Il file Nave.c contiene le funzioni per gestire le navi

```
#include "Griglia.h"
#include <stdio.h>
#include <assert.h>


/*Funzione per inizializzare una nave

IP+OP nave: nave da inizializzare
IP l: lunghezza della nave

*/
void inizializzaNave(Nave *n, int l){
```

```
    n->lunghezza = l;
    n->caselleAff = 0;
    n->affondata = false;
    n->posizione = malloc(sizeof(Coordinate) * l);
    assert(n->posizione != NULL);
}/* inizializzaNave */


/*Funzione che inizializza una nave incognita

IP+OP nave: puntatore alla nave da inizializzare
IP l: lunghezza dell'array delle posizioni della nave

*/
void inizializzaNaveIncognita(Nave *n, int l){
    n->lunghezza = 0;
    n->caselleAff = 0;
    n->affondata = false;
    n->posizione = malloc(sizeof(Coordinate) * l);
    assert(n->posizione != NULL);
}/* inizializzaNaveIncognita */


/*Funizone che dice se una nave Ã¨ affondata o meno

IP nave: puntaore alla nave di cui si vuole sapere se Ã¨ affondata
OP true se Ã¨ affondata, false altrimenti

*/
bool controlloAffondo(Nave* nave){
    if(nave->caselleAff == nave->lunghezza)
        nave->affondata = true;
    return nave->affondata;
}/* controlloAffondo */


/*Funzione che stampa le informazioni di una nave

IP nave: puntatore alla nave di cui stampare le indoformazione

*/
void stampaNave(const Nave* n){
    int i;
    printf("Lunghezza nave: %d\n", n->lunghezza);
    printf("Caselle affondate: %d\n", n->caselleAff);
    printf("Nave affondata: %s\n", n->affondata? "Si":"No");
    printf("Caselle: ");
    for(i = 0; i < n->lunghezza; i++){
        printf("%d%c ", n->posizione[i].riga, n->posizione[i].colonna + 'A');
    }
    printf("\n");
}/* stampaNave */

/*Funzione che stampa le informazioni di un array di navi

IP navi: array contenente le navi da stampare
IP nNavi: numero delle navi

*/
void stampaNavi(const Nave* navi, int nNavi){
    int i;
    for(i=0; i<nNavi; i++){
        printf("Nave %d:\n",i);
```

```
        stampaNave(&navi[i]);
    }
}/* stampaNavi */



/*Funzione che ordina un array di coordinate rispetto alle righe

IP+OP posizione: array di coordinate da ordinare
IP verso: verso in cui ordinarle
IP n: numero di elementi dell'array

*/
void insertionSortRighe(Coordinate posizione[], bool verso, short n){
    int i, j;
    Coordinate key;
    for (i = 1; i < n; i++) {
        key = posizione[i];
        j = i - 1;
        if(verso){
            while (j >= 0 && posizione[j].riga > key.riga) {
                posizione[j + 1] = posizione[j];
                j = j - 1;
            }
        }else{
            while (j >= 0 && posizione[j].riga < key.riga) {
                posizione[j + 1] = posizione[j];
                j = j - 1;
            }
        }
        posizione[j + 1] = key;
    }
}/* insertionSortRighe */


/*Funzione che ordina un array di coordinate rispetto alle colonne

IP+OP posizione: array di coordinate da ordinare
IP verso: verso in cui ordinarle
IP n: numero di elementi dell'array

*/
void insertionSortColonne(Coordinate posizione[], bool verso, short n){
    int i, j;
    Coordinate key;
    for (i = 1; i < n; i++) {
        key = posizione[i];
        j = i - 1;
        if(verso){
            while (j >= 0 && posizione[j].colonna > key.colonna) {
                posizione[j + 1] = posizione[j];
                j = j - 1;
            }
        }else{
            while (j >= 0 && posizione[j].colonna < key.colonna) {
                posizione[j + 1] = posizione[j];
                j = j - 1;
            }
        }
        posizione[j + 1] = key;
    }
}/* insertionSortColonne */
```

```
/*Funzione che trova direzione e verso di una nave

IP nave: puntatore alla nave di vui conoscere direzione e verso
IP+OP direzione: puntatore a una variabile booleana che conterrÃ  la direzione
della nave
IP+OP verso: puntatore a una variabile booleana che conterrÃ  la verso della na-
ve

*/
void trovaDirezioneVersoNave(const Nave* nave, bool *direzione, bool *verso){
    int ind_riga = nave->posizione[(nave->caselleAff)-1].riga - nave-
>posizione[(nave->caselleAff)-2].riga;
    int ind_colonna = nave->posizione[(nave->caselleAff)-1].colonna - nave-
>posizione[(nave->caselleAff)-2].colonna;

    if(ind_riga){
        *direzione = 0;
        if(ind_riga > 0)
            *verso = 1;
        else
            *verso = 0;
    }else{
        *direzione = 1;
        if(ind_colonna > 0)
            *verso = 1;
        else
            *verso = 0;
    }
}/* trovaDirezioneVersoNave */

/*Funzione che ordina le caselle di una nave

IP+OP nave: puntatore alla nave di cui ordinare le caselle

*/
void ordinaNave(Nave* nave){
    bool direzione;
    bool verso;
    trovaDirezioneVersoNave(nave, &direzione, &verso);

    if(direzione){
        insertionSortColonne(nave->posizione, verso, nave->lunghezza);
    }else{
        insertionSortRighe(nave->posizione, verso, nave->lunghezza);
    }
}/* ordinaNave */
```

Il file Nave.h contiene le dichiarazioni delle funzioni di Nave.c

```
#include <stdbool.h>
#include <stdlib.h>
#include <time.h>

typedef struct{
    int riga;/* riga della griglia */
    int colonna;/* colonna ella griglia */
}Coordinate;
```

```
typedef struct{
    short lunghezza;/* lunghezza della nave */
    int caselleAff;/* numero di caselle affondate */
    bool affondata;/* valore booleano se true la nave è affondata */
    Coordinate* posizione;/* array contenente le coordinate di tutte le caselle
della nave*/
}Nave;

void inizializzaNave(Nave*, int);

void inizializzaNaveIncognita(Nave*, int);

bool controlloAffondo(Nave*);

void stampaNavi(const Nave*, int);

void stampaNave(const Nave*);

void ordinaNave(Nave*);

void trovaDirezioneVersoNave(const Nave*, bool*, bool*);
```

Il file Griglia.c contiene le funzioni per gestire le griglie

```
#include <stdio.h>
#include <assert.h>
#include "Mossa.h"


/*Funzione che chiede in input la casella

IP+OP c: stringa che contiene la posizione della casella

*/
bool leggiCasella(char* c, int righe, int colonne){
    char c_temp;
    scanf("%s",c);
    if(c[0] > c[1]){
        c_temp = c[0];
        c[0] = c[1];
        c[1] = c_temp;
    }
    if(c[1] > 96 && c[1] < 123){
        c[1] = c[1] - 32;
    }
    if((c[0] < '0' || c[0] > ('0'+righe)) || (c[1] < 'A' || c[1] >
('A'+colonne))){
        printf("CASELLA INSERITA NON VALIDA\n");
        return false;
    }
    return true;
}/* leggiCasella */


/*Funizione che trova le coordinate della casella

IP c: puntatore alla stringa contenete la posizione della casella
OP cas: coordinate della casella
```

```
*/
Coordinate coordinateCasella(char* c){
    Coordinate cas;
    cas.colonna = (int)(c[1] - 'A');
    cas.riga = (int)(c[0] - '0');
    return cas;
}/* coordinateCasella */


/*Funzione che alloca lo spazio per una griglia

IP righe: numero di righe della parita
IP colonne: numero di colonne della partita
OP puntatore all'area di memoria allocata

*/
Casella** allocaGriglia(int righe, int colonne){
    int i;
    Casella** griglia = malloc(sizeof(Casella*) * righe);
    assert(griglia != NULL);

    for(i = 0; i < righe; i++){
        griglia[i] = malloc(sizeof(Casella) * colonne);
        assert(griglia[i] != NULL);
    }
    return griglia;
}/* allocaGriglia */


/*Funizone che inizializza una griglia

IP griglia: griglia da inizializzare
IP righe: numero di righe della parita
IP colonne: numero di colonne della partita

*/
void inizializzaGriglia(Casella** griglia,int righe, int colonne){
    int i,j;

    for(i = 0; i < righe; i++){
        for(j = 0; j < colonne; j++){
            griglia[i][j].pN = NULL;
            griglia[i][j].colpito = 0;
            griglia[i][j].contenuto = ACQUA;
        }
    }
}/* inizializzaGriglia */

/*--------------FUNZIONI STAMPA GRIGLIA-----------------------------*/

/*Funzione che stampa il contenuto della casella

IP a: casella da stampare

*/
void stampaCasella(const Casella* a){
    if(!a->contenuto && a->colpito) printf("X");
    if(!a->contenuto && !a->colpito) printf(" ");
    if(a->contenuto && a->colpito) printf("\x1b[31mO\x1b[0m");
    if(a->contenuto && !a->colpito) printf("\x1b[32mO\x1b[0m");
}/*stampaCasella*/

/*Funzione che stampa la riga superiore della griglia
```

```
IP colonne: numero di colonne della partita

*/
void stampaPrimaRiga(int colonne){
    int i;

    printf("                          ");
    for(i = 0; i < colonne; i++)
        printf("_____ ");
    printf("\n");
}/*stampaPrimaRiga*/

/*Funzione che stampa la prima riga vuota

IP colonne: numero di colonne della partita

*/
void stampaRigaVuota(int colonne){
    int i;
    printf("                         |");
    for(i = 0; i < colonne; i++)
        printf("     |");
    printf("\n");
}/*stampaRigaVuota*/


/*Funzione che stampa la riga centrale della casella

IP griglia: griglia da stampare
IP colonne: numero di colonne della griglia di gioco
IP i: numero della riga

*/
void stampaRigaContenuti(Casella** griglia,int colonne, int i){
    int j;
    printf("                     %2d |",i);
    for(j = 0; j < colonne; j++){
        printf("  ");
        stampaCasella(&griglia[i][j]);
        printf("  |");
    }
    printf("\n");
}/*stampaRigaContenuti*/

/*Funzione che stampa l'ultima riga della casella

IP colonne: numero di colonne della partita

*/
void stampaRigaConclusiva(int colonne){
    int i;
    printf("                         |");
    for(i = 0; i < colonne; i++)
        printf("_____|");
    printf("\n");
}/*stampaRigaConclusiva*/


/*Funzione che stampa l'ultima riga contenente le indicazioni delle colonne

IP colonne: numero di colonne della partita
```

```
*/
void stampaUltimaRiga(int colonne){
    int i;
    printf("                      ");
    for(i = 0; i < colonne; i++)
        printf("     %c", 'A'+i);
    printf("\n");
}/* stampaUltimaRiga */


/*Funzione che stampda la griglia

IP griglia: griglia da stampare
IP righe: numero di righe della griglia di gioco
IP colonne: numero di colonne della griglia di gioco

*/
void stampaGriglia(Casella** griglia, int righe, int colonne){
    int i;
    stampaPrimaRiga(colonne);
    for(i = 0; i < righe; i++){
        stampaRigaVuota(colonne);
        stampaRigaContenuti(griglia,colonne,i);
        stampaRigaConclusiva(colonne);
    }
    stampaUltimaRiga(colonne);
}/*stampaGriglia*/


/*----------------------FUNZIONI STAMPA DOPPIA GRIGLIA----------------------*/


/*Funzione che stampa la riga superiore della griglia

IP colonne: numero di colonne della partita

*/
void stampaDoppiaPrimaRiga(int colonne){
    int i;
    printf("    ");
    for(i = 0; i < colonne; i++)
        printf("_____ ");
    printf("    ");
    for(i = 0; i < colonne; i++)
        printf("_____ ");
    printf("\n");
}/*stampaPrimaRiga*/

/*Funzione che stampa la prima riga vuota

IP colonne: numero di colonne della partita

*/
void stampaDoppiaRigaVuota(int colonne){
    int i;
    printf("   |");
    for(i = 0; i < colonne; i++)
        printf("     |");
    printf("   |");
    for(i = 0; i < colonne; i++)
        printf("     |");
    printf("\n");
}/*stampaRigaVuota*/
```

```
/*Funzione che stampa la riga centrale della casella

IP griglia: griglia da stampare
IP griglia_avv: griglia da stampare
IP colonne: numero di colonne della griglia di gioco
IP i: numero della riga

*/
void stampaDoppiaRigaContenuti(Casella** griglia, Casella **griglia_avv,int co-
lonne, int i){
    int j;
    printf("%2d |",i);
    for(j = 0; j < colonne; j++){
        printf("  ");
        stampaCasella(&griglia[i][j]);/**(*a + j + DIMENSIONE*i)*/
        printf("  |");
    }
    printf("%2d |",i);
    for(j = 0; j < colonne; j++){
        printf("  ");
        stampaCasella(&griglia_avv[i][j]);/**(*a + j + DIMENSIONE*i)*/
        printf("  |");
    }
    printf("\n");
}/*stampaRigaContenuti*/


/*Funzione che stampa l'ultima riga della casella

IP colonne: numero di colonne della partita

*/
void stampaDoppiaRigaConclusiva(int colonne){
    int i;
    printf("   |");
    for(i = 0; i < colonne; i++)
        printf("_____|");
    printf("   |");
    for(i = 0; i < colonne; i++)
        printf("_____|");
    printf("\n");
}/*stampaRigaConclusiva*/


/*Funzione che stampa l'ultima riga contenente le indicazioni delle colonne

IP colonne: numero di colonne della partita

*/
void stampaDoppiaUltimaRiga(int colonne){
    int i;
    printf(" ");
    for(i = 0; i < colonne; i++)
        printf("     %c", 'A'+i);
    printf("    ");
    for(i = 0; i < colonne; i++)
        printf("     %c", 'A'+i);
    printf("\n");
}/* stampaDoppiaUltimaRiga */
```

```
/*Funzione che stampda due griglie

IP griglia: griglia da stampare
IP griglia_avv: griglia da stampare
IP righe: numero di righe della griglia di gioco
IP colonne: numero di colonne della griglia di gioco

*/
void stampaDoppiaGriglia(Casella** griglia, Casella** griglia_avv, int righe,
int colonne){
    int i;
    stampaDoppiaPrimaRiga(colonne);
    for(i = 0; i < righe; i++){
        stampaDoppiaRigaVuota(colonne);
        stampaDoppiaRigaContenuti(griglia, griglia_avv, colonne,i);
        stampaDoppiaRigaConclusiva(colonne);
    }
    stampaDoppiaUltimaRiga(colonne);
}/*stampaGriglia*/



/*--------------------FUNZIONI INSERIMENTO NAVI----------------------------*/

/*Funzione che inserisce una nave nella griglia

IP+OP griglia: array bidimensionale in cui aggiungere le navi
IP nave: nave da inserire

*/
bool inserisciNaveGriglia(Casella** griglia, Nave* nave){
    int i;
    for(i = 0; i < nave->lunghezza; i++){
        if(griglia[nave->posizione[i].riga][nave-
>posizione[i].colonna].contenuto)
            return false;
        griglia[nave->posizione[i].riga][nave->posizione[i].colonna].contenuto =
NAVE;
        griglia[nave->posizione[i].riga][nave->posizione[i].colonna].pN = nave;
    }
    return true;
}/* inserisciNaveGriglia */


/*Funzione che inserisce le navi nella griglia

IP griglia: array bidimensionale in cui aggiungere le navi
IP navi: array di navi da inserire
IP numNavi: numero delle navi

*/
bool inserisciNaviGriglia(Casella** griglia, Nave* navi, int numNavi){
    int i;
    for(i = 0; i < numNavi; i++){
        if(!inserisciNaveGriglia(griglia, &navi[i]))
            return false;
    }
    return true;
}/* inserisciNaviGriglia */
```

Il file Griglia.h contiene le dichiarazioni delle funzioni di Griglia.c

```
#include "Nave.h"
#define ACQUA 0
#define NAVE 1
#define L 3

typedef struct{
    Nave* pN;/*Puntatore alla nave contenuta*/
    bool contenuto;/*True se è presente un pezzo di una nave, false altrimenti*/
    bool colpito;/*True se è stata colpita, false altrimenti*/
}Casella;

typedef struct{
    int tipo; /*Contiene il tipo della nave, cioè la sua lunghezza*/
    int nNav; /*Contiene il numero di navi di questo tipo*/
}InfoNavi;

Casella** allocaGriglia(int,int);

void inizializzaGriglia(Casella**, int, int);

void stampaGriglia(Casella**, int, int);

void stampaDoppiaGriglia(Casella**, Casella**, int, int);

void stampaPrimaRiga(int);

void stampaCasella(const Casella*);

Coordinate coordinateCasella(char*);

bool leggiCasella(char*, int, int);

bool inserisciNaviGriglia(Casella**, Nave*, int);
```

Il file Generale.c contiene le funzioni di inserimento navi e altre funzioni utili

```
#include "Mossa.h"
#include <stdio.h>
#include <assert.h>


/*Funzione che trova ls dimensione piÃ¹ grande

IP righe: dimensione 1
IP colonne: dimensione 2
OP ritorna la dimensione piÃ¹ grande

*/
int dimensioneMax(int righe, int colonne){
    if(righe > colonne)
        return righe;
    else
        return colonne;
}/* dimensioneMax */
```

```
/*Funzione che inserisce le caselle della nave

IP l: lunghezza della nave
IP+OP nave: nave di cui inserire la posizione

*/
bool inserisciNave(int l, Nave* nave, int righe, int colonne){
    int i;
    int direzione;
    int n1;
    Coordinate cas;
    char c[3];
    do{
        printf("Inserisci la prima casella: ");
        leggiCasella(c, righe, colonne);
        printf("%s\n", c);
        cas = coordinateCasella(c);
    }while(!isValid(&cas, righe, colonne));
    nave->posizione[0] = cas;
    printf("Scegli la direzione in cui mettere il resto delle caselle della nave
(1 = orizzontele | 0 = verticale): ");
    scanf("%d", &direzione);
    if(direzione)
        printf("quante caselle vuoi mettere a destra? ");
    else
        printf("quante caselle vuoi mettere in basso? ");
    scanf("%d", &n1);
    while(n1 > (l-1)){
        printf("Troppe caselle reinserisci: ");
        scanf("%d", &n1);
    }
    for(i = 1; i <= n1; i++){
        aumentaMossa(&cas, (bool)direzione, true, 1);
        stampa_mossa(&cas);
        if(!isValid(&cas, righe, colonne))
            return false;
        nave->posizione[i] = cas;
    }
    cas = nave->posizione[0];
    for(i = n1+1; i < l; i++){
        aumentaMossa(&cas, (bool)direzione, false, 1);
        stampa_mossa(&cas);
        if(!isValid(&cas, righe, colonne))
            return false;
        nave->posizione[i] = cas;
    }

    return true;
}/* inserisciNave */



/*Funzione che inserisce tutte le navi dello stesso tipo

IP naviPartita: array contente le informazioni delle navi della partita
IP nave: puntatore alla nave da inserire
OP puntatore alla prosssima nave da inserire

*/
Nave* inserisciNaviTipo(InfoNavi* naviTipo, Nave* nave, int righe, int colonne){
    int i;
    for(i = 0; i < naviTipo->nNav; i++){
        printf("Inserisci nave da %d:\n", naviTipo->tipo);
```

```
        inizializzaNave(nave, naviTipo->tipo);
        if(!inserisciNave(naviTipo->tipo, nave, righe, colonne)){
            printf("Errore inseirmento nave, reinserire\n");
            i--;
        }else{
            nave++;
        }
    }
    return nave;
}/* inserisciNaveTipo */


/*Funzione che crea la flotta del computer e la sua griglia

IP naviPartita: array contente le informazioni delle navi della partita
IP nTipi: numero di tipi di navi della partita
IP numeroNavi: numero di navi della partita
OP array contenente le navi del giocatore

*/
Nave* creaFlotta(InfoNavi* naviPartita, int nTipi, int numeroNavi, int righe,
int colonne){
    int i;
    Nave* cp_navi;
    Nave* navi = malloc(sizeof(Nave) * numeroNavi);
    assert(navi != NULL);
    cp_navi = navi;
    for(i = 0; i < nTipi; i++){
        cp_navi = inserisciNaviTipo(&naviPartita[i], cp_navi, righe, colonne);
    }
    return navi;
}/* creaFlotta */


/*----------------------FLOTTA COMPUTER-----------------------------*/


/*Funzione che controlla se Ã¨ possibile inserire una nave nella direzione scel-
ta

IP comp_griglia: array bidimensionale contenente la griglia in cui il computer
metterÃ  le sue navi
IP direzione: direzione in cui si vorrebbe mettere la nave
IP cas: puntatore alla prima casella della nave
IP l: lunghezza della nave
IP righe: numero delle righe della griglia di gioco
IP colonne: numero di colonne della griglia di gioco

*/
bool controlloDirezione(Casella** comp_griglia, bool direzione, Coordinate* cas,
int l, int righe, int colonne){
    int caselle_libere = 1;
    int i = 1;
    bool v1 = true, v2 = true;
    while(caselle_libere < l && (v1 || v2)){
        if(direzione){
            if(v1){
                cas->colonna+=i;
                (isValid(cas, righe,colonne) && !comp_griglia[cas->riga][cas-
>colonna].contenuto)? caselle_libere++ : (v1 = false);
                cas->colonna-=i;
            }if(v2){
                cas->colonna-=i;
```

```
                (isValid(cas, righe,colonne) && !comp_griglia[cas->riga][cas-
>colonna].contenuto)? caselle_libere++ : (v2 = false);
                cas->colonna+=i;
            }
        }else{
            if(v1){
                cas->riga+=i;
                (isValid(cas, righe,colonne) && !comp_griglia[cas->riga][cas-
>colonna].contenuto)? caselle_libere++ : (v1 = false);
                cas->riga-=i;
            }if(v2){
                cas->riga-=i;
                (isValid(cas, righe,colonne) && !comp_griglia[cas->riga][cas-
>colonna].contenuto)? caselle_libere++ : (v2 = false);
                cas->riga+=i;
            }
        }
        i++;
    }
    if(caselle_libere >= l)
        return true;
    else
        return false;
}/* controlloDirezione */


/*Funzione che inserisce unanave del computer

IP+OP comp_griglia: array bidimensionale contenente la griglia in cui il compu-
ter metterÃ  le sue navi
IP+OP nave: puntatore alla nave da inserire
IP l: lunghezza della nave da inserire
IP righe: numero delle righe della griglia di gioco
IP colonne: numero di colonne della griglia di gioco

*/
void inserisciNave_computer(Casella** comp_griglia, Nave* nave, int l, int ri-
ghe, int colonne){
    Coordinate cas;
    int i = 0;
    bool direzione, verso;
    srand(time(NULL));
    do{
        cas.riga = rand() % righe;
        cas.colonna = rand() % colonne;
        direzione = rand() % 2;
        if(!controlloDirezione(comp_griglia, direzione, &cas, l, righe, colon-
ne))
            direzione = !direzione;
    }while(!(isValid(&cas, righe,colonne) &&
!comp_griglia[cas.riga][cas.colonna].contenuto && controlloDirezio-
ne(comp_griglia, direzione, &cas, l, righe, colonne)));
    verso = rand() % 2;
    do{
        nave->posizione[i] = cas;
        comp_griglia[cas.riga][cas.colonna].contenuto = true;
        comp_griglia[cas.riga][cas.colonna].pN = nave;
        aumentaMossa(&cas, direzione, verso, 1);
        i++;
    }while(i<l && isValid(&cas, righe, colonne) &&
!comp_griglia[cas.riga][cas.colonna].contenuto);
    if(i<l){
        verso = !verso;
```

```
        aumentaMossa(&cas, direzione, verso, i+1);
        do{
            nave->posizione[i] = cas;
            comp_griglia[cas.riga][cas.colonna].contenuto = true;
            comp_griglia[cas.riga][cas.colonna].pN = nave;
            if(direzione){
                (verso)? cas.colonna++:cas.colonna--;
            }else{
                (verso)? cas.riga++:cas.riga--;
            }
            i++;
        }while(i<l && isValid(&cas, righe, colonne) &&
!comp_griglia[cas.riga][cas.colonna].contenuto);
    }
}/* inserisciNave_computer */

/*Funzione che inserisce tutte le navi del computer dello stesso tipo

IP comp_griglia: array bidimensionale contenente la griglia in cui il computer
metterÃ  le sue navi
IP naviPartita: array contente le informazioni delle navi della partita
IP nave: puntatore alla nave da inserire
IP righe: numero delle righe della griglia di gioco
IP colonne: numero di colonne della griglia di gioco
OP puntatore alla prosssima nave da inserire

*/
Nave* inserisciNaviTipo_computer(Casella** comp_griglia, InfoNavi* naviTipo, Na-
ve* nave, int righe, int colonne){
    int i;
    for(i = 0; i < naviTipo->nNav; i++){
        inizializzaNave(nave, naviTipo->tipo);
        inserisciNave_computer(comp_griglia, nave, naviTipo->tipo, righe, colon-
ne);
        nave++;
    }
    return nave;
}/* inserisciNaveTipo_computer */


/*Funzione che crea la flotta del computer e la sua griglia

IP comp_griglia: array bidimensionale contenente la griglia in cui il computer
metterÃ  le sue navi
IP naviPartita: array contente le informazioni delle navi della partita
IP nTipi: numero di tipi di navi della partita
IP numeroNavi: numero di navi della partita
IP righe: numero delle righe della griglia di gioco
IP colonne: numero di colonne della griglia di gioco
OP array contenente le navi del computer

*/
Nave* creaFlotta_computer(Casella** comp_griglia, InfoNavi* naviPartita, int
nTipi, int numeroNavi, int righe, int colonne){
    int i;
    Nave* cp_navi;
    Nave* navi = malloc(sizeof(Nave) * numeroNavi);
    assert(navi != NULL);
    cp_navi = navi;
    for(i = 0; i < nTipi; i++){
        cp_navi = inserisciNaviTipo_computer(comp_griglia, &naviPartita[i],
cp_navi, righe, colonne);
    }
```

```
    return navi;
}/* creaFlotta_computer */


/*Funzione che stampa le informazioni della partita attuale

IP naviPartita: array contenente le informazioni delle navo di questa partita
IP righe: numero delle righe della griglia di gioco
IP colonne: numero di colonne della griglia di gioco
IP nNavi: numero di navi totali della partita
IP nTipi: numero di tipi di navi della partita

*/
void stampaInfoPartita(InfoNavi* naviPartita,int righe, int colonne, int nNavi,
int nTipi){
    int i;
    printf("\n\nINFORMAZIONI PARTITA:\n\n");
    printf("    DIMENSIONE GRIGLIA: %dx%d\n", righe,colonne);
    printf("    SIMBOLI NELLA GRIGLIA:\n");
    printf("        X = casella acqua colpita\n");
    printf("        \x1b[31mO\x1b[0m = casella con nave colpita\n");
    printf("        \x1b[32mO\x1b[0m = casella con nave\n");
    printf("    NUMERO TOTALE DI NAVI: %d\n",nNavi);
    for(i = 0; i < nTipi; i++)
        printf("        %d navi di lunghezza %d casel-
le\n",naviPartita[i].nNav,naviPartita[i].tipo);
    printf("\n\n");
}/* stampaInfoPartita*/
```

Il file Generale.h contiene le dichiarazioni delle funzioni di Generale.c

```
#include "Griglia.h"

int dimensioneMax(int, int);

void stampaInfoPartita(InfoNavi* ,int ,int ,int ,int);

Nave* creaFlotta(InfoNavi* , int , int, int, int);

Nave* creaFlotta_computer(Casella**, InfoNavi*, int, int , int , int);
```

Il file Mossa.c contine le funzioni per il controllo sulle mosse, aggiornamento dei risultati ed esecuzione mossa

```
#include <stdio.h>
#include <assert.h>
#include <math.h>
#include "Mossa.h"


/*-------------------------SEZIONE GENERALE--------------------------------*/

/* Funzione che stampa a video una mossa

IP mossa: puntatore alla mossa da stampare
```

```
*/
void stampa_mossa(const Coordinate* mossa){
    printf("%d%c\n", mossa->riga, mossa->colonna+'A');
}/*stampa_mossa*/

/*Funzione che cerca una nave di lunghezze uguale a $caselle_affondate

IP navi_trovate: array che contiene le navi giÃ  affondate dal computer
IP count_navi_trovate: numero di navi affondate finora
IP caselle_affondate: nuero di caselle affondate
IP indiceNave: indice della nave precedente che aveva la stessa lunghezza ma
senza possibili mosse
OP i: indice della nave con la stessa lunghezza o -1 nel caso in cui non si tro-
va

*/
int cercaNave(const Nave* navi_trovate, short count_navi_trovate, short casel-
le_affondate, int indiceNave){
    int i;
    for(i = count_navi_trovate-1; i >= 0; i--){
        if(navi_trovate[i].lunghezza == caselle_affondate){
            if(i < indiceNave)
                return i;
        }
    }

    return -1;
}/*cercaNave*/


/*Funzione che cerca in quale posizione all'interno dell'array posizione di $na-
ve di trovi la casella $mossa

IP nave: puntatore alla nave in cui cercare la casella
IP mossa: puntatore alla casella da trovare
OP i: indice della posizione in cui si trova la casella o -1 se non appartiene a
questa nave

*/
int cercaPosizione(const Nave* nave, const Coordinate* mossa){
    int i;
    for(i = 0; i < nave->caselleAff; i++){
        if(nave->posizione[i].colonna == mossa->colonna && nave-
>posizione[i].riga == mossa->riga)
            return i;
    }
    return -1;
}/*cercaPosizione*/

/*Funzione che cerca a quale nave appartine una determinata casella

IP navi_trovate: array che contiene le navi trovate e affondate dal computer
IP count_navi_trovate: numero di navi affondate finora
IP navi_attuale: numero di navi trovate ma ancora da affondare
IP indiceNave: indice della nave precedente che aveva la stessa lunghezza ma
senza possibili mosse
IP+OP indicePosizione: indice della posizione in cui si trova la casella
nell'array posizione di $navi_trovate[indiceNave]
IP mossa: casella da cercare
OP indice della nave contenente quasta casella

*/
```

```
int cercaNavePosizione(const Nave* navi_trovate, short count_navi_trovate, short
navi_attuali, int* indiceNave, int*indicePosizione, const Coordinate* mossa){
    int i;
    for(i = 0; i < (count_navi_trovate + navi_attuali); i++){
        if(i == *indiceNave)/* non cerco nel caso l'indice della nave corrispon-
da a quello precedente*/
            continue;
        *indicePosizione = cercaPosizione(&navi_trovate[i], mossa);
        if(*indicePosizione != -1)
            return i;
    }
    return -1;
}/*cercaNavePosizione*/


/*Funzione che aggiunge una nave che prima era stata interpretata erroneamente

IP+OP naviPartita: puntatore all'array contenente le informazioni sulle navi
della partita
IP caselle_affondate: numero di caselle affondate della nave
IP nTipi: grandezza dell'array

*/
void aggiungiNavePartita(InfoNavi* naviPartita, int caselle_affondate, int nTi-
pi){
    int i;
    for(i = 0; i < nTipi; i++){
        if(naviPartita[i].tipo == caselle_affondate){
            naviPartita[i].nNav++;
        }
    }
}/* aggiungiNavePartita */


/*Funzione che trova direzione e verso a partire da due coordinate e ritorna
quanto distano nella direzione trovata

IP  cas1: puntatore alla prima coordinata
IP  cas2: puntatore alla seconda coordinata
IP + OP direzione: puntatore alla direzione da restituire
IP + OP verso: puntatore al verso da restituire
OP  distanza tra le due caselle in base alla direzione

*/
int trovaDirezioneVerso(const Coordinate* cas1,const Coordinate* cas2, bool
*direzione, bool *verso){
    int ind_riga = cas2->riga - cas1->riga;
    int ind_colonna = cas2->colonna - cas1->colonna;

    if(ind_riga){
        *direzione = 0;
        if(ind_riga > 0)
            *verso = 1;
        else
            *verso = 0;
        return ind_riga;
    }else{
        *direzione = 1;
        if(ind_colonna > 0)
            *verso = 1;
        else
            *verso = 0;
        return ind_colonna;
```

```
    }
}/* trovaDirezioneVerso */

/*Funzione di scambio due coordinate

IP + OP cas1: prima coordinata da scambiare
IP + OP cas2: seconda coordinata da scambiare

*/
void scambiaCaselle(Coordinate* cas1, Coordinate* cas2){
    Coordinate cas_temp;
    cas_temp = *cas1;
    *cas1 = *cas2;
    *cas2 = cas_temp;
}/* scambiaCaselle */

/*Funzione che scambia due navi

IP+OP nav1: prima nave da scambiare
IP+OP nav2: seconda nave da scambiare

*/
void scambiaNavi(Nave* nav1, Nave* nav2){
    Nave nav_temp;
    nav_temp = *nav1;
    *nav1 = *nav2;
    *nav2 = nav_temp;
}/* scambiaNavi */

/*Funzione che aumenta o diminuisce la mossa di una quantitÃ  variabile in base
a direzione e verso

IP+OP new_mossa: puntatore alla nuova mossa
IP direzione: variabile che contiene la direzione in cui spostare la mossa
IP verso: variabile che contiene il verso in cui spostare la mossa
IP n: numero di cui spostare la mossa

*/
void aumentaMossa(Coordinate* new_mossa, bool direzione, bool verso, int n){
    if(direzione){
        if (verso)
            new_mossa->colonna += n;
        else
            new_mossa->colonna -= n;
    }else{
        if (verso)
            new_mossa->riga += n;
        else
            new_mossa->riga -= n;
    }
}/* aumentaMossa */


/*------------------------SEZIONE CONTROLLI---------------------------------*/

/*Funzione che controlla se una mossa Ã¨ valida

IP m: mossa da controllare
IP righe: numero di righe della griglia
IP colonne: numero di colonne della griglia
OP true se Ã¨ valida
   false altrimenti
```

```
*/
bool isValid(const Coordinate* m, int righe, int colonne){
    return (m->riga >= 0) && (m->riga < righe) && (m->colonna >= 0) && (m-
>colonna < colonne);
}/*isValid*/



/*Funzione che controlla se la casella a destra Ã¨ stata colpita

IP comp_griglia_avv: griglia su cui controllare le caselle
IP m: mossa da controllare
IP righe: numero di righe della griglia
IP colonne: numero di colonne della griglia
OP true se Ã¨ valida
   false altrimenti

*/
bool possibileColpo_destra(Casella** comp_griglia_avv, Coordinate* m, int righe,
int colonne){
    Coordinate new_m = *m;
    new_m.colonna++;
    if(isValid(&new_m, righe,colonne))
        return comp_griglia_avv[new_m.riga][new_m.colonna].colpito != 1;
    else
        return false;

}/* possibileColpo_destra */

/*Funzione che controlla se la casella a sinistra Ã¨ stata colpita

IP comp_griglia_avv: griglia su cui controllare le caselle
IP m: mossa da controllare
IP righe: numero di righe della griglia
IP colonne: numero di colonne della griglia
OP true se Ã¨ valida
   false altrimenti

*/
bool possibileColpo_sinistra(Casella** comp_griglia_avv, Coordinate* m, int ri-
ghe, int colonne){
    Coordinate new_m = *m;
    new_m.colonna--;
    if(isValid(&new_m, righe,colonne))
        return comp_griglia_avv[new_m.riga][new_m.colonna].colpito != 1;
    else
        return false;
}/* possibileColpo_sinistra */

/*Funzione che controlla se la casella in alto Ã¨ stata colpita

IP comp_griglia_avv: griglia su cui controllare le caselle
IP m: mossa da controllare
IP righe: numero di righe della griglia
IP colonne: numero di colonne della griglia
OP true se Ã¨ valida
   false altrimenti

*/
bool possibileColpo_alto(Casella** comp_griglia_avv, Coordinate* m, int righe,
int colonne){
    Coordinate new_m = *m;
    new_m.riga++;
```

```
    if(isValid(&new_m, righe,colonne))
        return comp_griglia_avv[new_m.riga][new_m.colonna].colpito != 1;
    else
        return false;
}/* possibileColpo_alto */

/*Funzione che controlla se la casella in basso Ã¨ stata colpita

IP comp_griglia_avv: griglia su cui controllare le caselle
IP m: mossa da controllare
IP righe: numero di righe della griglia
IP colonne: numero di colonne della griglia
OP true se Ã¨ valida
   false altrimenti

*/
bool possibileColpo_basso(Casella** comp_griglia_avv, Coordinate* m, int righe,
int colonne){
    Coordinate new_m = *m;
    new_m.riga--;
    if(isValid(&new_m, righe,colonne))
        return comp_griglia_avv[new_m.riga][new_m.colonna].colpito != 1;
    else
        return false;
}/* possibileColpo_basso */


/*Funzione che dice se c'Ã¨ una casella attorno che non Ã¨ ancora stata colpita

IP comp_griglia_avv: griglia su cui controllare le caselle
IP m: mossa da controllare
IP righe: numero di righe della griglia
IP colonne: numero di colonne della griglia
OP true se Ã¨ valida
   false altrimenti

*/
bool possibileColpo(Casella** comp_griglia_avv, Coordinate* possibile_mossa, int
righe, int colonne){
    return possibileColpo_destra(comp_griglia_avv, possibile_mossa, righe, co-
lonne) ||
           possibileColpo_sinistra(comp_griglia_avv, possibile_mossa, righe,
colonne) ||
           possibileColpo_alto(comp_griglia_avv, possibile_mossa, righe, colon-
ne) ||
           possibileColpo_basso(comp_griglia_avv, possibile_mossa, righe, co-
lonne);
}/* possibileColpo */


/*Funuzione che controllase ci sono ancora navi del tipo che stiamo cercando di
affondare

IP naviPartita: array che contiene le informazioni sulle navi della partita
IP nTipi: numro di navi di questa partita
IP caselle_affondate: numero di caselle affoondate della nave che abbiamo trova-
to
IP+OP indiceNaviPartita: puntatore che conterrÃ  la posizione nell'array della
nave del tipo di quella affondata
OP restituisce TRUE se trova ancora una nave di questo tipo, FALSE altrimenti

*/
```

```
bool controlloTipoNave(InfoNavi* naviPartita,short nTipi, short casel-
le_affondate, int* indiceNaviPartita){
    int i;
    for(i = 0; i < nTipi; i++){
        if(naviPartita[i].tipo == caselle_affondate){
            if(naviPartita[i].nNav > 0){
                *indiceNaviPartita = i;
                return true;
            }else{
                return false;
            }
        }
    }
    *indiceNaviPartita = -1;
    return false;
}/* controlloTipoNave */


/*Funzione che sposta una nave nel primo posto disponibile dell'array

IP+OP navi_trovate: array in cui il computer salva le navi trovate
IP indiceNave: indice della nave da spostare
IP count_navi_trovate: numero di navi attualmente affondate
IP navi_attuali: numero di navi attualmente trovate da affondare

*/
void spostaNavi(Nave* navi_trovate, int indiceNave, short count_navi_trovate,
short navi_attuali){
    int i;
    Nave nave_temp = navi_trovate[indiceNave];
    for(i = indiceNave; i < (count_navi_trovate + navi_attuali - 1); i++){
        navi_trovate[i] = navi_trovate[i+1];
    }
    navi_trovate[count_navi_trovate + navi_attuali - 1] = nave_temp;
}/* spostaNavi */

/*-------------------SEZIONE RISULTATI MOSSA-----------------------------*/


/*Funzione che aggiorna i valori di griglia e navi dopo una mossa con acqua

IP+OP comp_griglia_avv: array bidimensionale di caselle che rappresenta la gri-
glia in cui il computer fa le mosse per trovare le navi dell'avv
IP mossa: puntatore alla mossa del computer
IP res_prec_mossa: valore che indica se la mossa precedente era ACQUA o COLPITO
IP naviincrociate: valore booleano che indice se abbiamo trovato due navi tra
loro incrociate

*/
void aggiornamentoAcqua(Casella** comp_griglia_avv, Coordinate* mossa, bool*
res_prec_mossa, bool *navincrociate){
    comp_griglia_avv[mossa->riga][mossa->colonna].colpito = true;
    *navincrociate = false;
    *res_prec_mossa = false;
}/* aggiornamentoAcqua */


/*Funzione che aggiorna i valori di griglia e navi dopo una mossa con colpito

IP+OP navi_trovate: array in cui il computer salva le navi trovate
IP+OP comp_griglia_avv: array bidimensionale di caselle che rappresenta la gri-
glia in cui il computer fa le mosse per trovare le navi dell'avv
```

```
IP+OP griglia: array bidimensionale di caselle che rappresenta la vera griglia
del giocatore che gioca contro il computer
IP naviPartita: array con le informazioni sulle navi della partita
IP nTipi: dimensione dell'array $naviPartita
IP mossa: puntatore alla mossa del computer
IP+OP navi_attuali: puntatore al numero di navi attualmente trovate ancra da af-
fondare
IP+OP count_navi_trovate: puntatore al numero di navi affondate
IP res_prec_mossa: valore che indica se la mossa precedente era ACQUA o COLPITO
IP indiceNave: puntatore alla ave da valutare
IP indicePosizione: posizione di una coordinata all'interno dell'array posizione
di una nave
IP naviincrociate: valore booleano che indice se abbiamo trovato due navi tra
loro incrociate
IP righe: numero di righe della griglia (dimensione uno di $griglia e
$comp_griglia_avv)
IP colonne: numero di colonne della griglia (dimensione due di $griglia e
$comp_griglia_avv)

*/
void aggiornamentoColpito(Nave* navi_trovate, Casella** comp_griglia_avv, Casel-
la** griglia, InfoNavi* naviPartita, int nTipi, Coordinate* mossa, short* na-
vi_attuali, short* count_navi_trovate, bool* res_prec_mossa, int* indiceNave,
int * indicePosizione, bool *navincrociate, int righe, int colonne){

    if(*navincrociate){
        eccezioneNaviIncrociateMultiple(comp_griglia_avv, navi_trovate, naviPar-
tita, mossa, nTipi, count_navi_trovate, navi_attuali, indiceNave, indicePosizio-
ne, righe, colonne, (int)*count_navi_trovate);
        *indiceNave = *count_navi_trovate;
        *navincrociate = false;
    }else if(navi_trovate[*count_navi_trovate].affondata){
        eccezioneNaviDoppie(comp_griglia_avv, navi_trovate, naviPartita, mossa,
nTipi, count_navi_trovate, navi_attuali, indiceNave, indicePosizione, righe, co-
lonne);
        *indiceNave = *count_navi_trovate;
    }else if(!(navi_trovate[*indiceNave].caselleAff)){
        (*navi_attuali)++;
    }

    navi_trovate[*indiceNave].posizione[navi_trovate[*indiceNave].caselleAff] =
*mossa;
    (navi_trovate[*indiceNave].caselleAff)++;
    comp_griglia_avv[mossa->riga][mossa->colonna].contenuto = true;
    comp_griglia_avv[mossa->riga][mossa->colonna].colpito = true;
    griglia[mossa->riga][mossa->colonna].pN->caselleAff++;
    *res_prec_mossa = true;

}/* aggiornamentoColpito */


/*Funzione che aggiorna griglia e navi dopo una mossa con affondato

IP comp_griglia_avv: array bidimensionale di caselle che rappresenta la griglia
in cui il computer fa le mosse per trovare le navi dell'avv
IP+OP navi_trovate: array in cui il computer salva le navi trovate
IP+OP naviPartita: array con le informazioni sulle navi della partita
IP nTipi: dimensione dell'array $naviPartita
IP+OP navi_attuali: puntatore al numero di navi attualmente trovate ancra da af-
fondare
IP+OP count_navi_trovate: puntatore al numero di navi affondate
IP indiceNave: puntatore alla ave da valutare
```

```
*/
bool aggiornamentoAffondato(Casella** comp_griglia_avv,Nave* navi_trovate, Info-
Navi* naviPartita, int nTipi, short* navi_attuali, short* count_navi_trovate,
int* indiceNave){
    int indiceNaviPartita;
    navi_trovate[*indiceNave].lunghezza = navi_trovate[*indiceNave].caselleAff;
    navi_trovate[*indiceNave].affondata = true;

    if(!(navi_trovate[*indiceNave].caselleAff < naviPartita[nTipi-1].tipo)){
        realloc(navi_trovate[*indiceNave].posizione, sizeof(Coordinate) * (na-
vi_trovate[*indiceNave].caselleAff));
        assert(navi_trovate[*indiceNave].posizione != NULL);
    }

    ordinaNave(&navi_trovate[*indiceNave]);

    if(controlloTipoNave(naviPartita, nTipi, na-
vi_trovate[*indiceNave].caselleAff, &indiceNaviPartita)){
        naviPartita[indiceNaviPartita].nNav--;
        if(*indiceNave >= *count_navi_trovate){
            (*count_navi_trovate)++;
            (*navi_attuali)--;
        }
        return true;
    }

    return false;
}/* aggiornamentoAffondato */


/*Funzione che elabora i risultati della mossa del computer

IP comp_griglia_avv: array bidimensionale di caselle che rappresenta la griglia
in cui il computer fa le mosse per trovare le navi dell'avv
IP+OP griglia: array bidimensionale di caselle che rappresenta la vera griglia
del giocatore che gioca contro il computer
IP navi_trovate: array in cui il computer salva le navi trovate
IP naviPartita: array con le informazioni sulle navi della partita
IP nTipi: dimensione dell'array $naviPartita
IP mossa: puntatore alla mossa del computer
IP navi_attuali: puntatore al numero di navi attualmente trovate ancra da affon-
dare
IP count_navi_trovate: puntatore al numero di navi affondate
IP res_prec_mossa: valore che indica se la mossa precedente era ACQUA o COLPITO
IP indiceNave: puntatore alla ave da valutare
IP indicePosizione: posizione di una coordinata all'interno dell'array posizione
di una nave
IP naviincrociate: valore booleano che indice se abbiamo trovato due navi tra
loro incrociate
IP righe: numero di righe della griglia (dimensione uno di $griglia e
$comp_griglia_avv)
IP colonne: numero di colonne della griglia (dimensione due di $griglia e
$comp_griglia_avv)

*/
int risultatoMossa(Casella** comp_griglia_avv, Casella** griglia, Nave* na-
vi_trovate, InfoNavi* naviPartita, int nTipi, Coordinate* mossa, short* na-
vi_attuali, short* count_navi_trovate, bool* res_prec_mossa, int *indiceNave,
int *indicePosizione, bool *navincrociate, int righe, int colonne){
    printf("MOSSA: ");
    stampa_mossa(mossa);
    griglia[mossa->riga][mossa->colonna].colpito = true;
    if(griglia[mossa->riga][mossa->colonna].contenuto){
```

```
        aggiornamentoColpito(navi_trovate, comp_griglia_avv, griglia, naviPartita, nTipi, mossa, navi_attuali, count_navi_trovate, res_prec_mossa, indiceNave, indicePosizione, navincrociate, righe, colonne);
        if(controlloAffondo(griglia[mossa->riga][mossa->colonna].pN)){
            aggiornamentoAffondato(comp_griglia_avv, navi_trovate, naviPartita, nTipi, navi_attuali, count_navi_trovate, indiceNave);
            printf("COLPITO E AFFONDATO\n");
            return 2;
        }
        printf("COLPITO\n");
        return 1;
    }else{
        aggiornamentoAcqua(comp_griglia_avv, mossa, res_prec_mossa, navincrociate);
        printf("ACQUA\n");
        return 0;
    }
}/* risultatoMossa */


/*----------------------SEZIONE MOSSA RANDOM-------------------------------*/

/* Funzione che trova la mossa da fare in maniera casuale

IP comp_griglia_avv: array bidimensionale di caselle che rappresenta la griglia in cui il computer fa le mosse per trovare le navi dell'avv
IP righe: numero di righe della griglia (dimensione uno di $comp_griglia_avv)
IP colonne: numero di colonne della griglia (dimensione due di $comp_griglia_avv)
OP mossa: mossa da eseguire

*/
Coordinate mossa_random(Casella** comp_griglia_avv, int righe, int colonne){
    Coordinate mossa;
    int i=0;
    srand(time(NULL));
    do{
        mossa.riga = rand() % righe;
        mossa.colonna = rand() % colonne;
        i++;
    }
    while(!(isValid(&mossa, righe,colonne) && !comp_griglia_avv[mossa.riga][mossa.colonna].colpito));
    return mossa;
}/*mossaRandom*/


/*-------------------------SEZIONE ESECUZIONE MOSSA-------------------------*/


/* Funzione che trova la mossa da fare e restituisce il risultato

IP+OP comp_griglia_avv: array bidimensionale di caselle che rappresenta la griglia in cui il computer fa le mosse per trovare le navi dell'avv
IP+OP griglia: array bidimensionale di caselle che rappresenta la vera griglia del giocatore che gioca contro il computer
IP+OP navi_trovate: array in cui il computer salva le navi trovate
IP naviPartita: array con le informazioni sulle navi della partita
IP nTipi: dimensione dell'array $naviPartita
IP righe: numero di righe della griglia (dimensione uno di $griglia e $comp_griglia_avv)
IP colonne: numero di colonne della griglia (dimensione due di $griglia e $comp_griglia_avv)
```

```
*/
short eseguiMossa(Casella** comp_griglia_avv, Casella** griglia, Nave* na-
vi_trovate,InfoNavi* naviPartita, int nTipi, int righe, int colonne, int
num_navi){
    static short count_navi_trovate = 0; /*NUMERO DI NAVI AFFONDATE*/
    static short navi_attuali = 0; /*NUMERO DI NAVI ATTUALMENTE TROVATE DA
AFFONDARE*/
    static bool res_prec_mossa; /*RISULTATO DELLA MOSSA PRECEDENTE*/
    static bool navincrociate = false; /*VARIABILE PER CAPIRE SE SONO IN CASO DI
NAVI INCROCIATE TRA DI LORO*/
    int indiceNave; /*INDICE DELLA NAVE DA CONSIDERARE*/
    int indicePosizione = 0; /*INDICE DELLA POSIZIONE DELLA NAVE DA
CONSIDERARE*/
    Coordinate mossa; /*MOSSA DA ESEGUIRE*/

    indiceNave = count_navi_trovate;
    if(!navi_attuali){
        mossa = mossa_random(comp_griglia_avv, righe, colonne);
    }else{
        mossa = mossa_ragionata(comp_griglia_avv, navi_trovate, naviPartita,
res_prec_mossa, nTipi, num_navi, righe, colonne, &navi_attuali,
&count_navi_trovate, &indiceNave, &indicePosizione, &navincrociate);
    }

    return risultatoMossa(comp_griglia_avv, griglia, navi_trovate, naviPartita,
nTipi, &mossa, &navi_attuali, &count_navi_trovate, &res_prec_mossa, &indiceNave,
&indicePosizione, &navincrociate, righe, colonne);
}/*eseguiMossa*/

/*----------------------MOSSA GIOCATORE-----------------------------------*/

/* Funzione che gestisce la mossa del giocatore

IP+OP comp_griglia: array bidimensionale contenente la griglia di navi del coom-
puter
IP+OP my_grigli_avv: array bidimensionale contenente la griglia in cui deve col-
pire il giocatore
IP righe: numero di righe della griglia (dimensione uno di $my_griglia_avv e
$comp_griglia)
IP colonne: numero di colonne della griglia (dimensione due di $my_griglia_avv e
$comp_griglia)
OP ritsultato della mossa

*/
short mossaGiocatore(Casella **comp_griglia, Casella **my_griglia_avv, int ri-
ghe, int colonne){
    char casella_g[3];
    Coordinate mossaGiocatore;
    /* FAI LA MOSSA */
    printf("Dove vuoi colpire? ");
    leggiCasella(casella_g, righe, colonne);
    mossaGiocatore = coordinateCasella(casella_g);
    while(!isValid(&mossaGiocatore, righe, colonne) ||
my_griglia_avv[mossaGiocatore.riga][mossaGiocatore.colonna].colpito){
        printf("Errore, casella gia' colpita, reinserisci\n");
        printf("Dove vuoi colpire? ");
        leggiCasella(casella_g, righe, colonne);
        mossaGiocatore = coordinateCasella(casella_g);
    }
    /* SISTEMAZIONE GRIGLIA E RISULTATO*/
    comp_griglia[mossaGiocatore.riga][mossaGiocatore.colonna].colpito = true;
    my_griglia_avv[mossaGiocatore.riga][mossaGiocatore.colonna].colpito = true;
```

```
    if(comp_griglia[mossaGiocatore.riga][mossaGiocatore.colonna].contenuto){
        my_griglia_avv[mossaGiocatore.riga][mossaGiocatore.colonna].contenuto =
true;
        comp_griglia[mossaGiocatore.riga][mossaGiocatore.colonna].pN-
>caselleAff++;

if(controlloAffondo(comp_griglia[mossaGiocatore.riga][mossaGiocatore.colonna].pN
)){
            printf("COLPITO E AFFONDATO\n");
            return 2;
        }
        printf("COLPITO\n");
        return 1;
    }else{
        printf("ACQUA\n");
        return 0;
    }
}/* mossaGiocatore */
```

Il file MossaRagionata.c contiene le funzioni per gestire i ragionamenti del computer nel fare le mosse

```
#include "Mossa.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <assert.h>

/*----------------------SEZIONE NAVI VICINE-----------------------------*/

/*Funzione che cerca la presenza di navi da ricomporre

IP comp_griglia_avv: array bidimensionale di caselle che rappresenta la griglia
in cui il computer fa le mosse per trovare le navi dell'avv
IP+OP navi_trovate: array in cui il computer salva le navi trovate
IP naviPartita: array con le informazioni sulle navi della partita
IP+OP count_navi_trovate: numero di navi affondate
IP+OP navi_attuali: puntatore alle navi attualmente trovate ancora da affondare
IP direzione: direzione in cui stiamo attualmente colpendo
IP mossa: coordinate di partenza - nuova mossa da fare se possibile
IP righe: dimensione uno di $comp_griglia_avv
IP colonne: dimensione due di $comp_griglia_avv
OP true se ha trovato una nave, false altrimneti

*/
bool ricomponiNave(Casella** comp_griglia_avv, Nave* navi_trovate, InfoNavi* na-
viPartita, short* count_navi_trovate, short* navi_attuali, bool direzione, Coor-
dinate* cas, int max_dimensione, int nTipi){
    int i, j;
    bool direzioneNave,versoNave;
    int indiceNave = -1;
    int diff_riga, diff_colonna;

    for(i = *count_navi_trovate + *navi_attuali - 1; i >= 0; i--){
        trovaDirezioneVersoNave(&navi_trovate[i], &direzioneNave, &versoNave);
        if((direzione && direzioneNave) || (!direzione && !direzioneNave) ||
(navi_trovate[i].caselleAff == 1)){
            for(j = 0; j < navi_trovate[i].caselleAff; j++){
                diff_riga = abs(navi_trovate[i].posizione[j].riga - cas->riga);
```

```
                diff_colonna = abs(navi_trovate[i].posizione[j].colonna - cas-
>colonna);
                if((direzione && (diff_colonna == 1)) || (!direzione &&
(diff_riga == 1))){
                    if(indiceNave == -1){
                        /* prima nave vicina trovata */
                        indiceNave = i;
                        navi_trovate[indiceNave].caselleAff++;
                        if(navi_trovate[indiceNave].affondata){
                            if(indiceNave < *count_navi_trovate){
                                aggiungiNavePartita(naviPartita, na-
vi_trovate[indiceNave].caselleAff-1, nTipi);
                            }
                            realloc(navi_trovate[indiceNave].posizione,
sizeof(Coordinate) * navi_trovate[indiceNave].caselleAff);
                            assert(navi_trovate[indiceNave].posizione != NULL);
                        }
                        na-
vi_trovate[indiceNave].posizione[navi_trovate[indiceNave].caselleAff - 1] =
*cas;
                    }else{
                        /* ho giÃ  trovato una nave vicina a questa casella */
                        if(!navi_trovate[i].affondata && na-
vi_trovate[indiceNave].affondata){
                            navi_trovate[indiceNave].affondata = false;
                            navi_trovate[i].affondata = true;
                            realloc(navi_trovate[indiceNave].posizione,
sizeof(Coordinate) * max_dimensione);
                            assert(navi_trovate[indiceNave].posizione != NULL);
                            navi_trovate[i].caselleAff += na-
vi_trovate[indiceNave].caselleAff;
                            realloc(navi_trovate[i].posizione,
sizeof(Coordinate) * navi_trovate[i].caselleAff);
                            assert(navi_trovate[i].posizione != NULL);
                            while(navi_trovate[indiceNave].caselleAff != 0){
                                na-
vi_trovate[i].posizione[navi_trovate[i].caselleAff - 1] = na-
vi_trovate[indiceNave].posizione[navi_trovate[indiceNave].caselleAff - 1];
                                navi_trovate[indiceNave].caselleAff--;
                            }
                            indiceNave = i;
                            (*navi_attuali)--;
                        }else if(!navi_trovate[indiceNave].affondata){
                            navi_trovate[i].caselleAff += na-
vi_trovate[indiceNave].caselleAff;
                            if(navi_trovate[i].affondata){
                                realloc(navi_trovate[i].posizione,
sizeof(Coordinate) * navi_trovate[i].caselleAff);
                                assert(navi_trovate[i].posizione != NULL);
                            }
                            while(navi_trovate[indiceNave].caselleAff != 0){
                                na-
vi_trovate[i].posizione[navi_trovate[i].caselleAff - 1] = na-
vi_trovate[indiceNave].posizione[navi_trovate[indiceNave].caselleAff - 1];
                                navi_trovate[indiceNave].caselleAff--;
                            }
                            indiceNave = i;
                            (*navi_attuali)--;
                        }
                    }
                }
            }
        }
```

```
    }
    if(indiceNave == -1)
        return false;
    else{
        if(indiceNave < *count_navi_trovate){
            scambiaCasel-
le(&navi_trovate[indiceNave].posizione[navi_trovate[indiceNave].caselleAff - 2],
&navi_trovate[indiceNave].posizione[navi_trovate[indiceNave].caselleAff - 1]);
            if(!aggiornamentoAffondato(comp_griglia_avv, navi_trovate, naviPar-
tita, nTipi, count_navi_trovate, navi_attuali, &indiceNave)){
                spostaNavi(navi_trovate, indiceNave, *count_navi_trovate,
*navi_attuali);
                (*count_navi_trovate)--;
                (*navi_attuali)++;
            }
        }
        return true;
    }
}/* ricomponiNave */


/*Funzione che gestisce il caso di navi affiancate

IP comp_griglia_avv: array bidimensionale di caselle che rappresenta la griglia
in cui il computer fa le mosse per trovare le navi dell'avv
IP+OP navi_trovate: array in cui il computer salva le navi trovate
IP naviPartita: array con le informazioni sulle navi della partita
IP nTipi: numero di tipi di navi nella partita
IP count_navi_trovate: puntatore al numero di navi affondate
IP+OP navi_attuali: puntatore al numero di navi attualmente trovate ancra da af-
fondare
IP righe: numero di righe della griglia (dimensione uno di $comp_griglia_avv)
IP colonne: numero di colonne della griglia (dimensione due di
$comp_griglia_avv)

*/
void eccezione_navi_affiancate(Casella** comp_griglia_avv,Nave* navi_trovate,
InfoNavi* naviPartita, int nTipi, short* count_navi_trovate, short* na-
vi_attuali, int max_dimensione){
    int i;
    bool direzione,verso;
    int caselle_affondate = navi_trovate[*count_navi_trovate].caselleAff;
    int caselle_spostate = 0;

    /* cerca navi vicine per ogni casella */
    trovaDirezioneVersoNave(&navi_trovate[*count_navi_trovate], &direzione,
&verso);
    for(i = 0; i < caselle_affondate; i++){
        if(ricomponiNave(comp_griglia_avv, navi_trovate, naviPartita,
count_navi_trovate, navi_attuali, !direzione,
&navi_trovate[*count_navi_trovate].posizione[i], max_dimensione, nTipi)){
            navi_trovate[*count_navi_trovate].caselleAff--;
        }else{
            navi_trovate[*count_navi_trovate].posizione[caselle_spostate] = na-
vi_trovate[*count_navi_trovate].posizione[i];
            caselle_spostate++;
        }
    }

    /* se tutte le caselle avevano navi vicine, elimina questa nave da quelle
attuali */
    if(navi_trovate[*count_navi_trovate].caselleAff == 0){
```

```
        spostaNavi(navi_trovate, *count_navi_trovate, *count_navi_trovate,
*navi_attuali);
        (*navi_attuali)--;
        return;
    }

    /* se sono rimaste caselle mettile in nuove navi da colpire */
    for(i = 1; i < navi_trovate[*count_navi_trovate].caselleAff; i++){
        navi_trovate[*count_navi_trovate + *navi_attuali + i - 1].posizione[0] =
na-
vi_trovate[*count_navi_trovate].posizione[navi_trovate[*count_navi_trovate].case
lleAff - i];
        navi_trovate[*count_navi_trovate + *navi_attuali + i - 1].caselleAff++;
    }
    (*navi_attuali) += navi_trovate[*count_navi_trovate].caselleAff - 1;
    navi_trovate[*count_navi_trovate].caselleAff = 1;

}/* eccezione_navi_affiancate */


/*Funzione che gestisce il caso di navi doppie

IP comp_griglia_avv: array bidimensionale di caselle che rappresenta la griglia
in cui il computer fa le mosse per trovare le navi dell'avv
IP+OP navi_trovate: array in cui il computer salva le navi trovate
IP naviPartita: array con le informazioni sulle navi della partita
IP mossa: puntatore alla mossa effettuata dal computer
IP nTipi: dimensione dell'array $naviPartita
IP+OP count_navi_trovate: puntatore al numero di navi affondate
IP+OP navi_attuali: puntatore al numero di navi attualmente trovate ancra da af-
fondare
IP indiceNave: puntatore alla ave da valutare
IP indicePosizione: posizione di una coordinata all'interno dell'array posizione
di una nave
IP righe: numero di righe della griglia (dimensione uno di $griglia e
$comp_griglia_avv)
IP colonne: numero di colonne della griglia (dimensione due di $griglia e
$comp_griglia_avv)

*/
void eccezioneNaviDoppie(Casella** comp_griglia_avv, Nave* navi_trovate, InfoNa-
vi* naviPartita, Coordinate* mossa, int nTipi, short* count_navi_trovate, short*
navi_attuali, int* indiceNave, int* indicePosizione, int righe, int colonne){
    int i;
    bool direzione, direzioneNave, verso, versoNave;

    (*navi_attuali)++;

    trovaDirezioneVerso(&navi_trovate[*indiceNave].posizione[0], mossa,
&direzione, &verso);
    trovaDirezioneVersoNave(&navi_trovate[*indiceNave], &direzioneNave,
&versoNave);

    /* se la casella Ã¨ nella stessa direzione gestisci prima questa */
    if((direzione && direzioneNave) || (!direzione && !direzioneNave)){
        if(*navi_attuali > 2){
            spostaNavi(navi_trovate, *count_navi_trovate + 1,
*count_navi_trovate, *navi_attuali);
        }
        spostaNavi(navi_trovate, *indiceNave, *count_navi_trovate,
*navi_attuali);
        return;
    }
```

```
    /* se non Ã¨ nella stessa direzione allora aggiorna le navi */
    navi_trovate[*count_navi_trovate + *navi_attuali - 1].posizione[0] = na-
vi_trovate[*indiceNave].posizione[0];
    navi_trovate[*count_navi_trovate + *navi_attuali - 1].caselleAff++;
    (navi_trovate[*indiceNave].caselleAff)--;
    for(i = 1; i <= navi_trovate[*indiceNave].caselleAff; i++){
        navi_trovate[*indiceNave].posizione[i-1] = na-
vi_trovate[*indiceNave].posizione[i];
    }

    if(*navi_attuali > 2){
        spostaNavi(navi_trovate, *count_navi_trovate + 1, *count_navi_trovate,
*navi_attuali);
    }

    if(*indiceNave < *count_navi_trovate){
        (*count_navi_trovate)++;
        (*navi_attuali)--;
    }
    if(!aggiornamentoAffondato(comp_griglia_avv, navi_trovate, naviPartita, nTi-
pi, navi_attuali, count_navi_trovate, indiceNave)){
        if(*indiceNave < *count_navi_trovate){
            (*count_navi_trovate)--;
            (*navi_attuali)++;
        }
        spostaNavi(navi_trovate, *indiceNave, *count_navi_trovate,
*navi_attuali);
    }

}/* eccezioneNaviDoppie */


/*Funzione che gestisce il caso di navi incrociate

IP comp_griglia_avv: array bidimensionale di caselle che rappresenta la griglia
in cui il computer fa le mosse per trovare le navi dell'avv
IP navi_trovate: array in cui il computer salva le navi trovate
IP naviPartita: array con le informazioni sulle navi della partita
IP mossa: puntatore alla mossa effettuata dal computer
IP nTipi: dimensione dell'array $naviPartita
IP count_navi_trovate: puntatore al numero di navi affondate
IP navi_attuali: puntatore al numero di navi attualmente trovate ancra da affon-
dare
IP+OP indiceNave: puntatore alla ave da valutare
IP indicePosizione: posizione di una coordinata all'interno dell'array posizione
di una nave
IP righe: numero di righe della griglia (dimensione uno di $griglia e
$comp_griglia_avv)
IP colonne: numero di colonne della griglia (dimensione due di $griglia e
$comp_griglia_avv)
IP indiceNaveOriginale: valore della nave che stavamo colpendo andando a trovare
una nave incrociata

*/
void eccezioneNaviIncrociateMultiple(Casella** comp_griglia_avv, Nave* na-
vi_trovate, InfoNavi* naviPartita, Coordinate* mossa, int nTipi, short*
count_navi_trovate, short* navi_attuali, int* indiceNave, int* indicePosizione,
int righe, int colonne, int indiceNaveOrig){
    int i,diff, diff_min = 0;
    bool direzione,verso;
    Coordinate next_mossa = *mossa;
```

```
    for(i = 0; i < navi_trovate[*count_navi_trovate].caselleAff; i++){
        diff = trovaDirezioneVerso(mossa,
&navi_trovate[*count_navi_trovate].posizione[i], &direzione, &verso);
        diff = abs(diff);
        if(diff < diff_min)
            diff_min = diff;
    }
    diff--;

    for(i = 0; i < diff; i++){
        aumentaMossa(&next_mossa, direzione, verso, 1);
        *indiceNave = cercaNavePosizione(navi_trovate, *count_navi_trovate,
*navi_attuali, (int *)&count_navi_trovate, indicePosizione, &next_mossa);
        eccezioneNaviIncrociate(comp_griglia_avv, navi_trovate, naviPartita,
nTipi, count_navi_trovate, navi_attuali, indiceNave, indicePosizione, righe, co-
lonne, indiceNaveOrig);
    }
}/* eccezioneNaviIncrociateMultiple */


/*Funzione che separa una nave

IP comp_griglia_avv: array bidimensionale di caselle che rappresenta la griglia
in cui il computer fa le mosse per trovare le navi dell'avv
IP+OP navi_trovate: array in cui il computer salva le navi trovate
IP naviPartita: array con le informazioni sulle navi della partita
IP nTipi: dimensione dell'array $naviPartita
IP+OP count_navi_trovate: puntatore al numero di navi affondate
IP+OP navi_attuali: puntatore al numero di navi attualmente trovate ancra da af-
fondare
IP indiceNave: puntatore alla ave da valutare
IP+OP indicePosizione: posizione di una coordinata all'interno dell'array posi-
zione di una nave
IP righe: numero di righe della griglia (dimensione uno di $griglia e
$comp_griglia_avv)
IP colonne: numero di colonne della griglia (dimensione due di $griglia e
$comp_griglia_avv)
IP indiceNaveOriginale: puntatore alla nave che stavamo colpendo originariamente

*/
void separaNave(Casella** comp_griglia_avv, Nave* navi_trovate, InfoNavi* navi-
Partita, int nTipi, short* count_navi_trovate, short* navi_attuali, int indice-
Nave, int* indicePosizione, int righe, int colonne, int indiceNaveOrig){
    int i;
    Coordinate new_mossa;
    int indicePosizioneOrig;
    bool direzione, verso, direzioneNave, versoNave;

    /* carca nave dalla parte opposta dell'affondato */
    indicePosizioneOrig = *indicePosizione;
    new_mossa  = navi_trovate[indiceNaveOrig].posizione[0];
    trovaDirezioneVersoNave(&navi_trovate[indiceNaveOrig], &direzione, &verso);
    verso = !verso;
    aumentaMossa(&new_mossa, direzione, verso, 1);
    if(isValid(&new_mossa, righe,colonne) &&
comp_griglia_avv[new_mossa.riga][new_mossa.colonna].contenuto){
        indiceNave = cercaNavePosizione(navi_trovate, *count_navi_trovate,
*navi_attuali, &indiceNave, indicePosizione, &new_mossa);
        trovaDirezioneVersoNave(&navi_trovate[indiceNave], &direzioneNave,
&versoNave);
        aggiungiNavePartita(naviPartita, navi_trovate[indiceNave].caselleAff,
nTipi);
        if((direzione && direzioneNave) || (!(direzione) && !direzioneNave)){
```

```
            navi_trovate[indiceNave].caselleAff += indicePosizioneOrig;
            realloc(navi_trovate[indiceNave].posizione, sizeof(Coordinate) *
(navi_trovate[indiceNave].caselleAff));
            assert(navi_trovate[indiceNave].posizione != NULL);
            for(i = 0; i < indicePosizioneOrig; i++){
                na-
vi_trovate[indiceNave].posizione[navi_trovate[indiceNave].caselleAff - i - 1] =
navi_trovate[indiceNaveOrig].posizione[i];
            }
            scambiaCasel-
le(&navi_trovate[indiceNave].posizione[navi_trovate[indiceNave].caselleAff - 1],
&navi_trovate[indiceNave].posizione[navi_trovate[indiceNave].caselleAff - indi-
cePosizioneOrig - 1] );
            if(!aggiornamentoAffondato(comp_griglia_avv, navi_trovate, naviPar-
tita, nTipi, navi_attuali, count_navi_trovate, &indiceNave)){
                spostaNavi(navi_trovate, indiceNave, *count_navi_trovate,
*navi_attuali);
                (*navi_attuali)++;
                if(indiceNave < *count_navi_trovate)
                    (*count_navi_trovate)--;
            }
        }

        *indicePosizione = indicePosizioneOrig;
        return;
    }

    /* cerca nave dalla parte dell'affondato */
    indicePosizioneOrig = *indicePosizione;
    new_mossa  = na-
vi_trovate[indiceNaveOrig].posizione[navi_trovate[indiceNaveOrig].caselleAff -
1];
    trovaDirezioneVersoNave(&navi_trovate[indiceNaveOrig], &direzione, &verso);
    aumentaMossa(&new_mossa, direzione, verso, 1);
    if(isValid(&new_mossa, righe,colonne) &&
comp_griglia_avv[new_mossa.riga][new_mossa.colonna].contenuto){
        indiceNave = cercaNavePosizione(navi_trovate, *count_navi_trovate,
*navi_attuali, &indiceNave, indicePosizione, &new_mossa);
        trovaDirezioneVersoNave(&navi_trovate[indiceNave], &direzioneNave,
&versoNave);
        aggiungiNavePartita(naviPartita, navi_trovate[indiceNave].caselleAff,
nTipi);
        if((direzione && direzioneNave) || (!(direzione) && !direzioneNave)){
            scambiaCaselle(&navi_trovate[indiceNaveOrig].posizione[0],
&navi_trovate[indiceNaveOrig].posizione[navi_trovate[indiceNaveOrig].caselleAff
- 1]);
            ordinaNave(&navi_trovate[indiceNaveOrig]);
            navi_trovate[indiceNave].caselleAff += (na-
vi_trovate[indiceNaveOrig].caselleAff - indicePosizioneOrig - 1);
            realloc(navi_trovate[indiceNave].posizione, sizeof(Coordinate) *
(navi_trovate[indiceNave].caselleAff));
            assert(navi_trovate[indiceNave].posizione != NULL);
            for(i = 0; i < (navi_trovate[indiceNaveOrig].caselleAff - indicePo-
sizioneOrig - 1); i++){
                na-
vi_trovate[indiceNave].posizione[navi_trovate[indiceNave].caselleAff - i - 1] =
navi_trovate[indiceNaveOrig].posizione[i];
            }
            scambiaCasel-
le(&navi_trovate[indiceNave].posizione[navi_trovate[indiceNave].caselleAff - 1],
&navi_trovate[indiceNave].posizione[navi_trovate[indiceNave].caselleAff - indi-
cePosizioneOrig - 1] );
```

```
            if(!aggiornamentoAffondato(comp_griglia_avv, navi_trovate, naviPar-
tita, nTipi, navi_attuali, count_navi_trovate, &indiceNave)){
                spostaNavi(navi_trovate, indiceNave, *count_navi_trovate,
*navi_attuali);
                (*navi_attuali)++;
                if(indiceNave < *count_navi_trovate)
                    (*count_navi_trovate)--;
            }
        }
        *indicePosizione = (navi_trovate[indiceNaveOrig].caselleAff - indicePo-
sizioneOrig - 1);
        return;
    }


    /* caselle rimaste */
    direzione = !direzione;
    for(i = 0; i < indicePosizioneOrig; i++){
        navi_trovate[*count_navi_trovate + *navi_attuali].posizione[i] = na-
vi_trovate[indiceNaveOrig].posizione[i];
        navi_trovate[*count_navi_trovate + *navi_attuali].caselleAff++;
    }
    (*navi_attuali)++;
    *indicePosizione = indicePosizioneOrig;

}/* separaNave */


/*Funzione che gestisce il caso di navi incrociate

IP comp_griglia_avv: array bidimensionale di caselle che rappresenta la griglia
in cui il computer fa le mosse per trovare le navi dell'avv
IP+OP navi_trovate: array in cui il computer salva le navi trovate
IP naviPartita: array con le informazioni sulle navi della partita
IP mossa: puntatore alla mossa effettuata dal computer
IP nTipi: dimensione dell'array $naviPartita
IP+OP count_navi_trovate: puntatore al numero di navi affondate
IP+OP navi_attuali: puntatore al numero di navi attualmente trovate ancra da af-
fondare
IP indiceNave: puntatore alla ave da valutare
IP indicePosizione: posizione di una coordinata all'interno dell'array posizione
di una nave
IP righe: numero di righe della griglia (dimensione uno di $griglia e
$comp_griglia_avv)
IP colonne: numero di colonne della griglia (dimensione due di $griglia e
$comp_griglia_avv)
IP indiceNaveOriginale: puntatore alla nave che stvamo originariamente colpendo

*/
bool eccezioneNaviIncrociate(Casella** comp_griglia_avv, Nave* na-
vi_trovate,InfoNavi* naviPartita, int nTipi, short* count_navi_trovate, short*
navi_attuali, int* indiceNave, int* indicePosizione, int righe, int colonne, int
indiceNaveOrig){
    int i;
    bool girata = false;

    aggiungiNavePartita(naviPartita, navi_trovate[*indiceNave].caselleAff, nTi-
pi);

    /* se Ã¨ l'ultima casella giraimo la nave */
    if(*indicePosizione == (navi_trovate[*indiceNave].caselleAff - 1)){
        scambiaCaselle(&navi_trovate[*indiceNave].posizione[0],
&navi_trovate[*indiceNave].posizione[*indicePosizione]);
```

```
        ordinaNave(&navi_trovate[*indiceNave]);
        *indicePosizione = 0;
        girata = true;
    }
    /* se Ã¨ una casella centrale dobbiamo separare la nave */
    if(*indicePosizione > 0){
        separaNave(comp_griglia_avv, navi_trovate, naviPartita, nTipi,
count_navi_trovate, navi_attuali, *indiceNave, indicePosizione, righe, colonne,
*indiceNave);
    }

    navi_trovate[*indiceNave].caselleAff -= (*indicePosizione + 1);
    navi_trovate[*indiceNave].lunghezza -= (*indicePosizione + 1);
    (navi_trovate[indiceNaveOrig].caselleAff)++;
    if(navi_trovate[indiceNaveOrig].affondata){
        realloc(navi_trovate[indiceNaveOrig].posizione, sizeof(Coordinate) *
(navi_trovate[indiceNaveOrig].caselleAff));
        assert(navi_trovate[indiceNaveOrig].posizione != NULL);
        na-
vi_trovate[indiceNaveOrig].posizione[navi_trovate[indiceNaveOrig].caselleAff -
1] = na-
vi_trovate[indiceNaveOrig].posizione[navi_trovate[indiceNaveOrig].caselleAff -
2];
        na-
vi_trovate[indiceNaveOrig].posizione[navi_trovate[indiceNaveOrig].caselleAff -
2] = navi_trovate[*indiceNave].posizione[*indicePosizione];
    }else{
        na-
vi_trovate[indiceNaveOrig].posizione[navi_trovate[indiceNaveOrig].caselleAff -
1] = navi_trovate[*indiceNave].posizione[*indicePosizione];
    }

    for(i = 0; i < navi_trovate[*indiceNave].caselleAff; i++){
        navi_trovate[*indiceNave].posizione[i] = na-
vi_trovate[*indiceNave].posizione[*indicePosizione + i + 1];
    }

    /* se abbiamo girato la nave all'inizio la rigiriamo */
    if(girata){
        scambiaCaselle(&navi_trovate[*indiceNave].posizione[0],
&navi_trovate[*indiceNave].posizione[navi_trovate[*indiceNave].caselleAff - 1]);
        ordinaNave(&navi_trovate[*indiceNave]);
    }

    if(!aggiornamentoAffondato(comp_griglia_avv, navi_trovate, naviPartita, nTi-
pi, navi_attuali, count_navi_trovate, indiceNave)){
        spostaNavi(navi_trovate, *indiceNave, *count_navi_trovate,
*navi_attuali);
        if(*indiceNave < *count_navi_trovate){
            (*count_navi_trovate)--;
            (*navi_attuali)++;
        }
        return false;
    }

    return true;
}/* eccezioneNaviIncrociate */


/*Funzione che gestisce il caso di navi incrociate ma in direzione opposta

IP comp_griglia_avv: array bidimensionale di caselle che rappresenta la griglia
in cui il computer fa le mosse per trovare le navi dell'avv
```

```
IP+OP navi_trovate: array in cui il computer salva le navi trovate
IP naviPartita: array con le informazioni sulle navi della partita
IP mossa: puntatore alla mossa effettuata dal computer
IP nTipi: dimensione dell'array $naviPartita
IP+OP count_navi_trovate: puntatore al numero di navi affondate
IP+OP navi_attuali: puntatore al numero di navi attualmente trovate ancra da af-
fondare
IP indiceNave: puntatore alla ave da valutare
IP indicePosizione: posizione di una coordinata all'interno dell'array posizione
di una nave
IP righe: numero di righe della griglia (dimensione uno di $griglia e
$comp_griglia_avv)
IP colonne: numero di colonne della griglia (dimensione due di $griglia e
$comp_griglia_avv)
IP indiceNaveOriginale: puntatore alla nave che stvamo originariamente colpendo

*/
bool eccezioneNaviIncrociateReverse(Casella** comp_griglia_avv, Nave* na-
vi_trovate,InfoNavi* naviPartita, int nTipi, short* count_navi_trovate, short*
navi_attuali, int* indiceNave, int* indicePosizione, int righe, int colonne,
int* indiceNaveOrig){
    int i;
    bool girata = false;

    aggiungiNavePartita(naviPartita, navi_trovate[*indiceNave].caselleAff, nTi-
pi);

    /* se Ã¨ l'ultima casella giraimo la nave */
    if(*indicePosizione == navi_trovate[*indiceNave].caselleAff-1){
        scambiaCaselle(&navi_trovate[*indiceNave].posizione[0],
&navi_trovate[*indiceNave].posizione[*indicePosizione]);
        ordinaNave(&navi_trovate[*indiceNave]);
        *indicePosizione = 0;
        girata = true;
    }
    navi_trovate[*indiceNave].caselleAff -= (*indicePosizione);
    (navi_trovate[*indiceNave].caselleAff)++;
    (navi_trovate[*indiceNaveOrig].caselleAff)--;
    realloc(navi_trovate[*indiceNave].posizione, sizeof(Coordinate) * (na-
vi_trovate[*indiceNave].caselleAff));
    assert(navi_trovate[*indiceNave].posizione != NULL);
    for(i = navi_trovate[*indiceNave].caselleAff - 1; i >= 0; i--){
        navi_trovate[*indiceNave].posizione[i + 1] = na-
vi_trovate[*indiceNave].posizione[*indicePosizione + i];
    }
    navi_trovate[*indiceNave].posizione[0] = na-
vi_trovate[*indiceNaveOrig].posizione[0];

    for(i = 0; i < navi_trovate[*indiceNaveOrig].caselleAff; i++){
        navi_trovate[*indiceNaveOrig].posizione[i] = na-
vi_trovate[*indiceNaveOrig].posizione[i + 1];
    }

    /* se abbiamo girato la nave all'inizio la rigiriamo */
    if(girata){
        scambiaCaselle(&navi_trovate[*indiceNave].posizione[0],
&navi_trovate[*indiceNave].posizione[navi_trovate[*indiceNave].caselleAff - 1]);
        ordinaNave(&navi_trovate[*indiceNave]);
    }

    if(!aggiornamentoAffondato(comp_griglia_avv, navi_trovate, naviPartita, nTi-
pi, navi_attuali, count_navi_trovate, indiceNave)){
```

```
        spostaNavi(navi_trovate, *indiceNave, *count_navi_trovate,
*navi_attuali);
        if(*indiceNave < *count_navi_trovate){
            (*count_navi_trovate)--;
            (*navi_attuali)++;
        }
        return false;
    }

    return true;
}/* eccezioneNaviIncrociateReverse */


/*Funzione che gestisce il caso di navi incrociate ma in direzione opposta

IP comp_griglia_avv: array bidimensionale di caselle che rappresenta la griglia
in cui il computer fa le mosse per trovare le navi dell'avv
IP+OP navi_trovate: array in cui il computer salva le navi trovate
IP naviPartita: array con le informazioni sulle navi della partita
IP mossa: puntatore alla mossa effettuata dal computer
IP nTipi: dimensione dell'array $naviPartita
IP+OP count_navi_trovate: puntatore al numero di navi affondate
IP+OP navi_attuali: puntatore al numero di navi attualmente trovate ancra da af-
fondare
IP+OP indiceNave: puntatore alla ave da valutare
IP indicePosizione: posizione di una coordinata all'interno dell'array posizione
di una nave
IP righe: numero di righe della griglia (dimensione uno di $griglia e
$comp_griglia_avv)
IP colonne: numero di colonne della griglia (dimensione due di $griglia e
$comp_griglia_avv)
IP indiceNaveOriginale: puntatore alla nave che stvamo originariamente colpendo

*/
bool eccezioneNaviInFilaDef(Casella** comp_griglia_avv, Nave* na-
vi_trovate,InfoNavi* naviPartita, int nTipi, short* count_navi_trovate, short*
navi_attuali, int* indiceNave, int* indicePosizione, int righe, int colonne,
int* indiceNaveOrig){
    int i, caselle_tot, cas1, cas2;
    int indiceNaviPartita;
    int indice_temp;
    Coordinate cas_temp;
    bool direzione,verso;

    /* cerco possibili combinazioni di navi */
    cas1 = navi_trovate[*indiceNaveOrig].caselleAff;
    cas2 = navi_trovate[*indiceNave].caselleAff;
    caselle_tot = cas1 + cas2;
    cas1 = caselle_tot/2 + caselle_tot%2;
    cas2 = caselle_tot/2;
    while(cas2 != 0){
        if(controlloTipoNave(naviPartita, nTipi, cas1, &indiceNaviPartita)){
            indice_temp = indiceNaviPartita;
            naviPartita[indice_temp].nNav--;
            if(controlloTipoNave(naviPartita, nTipi, cas2, &indiceNaviPartita)){
                naviPartita[indice_temp].nNav++;
                break;
            }
            naviPartita[indice_temp].nNav++;
        }
        cas1++;
        cas2--;
    }
```

```
    /* se non trovo una combinazione possibile ritorno false */
    if(cas2 == 0)
        return false;

    /* trovo la casella piÃ¹ piccola */
    trovaDirezioneVersoNave(&navi_trovate[*indiceNaveOrig], &direzione, &verso);
    if(verso){
        cas_temp = navi_trovate[*indiceNaveOrig].posizione[0];
    }else{
        cas_temp = na-
vi_trovate[*indiceNaveOrig].posizione[navi_trovate[*indiceNaveOrig].caselleAff -
1];
    }

    trovaDirezioneVersoNave(&navi_trovate[*indiceNave], &direzione, &verso);
    if(verso){
        if(direzione){
            if(cas_temp.colonna > na-
vi_trovate[*indiceNave].posizione[0].colonna)
                cas_temp = navi_trovate[*indiceNave].posizione[0];
        }else{
            if(cas_temp.riga > navi_trovate[*indiceNave].posizione[0].riga)
                cas_temp = navi_trovate[*indiceNave].posizione[0];
        }
    }else{
        if(direzione){
            if(cas_temp.colonna > na-
vi_trovate[*indiceNave].posizione[navi_trovate[*indiceNave].caselleAff -
1].colonna)
                cas_temp = na-
vi_trovate[*indiceNave].posizione[navi_trovate[*indiceNave].caselleAff - 1];
        }else{
            if(cas_temp.riga > na-
vi_trovate[*indiceNave].posizione[navi_trovate[*indiceNave].caselleAff -
1].riga)
                cas_temp = na-
vi_trovate[*indiceNave].posizione[navi_trovate[*indiceNave].caselleAff - 1];
        }
    }

    /* creo le nuove navi */
    for(i = 0; i < cas1; i++){
        realloc(navi_trovate[*indiceNaveOrig].posizione, sizeof(Coordinate) *
cas1);
        assert(navi_trovate[*indiceNaveOrig].posizione != NULL);
        navi_trovate[*indiceNaveOrig].caselleAff = cas1;
        navi_trovate[*indiceNaveOrig].posizione[i] = cas_temp;
        if(direzione){
            cas_temp.colonna++;
        }else{
            cas_temp.riga++;
        }
    }
    for(i = cas1; i < caselle_tot; i++){
        realloc(navi_trovate[*indiceNave].posizione, sizeof(Coordinate) * cas2);
        assert(navi_trovate[*indiceNave].posizione != NULL);
        navi_trovate[*indiceNave].posizione[i - cas1] = cas_temp;
        navi_trovate[*indiceNave].caselleAff = cas2;
        if(direzione){
            cas_temp.colonna++;
        }else{
            cas_temp.riga++;
```

```
        }
    }

    if(*indiceNave < *indiceNaveOrig){
        if(!aggiornamentoAffondato(comp_griglia_avv, navi_trovate, naviPartita,
nTipi, navi_attuali, count_navi_trovate, indiceNave)){
            spostaNavi(navi_trovate, *indiceNave, *count_navi_trovate,
*navi_attuali);
            if(*indiceNave < *count_navi_trovate){
                (*count_navi_trovate)--;
                (*navi_attuali)++;
            }
            *indiceNave = *indiceNaveOrig - 1;
        }else
            *indiceNave = *indiceNaveOrig;
    }else{
        if(!aggiornamentoAffondato(comp_griglia_avv, navi_trovate, naviPartita,
nTipi, navi_attuali, count_navi_trovate, indiceNaveOrig)){
            spostaNavi(navi_trovate, *indiceNaveOrig, *count_navi_trovate,
*navi_attuali);
            if(*indiceNave < *count_navi_trovate){
                (*count_navi_trovate)--;
                (*navi_attuali)++;
            }

            (*indiceNave)--;
        }
    }


    return true;
}/* eccezioneNaviInFilaDef */


/*Funzione che cerca una nave vicina tra quelle trovate e restituisce indicePo-
sizione e indiceNave

IP navi_trovate: array in cui il computer salva le navi trovate
IP casella: puntatore alla casella da cui cercare un'altra nave
IP count_navi_trovate: numero di navi affondate
IP navi_attuali: navi attualmente trovate ancra da affondare
IP indiceNave: puntatore alla ave da valutare
IP+OP indicePosizione: posizione di una coordinata all'interno dell'array posi-
zione di una nave
OP indice della nave vicina

*/
int cercaNaveVicina(Nave *navi_trovate, Coordinate* casella, short
count_navi_trovate, short navi_attuali,int *indiceNave, int* indicePosizione){
    int i, j;

    for(i = count_navi_trovate+navi_attuali-1; i >=0 ; i--){
        for(j = 0; j < navi_trovate[i].caselleAff; j++){
            if((navi_trovate[i].posizione[j].colonna - casella->colonna == 0) &&
(abs(navi_trovate[i].posizione[j].riga - casella->riga) == 1)){
                if(*indiceNave > i){
                    *indicePosizione = j;
                    return i;
                }
            }
            if((navi_trovate[i].posizione[j].riga - casella->riga == 0) &&
(abs(navi_trovate[i].posizione[j].colonna - casella->colonna) == 1)){
                if(*indiceNave > i){
```

```
                    *indicePosizione = j;
                    return i;
                }
            }
        }
    }
    return -1;
}/* cercaNaveVicina */


/*---------------------SEZIONE MOSSA RAGIONATA----------------------------*/

/*Funzione che restituisce la mossa da fare nel caso di possibili navi incrocia-
te

IP comp_griglia_avv: array bidimensionale di caselle che rappresenta la griglia
in cui il computer fa le mosse per trovare le navi dell'avv
IP navi_trovate: array in cui il computer salva le navi trovate
IP righe: numero di righe della griglia (dimensione uno di $comp_griglia_avv)
IP colonne: numero di colonne della griglia (dimensione due di
$comp_griglia_avv)
IP count_navi_trovate: numero di navi affondate
IP navi_attuali: navi attualmente trovate ancra da affondare
IP+OP mossa: coordinate di partenza - nuova mossa da fare se possibile
IP direzione: direzione in cui stiamo attualmente colpendo
IP verso: verso in cui ci stiamo attualmente muovendo
IP+OP indiceNave: puntatore alla ave da valutare
IP indicePosizione: posizione di una coordinata all'interno dell'array posizione
di una nave
IP+OP naviincrociate: valore booleano che indice se abbiamo trovato due navi tra
loro incrociate
OP true se la mossa Ã¨ possibile, false altrimneti

*/
bool mossaNaviIncrociate(Casella** comp_griglia_avv, Nave* navi_trovate, int ri-
ghe, int colonne, short count_navi_trovate, short navi_attuali, Coordinate* mos-
sa, bool direzione, bool verso,  int* indiceNave, int *indicePosizione, bool
*navincrociate){
    Coordinate next_mossa;
    bool direzioneNave, versoNave;
    int indiceNave_temp = *indiceNave;

    next_mossa = *mossa;
    do{
        direzioneNave = direzione;
        aumentaMossa(&next_mossa, direzione, verso, 1);
        if(isValid(&next_mossa, righe,colonne) &&
comp_griglia_avv[next_mossa.riga][next_mossa.colonna].contenuto){
            indiceNave_temp = cercaNavePosizione(navi_trovate,
count_navi_trovate, navi_attuali, &indiceNave_temp, indicePosizione,
&next_mossa);
            trovaDirezioneVersoNave(&navi_trovate[indiceNave_temp],
&direzioneNave, &versoNave);
        }
    }while(isValid(&next_mossa, righe,colonne) && !((direzione && direzioneNave)
|| (!direzione && !direzioneNave)));
    if(isValid(&next_mossa, righe,colonne) &&
!comp_griglia_avv[next_mossa.riga][next_mossa.colonna].colpito){
        /*Salva indiceNave e indicePosizione*/
        *indiceNave = indiceNave_temp;
        *indiceNave = cercaNavePosizione(navi_trovate, count_navi_trovate, na-
vi_attuali, indiceNave, indicePosizione, mossa);
        mossa->colonna = next_mossa.colonna;
```

```
        mossa->riga = next_mossa.riga;
        *navincrociate = true;
        return true;
    }else{
        return false;
    }
}


/*Funzione che restituisce la mossa da fare nel caso di una casella giÃ  colpita

IP comp_griglia_avv: array bidimensionale di caselle che rappresenta la griglia
in cui il computer fa le mosse per trovare le navi dell'avv
IP+OP navi_trovate: array in cui il computer salva le navi trovate
IP naviPartita: array con le informazioni sulle navi della partita
IP nTipi: dimensione dell'array $naviPartita
IP+OP count_navi_trovate: puntatore al numero di navi affondate
IP+OP navi_attuali: puntatore al numero di navi attualmente trovate ancra da af-
fondare
IP righe: numero di righe della griglia (dimensione uno di $comp_griglia_avv)
IP colonne: numero di colonne della griglia (dimensione due di
$comp_griglia_avv)
IP+OP direzione: direzione in cui stiamo attualmente colpendo
IP+OP verso: verso in cui ci stiamo attualmente muovendo
IP+OP indiceNave: puntatore alla ave da valutare
IP indicePosizione: posizione di una coordinata all'interno dell'array posizione
di una nave
IP naviincrociate: valore booleano che indice se abbiamo trovato due navi tra
loro incrociate
IP num_navi: numero di navi ancora da affondare
OP mossa da fare

*/
Coordinate mossaUnaCasella(Casella** comp_griglia_avv, Nave* navi_trovate, Info-
Navi* naviPartita, int nTipi, short *count_navi_trovate, short *navi_attuali,
int righe, int colonne, bool* direzione, bool* verso,  int* indiceNave, int
*indicePosizione, bool *navincrociate, int num_navi){
    Coordinate new_mossa;
    int i,j;
    bool mossaIncrociata = false;
    bool isAffondata = false;
    int caselle_affondate = navi_trovate[*indiceNave].caselleAff - na-
vi_trovate[*indiceNave].lunghezza;

    if(!caselle_affondate)
        caselle_affondate++;
    srand(time(NULL));
    if(!possibileColpo(comp_griglia_avv,
&navi_trovate[*indiceNave].posizione[caselle_affondate - 1], righe, colonne)){
        *direzione = rand() % 2;
        *verso = rand() % 2;
        for(i = 0; i < 2; i++){/* termina quando troviamo una possibile mossaIn-
crociata */
            *direzione = !(*direzione);
            for(j = 0; j < 2; j++){/* termina quando troviamo una possibile mos-
saIncrociata */
                new_mossa  = na-
vi_trovate[*indiceNave].posizione[caselle_affondate - 1];
                *verso = !(*verso);
                aumentaMossa(&new_mossa, *direzione, *verso, 1);
                if(isValid(&new_mossa, righe,colonne) &&
comp_griglia_avv[new_mossa.riga][new_mossa.colonna].contenuto){
```

```
                    mossaIncrociata = mossaNaviIncrociate(comp_griglia_avv, na-
vi_trovate, righe, colonne, *count_navi_trovate, *navi_attuali, &new_mossa,
*direzione, *verso, indiceNave, indicePosizione, navincrociate);
                }
                if(mossaIncrociata)
                    break;
            }
            if(mossaIncrociata)
                break;
        }
        if(!mossaIncrociata){
            /*FALLIMANTO MOSSA INCROCIATA*/
            do{
                new_mossa  = na-
vi_trovate[*indiceNave].posizione[caselle_affondate - 1];
                *direzione = rand() % 2;
                *verso = rand() % 2;
                aumentaMossa(&new_mossa, *direzione, *verso, 1);
                if(isValid(&new_mossa, righe,colonne) &&
comp_griglia_avv[new_mossa.riga][new_mossa.colonna].contenuto){
                    *indiceNave = cercaNavePosizione(navi_trovate,
*count_navi_trovate, *navi_attuali, indiceNave, indicePosizione, &new_mossa);
                    isAffondata = navi_trovate[*indiceNave].affondata;
                }
            }while(!isAffondata);

            aggiungiNavePartita(naviPartita, na-
vi_trovate[*indiceNave].caselleAff, nTipi);
            navi_trovate[*indiceNave].caselleAff++;
            realloc(navi_trovate[*indiceNave].posizione, sizeof(Coordinate) *
(navi_trovate[*indiceNave].caselleAff));
            assert(navi_trovate[*indiceNave].posizione != NULL);
            na-
vi_trovate[*indiceNave].posizione[navi_trovate[*indiceNave].caselleAff - 1] =
navi_trovate[*count_navi_trovate].posizione[0];
            navi_trovate[*count_navi_trovate].caselleAff--;
            spostaNavi(navi_trovate, *count_navi_trovate, *count_navi_trovate,
*navi_attuali);
            (*navi_attuali)--;
            scambiaCasel-
le(&navi_trovate[*indiceNave].posizione[navi_trovate[*indiceNave].caselleAff -
1], &navi_trovate[*indiceNave].posizione[navi_trovate[*indiceNave].caselleAff -
2] );
            if(!aggiornamentoAffondato(comp_griglia_avv, navi_trovate, naviPar-
tita, nTipi, navi_attuali, count_navi_trovate, indiceNave)){
                spostaNavi(navi_trovate, *indiceNave, *count_navi_trovate,
*navi_attuali);
                if(*indiceNave < *count_navi_trovate){
                    (*count_navi_trovate)--;
                    (*navi_attuali)++;
                }
            }
            *indiceNave = *count_navi_trovate;

            if(*navi_attuali){
                return mossa_ragionata(comp_griglia_avv, navi_trovate, naviPar-
tita, true, nTipi, num_navi, righe, colonne, navi_attuali, count_navi_trovate,
indiceNave, indicePosizione, navincrociate);
            }else{
                return mossa_random(comp_griglia_avv, righe, colonne);
            }

        }
```

```
    }else{
        do{
            new_mossa  = navi_trovate[*indiceNave].posizione[caselle_affondate -
1];
            *direzione = rand() % 2;
            *verso = rand() % 2;
            aumentaMossa(&new_mossa, *direzione, *verso, 1);
        }
        while(!(isValid(&new_mossa, righe,colonne) &&
!comp_griglia_avv[new_mossa.riga][new_mossa.colonna].colpito));
    }
    return new_mossa;
}


/*Funzione che restituisce la mossa da fare nel caso con piÃ¹ di una casella
giÃ  colpita e risultato della precedente mossa = ACQUA

IP comp_griglia_avv: array bidimensionale di caselle che rappresenta la griglia
in cui il computer fa le mosse per trovare le navi dell'avv
IP+OP navi_trovate: array in cui il computer salva le navi trovate
IP naviPartita: array con le informazioni sulle navi della partita
IP nTipi: dimensione dell'array $naviPartita
IP counter_contenuto: puntatore al valore che indice se abbiamo giÃ  trovato una
casella con contenuto o no
IP count_navi_trovate: puntatore al numero di navi affondate
IP righe: numero di righe della griglia (dimensione uno di $comp_griglia_avv)
IP colonne: numero di colonne della griglia (dimensione due di
$comp_griglia_avv)
IP direzione: direzione in cui stiamo attualmente colpendo
IP+OP verso: verso in cui ci stiamo attualmente muovendo
IP navi_attuali: puntatore al numero di navi attualmente trovate ancra da affon-
dare
IP+OP indiceNave: puntatore alla ave da valutare
IP indicePosizione: posizione di una coordinata all'interno dell'array posizione
di una nave
IP naviincrociate: valore booleano che indice se abbiamo trovato due navi tra
loro incrociate
IP num_navi: numero di navi ancora da affondare
OP mossa da fare

*/
Coordinate mossaPiuCaselleAcqua(Casella** comp_griglia_avv, Nave* navi_trovate,
InfoNavi* naviPartita, int nTipi, short* counter_contenuto, short
*count_navi_trovate, int righe, int colonne, bool* direzione, bool* verso,
short* navi_attuali,  int* indiceNave, int *indicePosizione, bool
*navincrociate, int num_navi){
    Coordinate new_mossa = na-
vi_trovate[*count_navi_trovate].posizione[navi_trovate[*count_navi_trovate].case
lleAff - 1];

    *verso = !(*verso);
    aumentaMossa(&new_mossa, *direzione, *verso, na-
vi_trovate[*count_navi_trovate].caselleAff);
    if(isValid(&new_mossa, righe,colonne) &&
comp_griglia_avv[new_mossa.riga][new_mossa.colonna].contenuto){
        if(*counter_contenuto){
            *verso = !(*verso);
            return mossaPiuCaselleColpito(comp_griglia_avv, navi_trovate, navi-
Partita, nTipi, counter_contenuto, count_navi_trovate, righe, colonne, direzio-
ne, verso, navi_attuali,indiceNave,indicePosizione, navincrociate, num_navi);
        }else{
```

```
            if(!mossaNaviIncrociate(comp_griglia_avv, navi_trovate, righe, co-
lonne, *count_navi_trovate, *navi_attuali, &new_mossa, *direzione,
*verso,indiceNave,indicePosizione, navincrociate))
                return mossaPiuCaselleAcqua(comp_griglia_avv, navi_trovate, na-
viPartita, nTipi, counter_contenuto, count_navi_trovate, righe, colonne, dire-
zione, verso, navi_attuali,indiceNave,indicePosizione, navincrociate, num_navi);
        }
    }else if(!isValid(&new_mossa, righe,colonne) ||
comp_griglia_avv[new_mossa.riga][new_mossa.colonna].colpito){
        if(*counter_contenuto){
            *verso = !(*verso);
            return mossaPiuCaselleColpito(comp_griglia_avv, navi_trovate, navi-
Partita, nTipi, counter_contenuto, count_navi_trovate, righe, colonne, direzio-
ne, verso, navi_attuali,indiceNave,indicePosizione, navincrociate, num_navi);
        }else{
            eccezione_navi_affiancate(comp_griglia_avv, navi_trovate, naviParti-
ta, nTipi, count_navi_trovate, navi_attuali, dimensioneMax(righe,colonne));
            *indiceNave = *count_navi_trovate;
            return mossaUnaCasella(comp_griglia_avv, navi_trovate, naviPartita,
nTipi, count_navi_trovate, navi_attuali, righe, colonne, direzione,verso, indi-
ceNave, indicePosizione, navincrociate, num_navi);
        }
    }

    ordinaNave(&navi_trovate[*count_navi_trovate]);
    /*scambiaCaselle*/
    scambiaCaselle(&navi_trovate[*count_navi_trovate].posizione[0],
&navi_trovate[*count_navi_trovate].posizione[navi_trovate[*count_navi_trovate].c
aselleAff-1]);

    (*counter_contenuto) = 0;
    return new_mossa;
}/* mossaPiuCaselleAcqua */


/*Funzione che restituisce la mossa da fare nel caso con piÃ¹ di una casella
giÃ  colpita e risultato della precedente mossa = COLPITO

IP comp_griglia_avv: array bidimensionale di caselle che rappresenta la griglia
in cui il computer fa le mosse per trovare le navi dell'avv
IP navi_trovate: array in cui il computer salva le navi trovate
IP+OP naviPartita: array con le informazioni sulle navi della partita
IP nTipi: dimensione dell'array $naviPartita
IP+OP counter_contenuto: puntatore al valore che indice se abbiamo giÃ  trovato
una casella con contenuto o no
IP count_navi_trovate: puntatore al numero di navi affondate
IP righe: numero di righe della griglia (dimensione uno di $comp_griglia_avv)
IP colonne: numero di colonne della griglia (dimensione due di
$comp_griglia_avv)
IP direzione: direzione in cui stiamo attualmente colpendo
IP verso: verso in cui ci stiamo attualmente muovendo
IP navi_attuali: puntatore al numero di navi attualmente trovate ancra da affon-
dare
IP indiceNave: puntatore alla ave da valutare
IP indicePosizione: posizione di una coordinata all'interno dell'array posizione
di una nave
IP naviincrociate: valore booleano che indice se abbiamo trovato due navi tra
loro incrociate
IP num_navi: numero di navi ancora da affondare
OP mossa da fare

*/
```

```
Coordinate mossaPiuCaselleColpito(Casella** comp_griglia_avv, Nave* na-
vi_trovate, InfoNavi* naviPartita, int nTipi, short* counter_contenuto, short
*count_navi_trovate, int righe, int colonne, bool* direzione, bool* verso,
short* navi_attuali,  int* indiceNave, int *indicePosizione, bool
*navincrociate, int num_navi){
    Coordinate new_mossa = na-
vi_trovate[*count_navi_trovate].posizione[navi_trovate[*indiceNave].caselleAff -
1];
    trovaDirezioneVersoNave(&navi_trovate[*count_navi_trovate], direzione, ver-
so);
    aumentaMossa(&new_mossa, *direzione, *verso, 1);
    if(isValid(&new_mossa, righe,colonne) &&
comp_griglia_avv[new_mossa.riga][new_mossa.colonna].contenuto){
        if(*counter_contenuto){
            (*counter_contenuto) = 0;
            if(!mossaNaviIncrociate(comp_griglia_avv, navi_trovate, righe, co-
lonne, *count_navi_trovate, *navi_attuali, &new_mossa, *direzione,
*verso,indiceNave,indicePosizione, navincrociate)){
                return mossaPiuCaselleAcqua(comp_griglia_avv, navi_trovate, na-
viPartita, nTipi, counter_contenuto, count_navi_trovate, righe, colonne, dire-
zione, verso, navi_attuali,indiceNave,indicePosizione, navincrociate, num_navi);
            }
        }else{
            (*counter_contenuto) = 1;
            return mossaPiuCaselleAcqua(comp_griglia_avv, navi_trovate, naviPar-
tita, nTipi, counter_contenuto, count_navi_trovate, righe, colonne, direzione,
verso, navi_attuali,indiceNave,indicePosizione, navincrociate, num_navi);
        }
    }else if(!isValid(&new_mossa, righe,colonne) ||
comp_griglia_avv[new_mossa.riga][new_mossa.colonna].colpito){
        (*counter_contenuto) = 0;
        return mossaPiuCaselleAcqua(comp_griglia_avv, navi_trovate, naviPartita,
nTipi, counter_contenuto, count_navi_trovate, righe, colonne, direzione, verso,
navi_attuali,indiceNave,indicePosizione, navincrociate, num_navi);
    }
    return new_mossa;
}/* mossaPiuCaselleColpito */



/*Funzione che restituisce la mossa da fare nel caso con piÃ¹ di una casella
giÃ  colpita

IP comp_griglia_avv: array bidimensionale di caselle che rappresenta la griglia
in cui il computer fa le mosse per trovare le navi dell'avv
IP navi_trovate: array in cui il computer salva le navi trovate
IP naviPartita: array con le informazioni sulle navi della partita
IP nTipi: dimensione dell'array $naviPartita
IP res_prec_mossa: valore che indica se la mossa precedente era ACQUA o COLPITO
IP righe: numero di righe della griglia (dimensione uno di $comp_griglia_avv)
IP colonne: numero di colonne della griglia (dimensione due di
$comp_griglia_avv)
IP navi_attuali: puntatore al numero di navi attualmente trovate ancra da affon-
dare
IP count_navi_trovate: puntatore al numero di navi affondate
IP counter_contenuto: puntatore al valore che indice se abbiamo giÃ  trovato una
casella con contenuto o no
IP direzione: direzione in cui stiamo attualmente colpendo
IP verso: verso in cui ci stiamo attualmente muovendo
IP indiceNave: puntatore alla ave da valutare
IP indicePosizione: posizione di una coordinata all'interno dell'array posizione
di una nave
```

```
IP naviincrociate: valore booleano che indice se abbiamo trovato due navi tra
loro incrociate
IP num_navi: numero di navi ancora da affondare
OP mossa da fare

*/
Coordinate mossaPiuCaselle(Casella** comp_griglia_avv, Nave* navi_trovate, Info-
Navi* naviPartita, int nTipi, bool res_prec_mossa, int righe, int colonne,
short* navi_attuali, short *count_navi_trovate, short* counter_contenuto, bool*
direzione, bool*verso, int* indiceNave, int *indicePosizione, bool
*navincrociate, int num_navi){
    if(res_prec_mossa){
        return mossaPiuCaselleColpito(comp_griglia_avv, navi_trovate, naviParti-
ta, nTipi, counter_contenuto, count_navi_trovate, righe, colonne, direzione,
verso, navi_attuali,indiceNave,indicePosizione, navincrociate, num_navi);
    }else{
        return mossaPiuCaselleAcqua(comp_griglia_avv, navi_trovate, naviPartita,
nTipi, counter_contenuto, count_navi_trovate, righe, colonne, direzione, verso,
navi_attuali,indiceNave,indicePosizione, navincrociate, num_navi);
    }
}/* mossaPiuCaselle */


/*Funzione che cerca navi nelle vicinanze della nave di dimensioni < min appena
trovata

IP comp_griglia_avv: array bidimensionale di caselle che rappresenta la griglia
in cui il computer fa le mosse per trovare le navi dell'avv
IP navi_trovate: array in cui il computer salva le navi trovate
IP naviPartita: array con le informazioni sulle navi della partita
IP+OP count_navi_trovate: puntatore al numero di navi affondate
IP nTipi: dimensione dell'array $naviPartita
IP righe: numero di righe della griglia (dimensione uno di $griglia e
$comp_griglia_avv)
IP colonne: numero di colonne della griglia (dimensione due di $griglia e
$comp_griglia_avv)
IP+OP navi_attuali: puntatore al numero di navi attualmente trovate ancra da af-
fondare
IP+OP direzione: direzione in cui stiamo attualmente colpendo
IP+OP verso: verso in cui ci stiamo attualmente muovendo
IP+OP indiceNave: puntatore alla ave da valutare
IP indicePosizione: posizione di una coordinata all'interno dell'array posizione
di una nave
IP+OP indiceNave2: puntatore alla seconda nave da verificare
OP mossa da fare

*/
void eccezioneNavePiccolaNew(Casella** comp_griglia_avv,Nave* na-
vi_trovate,InfoNavi* naviPartita, short* count_navi_trovate, int nTipi, int ri-
ghe, int colonne, short* navi_attuali , bool* direzione, bool* verso, int* indi-
ceNave, int* indicePosizione, int* indiceNave2){

    Coordinate new_mossa;
    bool direzioneNave, versoNave;
    int contagiri = 0;
    int indiceNave_temp;
    new_mossa = na-
vi_trovate[*indiceNave].posizione[navi_trovate[*indiceNave].caselleAff - 1];
    *indiceNave = *count_navi_trovate + *navi_attuali;
    indiceNave_temp = *indiceNave;;
    while(contagiri < 2){
        *indiceNave = cercaNaveVicina(navi_trovate, &new_mossa,
*count_navi_trovate, *navi_attuali, indiceNave, indicePosizione);
```

```
        if(*indiceNave == -1){
            if(contagiri == 0)
                *indiceNave = indiceNave_temp;
            contagiri++;
            continue; /* se l'indice della nave Ã¨ -1 ricominciamo il giro senza
fare le operazioni successive*/
        }
        trovaDirezioneVersoNave(&navi_trovate[*indiceNave], &direzioneNave,
&versoNave);
        trovaDirezioneVerso(&new_mossa,
&navi_trovate[*indiceNave].posizione[*indicePosizione] , direzione, verso);
        if((*direzione && direzioneNave) || (!(*direzione) && !direzioneNave)){
            if(contagiri == 0){
                if(*indiceNave < *count_navi_trovate)
                    aggiungiNavePartita(naviPartita, na-
vi_trovate[*indiceNave].caselleAff, nTipi);
                if(eccezioneNaviInFilaDef(comp_griglia_avv, navi_trovate, navi-
Partita, nTipi, count_navi_trovate, navi_attuali, indiceNave, indicePosizione,
righe, colonne, indiceNave2)){
                    return;
                }else{
                    spostaNavi(navi_trovate, *indiceNave, *count_navi_trovate,
*navi_attuali);
                    if(*indiceNave < *count_navi_trovate){
                        (*count_navi_trovate)--;
                        (*navi_attuali)++;
                    }
                }
            }
        }else if(contagiri == 1){
            if(eccezioneNaviIncrociate(comp_griglia_avv, navi_trovate, naviPar-
tita, nTipi, count_navi_trovate, navi_attuali, indiceNave, indicePosizione, ri-
ghe, colonne, *indiceNave2))
                *indiceNave = *indiceNave2;
            else{
                if(*indiceNave < *indiceNave2)
                    *indiceNave = *indiceNave2 - 1;
                else
                    *indiceNave = *indiceNave2;
            }
            return;
        }

    }
}/* eccezioneNavePiccolaNew */



/*Funzione che controlla la presenza di altre navi nella direzione attuale

IP comp_griglia_avv: array bidimensionale di caselle che rappresenta la griglia
in cui il computer fa le mosse per trovare le navi dell'avv
IP navi_trovate: array in cui il computer salva le navi trovate
IP naviPartita: array con le informazioni sulle navi della partita
IP+OP count_navi_trovate: puntatore al numero di navi affondate
IP nTipi: dimensione dell'array $naviPartita
IP righe: numero di righe della griglia (dimensione uno di $griglia e
$comp_griglia_avv)
IP colonne: numero di colonne della griglia (dimensione due di $griglia e
$comp_griglia_avv)
IP+OP navi_attuali: puntatore al numero di navi attualmente trovate ancra da af-
fondare
IP+OP direzione: direzione in cui stiamo attualmente colpendo
```

```
IP+OP verso: verso in cui ci stiamo attualmente muovendo
IP+OP indiceNave: puntatore alla ave da valutare
IP indicePosizione: posizione di una coordinata all'interno dell'array posizione
di una nave
IP+OP indiceNave2: puntatore alla seconda nave da verificare
OP mossa da fare

*/
void controlloStessaDirezione(Casella** comp_griglia_avv,Nave* na-
vi_trovate,InfoNavi* naviPartita, short* count_navi_trovate, int nTipi, int ri-
ghe, int colonne, short* navi_attuali , bool* direzione, bool* verso, int* indi-
ceNave, int* indicePosizione, int* indiceNave2){

    Coordinate new_mossa;
    bool direzioneNave, versoNave;
    int contagiri = 0;

    while((contagiri < 2) && (*indiceNave != -1)){

        /* cerco dalla parte opposta all'affondato */
        new_mossa  = navi_trovate[*indiceNave].posizione[0];
        trovaDirezioneVersoNave(&navi_trovate[*indiceNave], direzione, verso);
        *verso = !(*verso);
        aumentaMossa(&new_mossa, *direzione, *verso, 1);
        if(isValid(&new_mossa, righe,colonne) &&
comp_griglia_avv[new_mossa.riga][new_mossa.colonna].contenuto){
            *indiceNave = cercaNavePosizione(navi_trovate, *count_navi_trovate,
*navi_attuali, indiceNave, indicePosizione, &new_mossa);
            trovaDirezioneVersoNave(&navi_trovate[*indiceNave], &direzioneNave,
&versoNave);
            if((*direzione && direzioneNave) || (!(*direzione) && !direzioneNa-
ve)){
                if(contagiri == 0){
                    if(*indiceNave < *count_navi_trovate)
                        aggiungiNavePartita(naviPartita, na-
vi_trovate[*indiceNave].caselleAff, nTipi);
                    if(eccezioneNaviInFilaDef(comp_griglia_avv, navi_trovate,
naviPartita, nTipi, count_navi_trovate, navi_attuali, indiceNave, indicePosizio-
ne, righe, colonne, indiceNave2)){
                        return;
                    }else{
                        spostaNavi(navi_trovate, *indiceNave,
*count_navi_trovate, *navi_attuali);
                        if(*indiceNave < *count_navi_trovate){
                            (*count_navi_trovate)--;
                            (*navi_attuali)++;
                        }
                        if(*indiceNave < *indiceNave2)
                            *indiceNave = *indiceNave2 - 1;
                        else
                            *indiceNave = *indiceNave2;
                    }
                }else{
                    *indiceNave = *indiceNave2;
                }
            }else if((contagiri == 1) && (*indicePosizione == 0 ||
*indicePosizione == navi_trovate[*indiceNave].caselleAff-1)){
                if(eccezioneNaviIncrociate(comp_griglia_avv, navi_trovate, navi-
Partita, nTipi, count_navi_trovate, navi_attuali, indiceNave, indicePosizione,
righe, colonne, *indiceNave2))
                    *indiceNave = *indiceNave2;
                else{
                    if(*indiceNave < *indiceNave2)
```

```
                        *indiceNave = *indiceNave2 - 1;
                    else
                        *indiceNave = *indiceNave2;
                }
                return;
            }
        }
        /* cerco dalla parte dell'affondato */
        new_mossa  = na-
vi_trovate[*indiceNave].posizione[navi_trovate[*indiceNave].caselleAff - 1];
        *verso = !(*verso);
        aumentaMossa(&new_mossa, *direzione, *verso, 1);
        if(isValid(&new_mossa, righe,colonne) &&
comp_griglia_avv[new_mossa.riga][new_mossa.colonna].contenuto){
            *indiceNave = cercaNavePosizione(navi_trovate, *count_navi_trovate,
*navi_attuali, indiceNave, indicePosizione, &new_mossa);
            trovaDirezioneVersoNave(&navi_trovate[*indiceNave], &direzioneNave,
&versoNave);
            if((*direzione && direzioneNave) || (!(*direzione) && !direzioneNa-
ve)){
                if(contagiri == 0){
                    if(*indiceNave < *count_navi_trovate)
                        aggiungiNavePartita(naviPartita, na-
vi_trovate[*indiceNave].caselleAff, nTipi);
                    if(eccezioneNaviInFilaDef(comp_griglia_avv, navi_trovate,
naviPartita, nTipi, count_navi_trovate, navi_attuali, indiceNave, indicePosizio-
ne, righe, colonne, indiceNave2)){
                        return;
                    }else{
                        spostaNavi(navi_trovate, *indiceNave,
*count_navi_trovate, *navi_attuali);
                        if(*indiceNave < *count_navi_trovate){
                            (*count_navi_trovate)--;
                            (*navi_attuali)++;
                        }
                        if(*indiceNave < *indiceNave2)
                            *indiceNave = *indiceNave2 - 1;
                        else
                            *indiceNave = *indiceNave2;
                    }
                }
            }else if((contagiri == 1) && (*indicePosizione == 0 ||
*indicePosizione == navi_trovate[*indiceNave].caselleAff-1)){
                if(eccezioneNaviIncrociate(comp_griglia_avv, navi_trovate, navi-
Partita, nTipi, count_navi_trovate, navi_attuali, indiceNave, indicePosizione,
righe, colonne, *indiceNave2))
                    *indiceNave = *indiceNave2;
                else{
                    if(*indiceNave < *indiceNave2)
                        *indiceNave = *indiceNave2 - 1;
                    else
                        *indiceNave = *indiceNave2;
                }
                return;
            }else{
                *indiceNave = *indiceNave2;
            }
        }
        *indiceNave = cercaNave(navi_trovate, *count_navi_trovate, na-
vi_trovate[*indiceNave].caselleAff, *indiceNave);

        if((*indiceNave == -1) && (contagiri == 0)){
            *indiceNave = *count_navi_trovate;
```

```
            contagiri++;
        }
        *indiceNave2 = *indiceNave;
    }

}/* controlloStessaDirezione */

/*Funzione che controlla la presenza di altre navi nella direzione opposta a
quella attuale

IP comp_griglia_avv: array bidimensionale di caselle che rappresenta la griglia
in cui il computer fa le mosse per trovare le navi dell'avv
IP navi_trovate: array in cui il computer salva le navi trovate
IP naviPartita: array con le informazioni sulle navi della partita
IP nTipi: dimensione dell'array $naviPartita
IP count_navi_trovate: puntatore al numero di navi affondate
IP righe: numero di righe della griglia (dimensione uno di $griglia e
$comp_griglia_avv)
IP colonne: numero di colonne della griglia (dimensione due di $griglia e
$comp_griglia_avv)
IP navi_attuali: puntatore al numero di navi attualmente trovate ancra da affon-
dare
IP+OP direzione: direzione in cui stiamo attualmente colpendo
IP+OP verso: verso in cui ci stiamo attualmente muovendo
IP+OP indiceNave: puntatore alla ave da valutare
IP indicePosizione: posizione di una coordinata all'interno dell'array posizione
di una nave
IP+OP indiceNave2: puntatore alla seconda nave da verificare
OP mossa da fare

*/
void controlloDirezioneIncrociata(Casella** comp_griglia_avv,Nave* na-
vi_trovate,InfoNavi* naviPartita, int nTipi, short* count_navi_trovate, int ri-
ghe, int colonne, short* navi_attuali , bool* direzione, bool* verso, int* indi-
ceNave, int* indicePosizione, int* indiceNave2){
    int i;
    Coordinate new_mossa;

    while(*indiceNave != -1){
        new_mossa  = navi_trovate[*indiceNave].posizione[0];
        trovaDirezioneVersoNave(&navi_trovate[*indiceNave], direzione, verso);
        *direzione = !(*direzione);
        *verso = rand() % 2;
        for(i = 0; i < 2; i++){
            new_mossa  = navi_trovate[*indiceNave].posizione[0];
            *verso = !(*verso);
            aumentaMossa(&new_mossa, *direzione, *verso, 1);
            if(isValid(&new_mossa, righe,colonne) &&
comp_griglia_avv[new_mossa.riga][new_mossa.colonna].contenuto){
                *indiceNave = cercaNavePosizione(navi_trovate,
*count_navi_trovate, *navi_attuali, indiceNave, indicePosizione, &new_mossa);
                if(eccezioneNaviIncrociateReverse(comp_griglia_avv, na-
vi_trovate, naviPartita, nTipi, count_navi_trovate, navi_attuali, indiceNave,
indicePosizione, righe, colonne, indiceNave2))
                    *indiceNave = *indiceNave2;
                else{
                    if(*indiceNave < *indiceNave2)
                        *indiceNave = *indiceNave2 - 1;
                    else
                        *indiceNave = *indiceNave2;
                }
                return;
            }
```

```
        }

        *indiceNave = cercaNave(navi_trovate, *count_navi_trovate, na-
vi_trovate[*indiceNave].caselleAff, *indiceNave);
        *indiceNave2 = *indiceNave;
    }
}/* controlloDirezioneIncrociata */


/*Funzione che restituisce la mossa da fare nel caso di navi doppie

IP comp_griglia_avv: array bidimensionale di caselle che rappresenta la griglia
in cui il computer fa le mosse per trovare le navi dell'avv
IP+OP navi_trovate: array in cui il computer salva le navi trovate
IP naviPartita: array con le informazioni sulle navi della partita
IP res_prec_mossa: valore che indica se la mossa precedente era ACQUA o COLPITO
IP+OP count_navi_trovate: puntatore al numero di navi affondate
IP nTipi: dimensione dell'array $naviPartita
IP righe: numero di righe della griglia (dimensione uno di $griglia e
$comp_griglia_avv)
IP colonne: numero di colonne della griglia (dimensione due di $griglia e
$comp_griglia_avv)
IP+OP navi_attuali: puntatore al numero di navi attualmente trovate ancra da af-
fondare
IP direzione: direzione in cui stiamo attualmente colpendo
IP verso: verso in cui ci stiamo attualmente muovendo
IP+OP indiceNave: puntatore alla ave da valutare
IP indicePosizione: posizione di una coordinata all'interno dell'array posizione
di una nave
IP naviincrociate: valore booleano che indice se abbiamo trovato due navi tra
loro incrociate
IP num_navi: numero di navi ancora da affondare
OP mossa da fare

*/
Coordinate mossaNaviDoppie(Casella** comp_griglia_avv,Nave* na-
vi_trovate,InfoNavi* naviPartita ,bool res_prec_mossa, short*
count_navi_trovate, int nTipi,
                           int righe, int colonne, short* navi_attuali , bool*
direzione, bool* verso, int* indiceNave, int* indicePosizione, bool
*navincrociate, int num_navi){

    int indiceNave2 = *count_navi_trovate;
    srand(time(NULL));
    if(navi_trovate[*indiceNave].caselleAff == 1 || na-
vi_trovate[*indiceNave].caselleAff < naviPartita[nTipi-1].tipo){
        eccezioneNavePiccolaNew(comp_griglia_avv, navi_trovate, naviPartita,
count_navi_trovate, nTipi, righe, colonne, navi_attuali, direzione, verso, indi-
ceNave, indicePosizione, &indiceNave2);
    }else{
        if(possibileColpo(comp_griglia_avv,
&navi_trovate[*count_navi_trovate].posizione[0], righe, colonne)){
            return mossaUnaCasella(comp_griglia_avv, navi_trovate, naviPartita,
nTipi, count_navi_trovate, navi_attuali, righe, colonne, direzione, verso, indi-
ceNave, indicePosizione, navincrociate, num_navi);
        }else{
            do{
                *indiceNave = cercaNave(navi_trovate, *count_navi_trovate, na-
vi_trovate[*indiceNave].caselleAff, *indiceNave);
            }while(!(*indiceNave == -1) && !possibileColpo(comp_griglia_avv,
&navi_trovate[*indiceNave].posizione[0], righe, colonne));
            if(*indiceNave != -1)
```

```
                return mossaUnaCasella(comp_griglia_avv, navi_trovate, naviPar-
tita, nTipi, count_navi_trovate, navi_attuali, righe,  colonne, direzione, ver-
so, indiceNave, indicePosizione, navincrociate, num_navi);
            *indiceNave = *count_navi_trovate;

            controlloStessaDirezione(comp_griglia_avv, navi_trovate, naviParti-
ta, count_navi_trovate, nTipi, righe, colonne, navi_attuali, direzione, verso,
indiceNave, indicePosizione, &indiceNave2);

            if(*indiceNave == -1){
                *indiceNave = *count_navi_trovate;
                indiceNave2 = *indiceNave;
                controlloDirezioneIncrociata(comp_griglia_avv, navi_trovate, na-
viPartita, nTipi, count_navi_trovate, righe, colonne, navi_attuali, direzione,
verso, indiceNave, indicePosizione, &indiceNave2);
            }

        }
    }

    if(*indiceNave < *count_navi_trovate){
            (*navi_attuali)--;
            (*count_navi_trovate)++;
    }else if(*indiceNave > *count_navi_trovate){
        scambiaNavi(&navi_trovate[*indiceNave],
&navi_trovate[*count_navi_trovate]);
        *indiceNave = *count_navi_trovate;
    }

    if((*indiceNave) != -1){
        if(!aggiornamentoAffondato(comp_griglia_avv, navi_trovate, naviPartita,
nTipi, navi_attuali, count_navi_trovate, indiceNave)){
            spostaNavi(navi_trovate, *indiceNave, *count_navi_trovate,
*navi_attuali);
            if(*indiceNave < *count_navi_trovate){
                (*navi_attuali)++;
                (*count_navi_trovate)--;
            }
        }
    }

    if((num_navi == 1) && (*navi_attuali != 0)){
        spostaNavi(navi_trovate, *count_navi_trovate, *count_navi_trovate,
*navi_attuali);
        *count_navi_trovate += *navi_attuali;
        *navi_attuali = 0;
    }
    *indiceNave = *count_navi_trovate;

    if(*navi_attuali){
       return mossa_ragionata(comp_griglia_avv, navi_trovate, naviPartita,
res_prec_mossa, nTipi, num_navi, righe, colonne, navi_attuali,
count_navi_trovate, indiceNave, indicePosizione, navincrociate);
    }else{
        return mossa_random(comp_griglia_avv, righe, colonne);
    }
}/* mossaNavoDoppie */



/*Funzione che restituisce la mossa da fare in base alla situazione
```

```
IP comp_griglia_avv: array bidimensionale di caselle che rappresenta la griglia
in cui il computer fa le mosse per trovare le navi dell'avv
IP navi_trovate: array in cui il computer salva le navi trovate
IP+OP naviPartita: array con le informazioni sulle navi della partita
IP res_prec_mossa: valore che indica se la mossa precedente era ACQUA o COLPITO
IP nTipi: dimensione dell'array $naviPartita
IP num_navi: numero di navi ancora da affondare
IP righe: numero di righe della griglia (dimensione uno di $griglia e
$comp_griglia_avv)
IP colonne: numero di colonne della griglia (dimensione due di $griglia e
$comp_griglia_avv)
IP+OP navi_attuali: puntatore al numero di navi attualmente trovate ancra da af-
fondare
IP+OP count_navi_trovate: puntatore al numero di navi affondate
IP indiceNave: puntatore alla ave da valutare
IP indicePosizione: posizione di una coordinata all'interno dell'array posizione
di una nave
IP naviincrociate: valore booleano che indice se abbiamo trovato due navi tra
loro incrociate
OP mossa da fare

*/
Coordinate mossa_ragionata(Casella** comp_griglia_avv,Nave* navi_trovate, Info-
Navi* naviPartita,bool res_prec_mossa, int nTipi, int num_navi,
                           int righe, int colonne, short* navi_attuali, short
*count_navi_trovate, int *indiceNave, int *indicePosizione, bool* navincrocia-
te){
    static short counter_contenuto = 0;
    static bool direzione = 0;
    static bool verso = 0;
    int indiceNaviPartita;
    stam-
pa_mossa(&navi_trovate[*count_navi_trovate].posizione[navi_trovate[*count_navi_t
rovate].caselleAff - 1]);
    if(navi_trovate[*count_navi_trovate].affondata){
        if(controlloTipoNave(naviPartita, nTipi, na-
vi_trovate[*count_navi_trovate].caselleAff, &indiceNaviPartita)){
            naviPartita[indiceNaviPartita].nNav--;
            (*count_navi_trovate)++;
            (*navi_attuali)--;
            if(*navi_attuali)
                return mossa_ragionata(comp_griglia_avv, navi_trovate, naviPar-
tita, res_prec_mossa, nTipi, num_navi, righe, colonne, navi_attuali,
count_navi_trovate, indiceNave, indicePosizione, navincrociate);
            else
                return mossa_random(comp_griglia_avv, righe, colonne);
        }else
            return mossaNaviDoppie(comp_griglia_avv, navi_trovate, naviPartita,
res_prec_mossa, count_navi_trovate, nTipi, righe, colonne, navi_attuali,
&direzione, &verso, indiceNave, indicePosizione, navincrociate, num_navi);
    }
    if(navi_trovate[*count_navi_trovate].caselleAff == 1){
        return mossaUnaCasella(comp_griglia_avv, navi_trovate, naviPartita, nTi-
pi, count_navi_trovate, navi_attuali, righe, colonne, &direzione, &verso, indi-
ceNave, indicePosizione, navincrociate, num_navi);

    }else{
        return mossaPiuCaselle(comp_griglia_avv, navi_trovate, naviPartita, nTi-
pi, res_prec_mossa, righe, colonne, navi_attuali, count_navi_trovate,
&counter_contenuto, &direzione, &verso, indiceNave, indicePosizione, navincro-
ciate, num_navi);
    }
}/*mossa_ragionata*/
```

Il file Mossa.h contiene le dichiarazioni delle funzioni di Mossa.c e MossaRagionata.c

```
#include "Generale.h"

void aggiungiNavePartita(InfoNavi*, int, int);

void eccezioneNaviDoppie(Casella**, Nave*, InfoNavi* , Coordinate*, int, short*,
short*, int*, int*, int, int);

bool eccezioneNaviIncrociate(Casella**, Nave*, InfoNavi* , int, short*, short*,
int*, int*, int, int, int);

void eccezioneNaviIncrociateMultiple(Casella**, Nave*, InfoNavi*, Coordinate*,
int, short*, short*, int*, int*, int, int, int);

Coordinate mossa_ragionata(Casella**, Nave*, InfoNavi*, bool, int, int, int,
int, short*, short*, int*, int*, bool*);

short eseguiMossa(Casella** , Casella** , Nave*, InfoNavi* ,int ,  int , int,
int);

Coordinate mossa_random(Casella** , int , int );

bool possibileColpo(Casella**, Coordinate*, int, int);

void spostaNavi(Nave*, int, short, short);

bool aggiornamentoAffondato(Casella** ,Nave*, InfoNavi*, int, short*, short*,
int*);

void stampa_mossa(const Coordinate*);

int cercaNave(const Nave*, short, short, int);

int cercaPosizione(const Nave*, const Coordinate*);

int cercaNavePosizione(const Nave*, short, short, int*, int*, const Coordina-
te*);

bool appartieneNave(Nave*, Coordinate*, int*);

bool isValid(const Coordinate*, int, int);

Coordinate mossaPiuCaselleColpito(Casella**, Nave*, InfoNavi*, int, short*,
short*, int, int, bool*, bool*, short*,  int*, int*, bool*, int);

int trovaDirezioneVerso(const Coordinate*, const Coordinate*, bool*, bool*);

bool controlloTipoNave(InfoNavi*,short, short, int*);

void scambiaCaselle(Coordinate*, Coordinate*);

void spostaNaviAvati(Nave*, short, short);

short mossaGiocatore(Casella **, Casella **, int , int );

void aumentaMossa(Coordinate*, bool, bool, int);

void scambiaNavi(Nave*, Nave*);
```

SITOGRAFIA

- https://www.gamesplus.it/
- https://www.barinedita.it/
- https://poki.com/it/g/battleships-armada