# Relazione per “Il Sudoku”

Si cominci con l’illustrazione della descrizione del progetto:

| | | |
|---|---|---|
| **Titolo Progetto:** | IlSudoku | |
| **Nome del programmatore:** | Giuseppe Carlino | |
| **Specifiche tecniche:** | Linguaggio (con versione): | C99 |
| | IDE (con versione): | CLion 2023.2.2 |
| **Descrizione:** | IlSudoku è un videogioco di logica multipiattaforma basato sul classico gioco del sudoku. Lo scopo è quello di ricreare la griglia numerica preimpostata, inserendo i numeri nelle apposite caselle vuote. Il valore nelle caselle presente a inizio partita (a differenza di quello che viene assegnato dal giocatore alle caselle vuote) non è modificabile, perché in tal modo si aiuta il giocatore nel raggiungimento della vittoria. | |

Si mostri anche la guida del videogioco:

# Guida per "IlSudoku"

REGOLE:

Lo scopo del gioco è quello di far compilare la griglia del Sudoku al giocatore;

Il posizionamento della caselle va in ordine dall'alto verso il basso e da sinistra verso destra (dalla n. 1 alla n. 81);

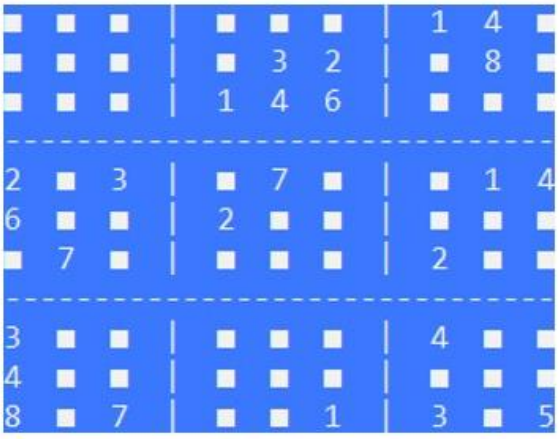
Il giocatore può scegliere quale casella compilare;

Inoltre il contenuto di una casella è sovrascrivibile (solo se non è stata già precompilata);

Per vincere, il giocatore deve inserire i numeri nella griglia facendo in modo che questi non si ripetano in quella stessa riga, in quella stessa colonna e in quello stesso blocco 3x3;

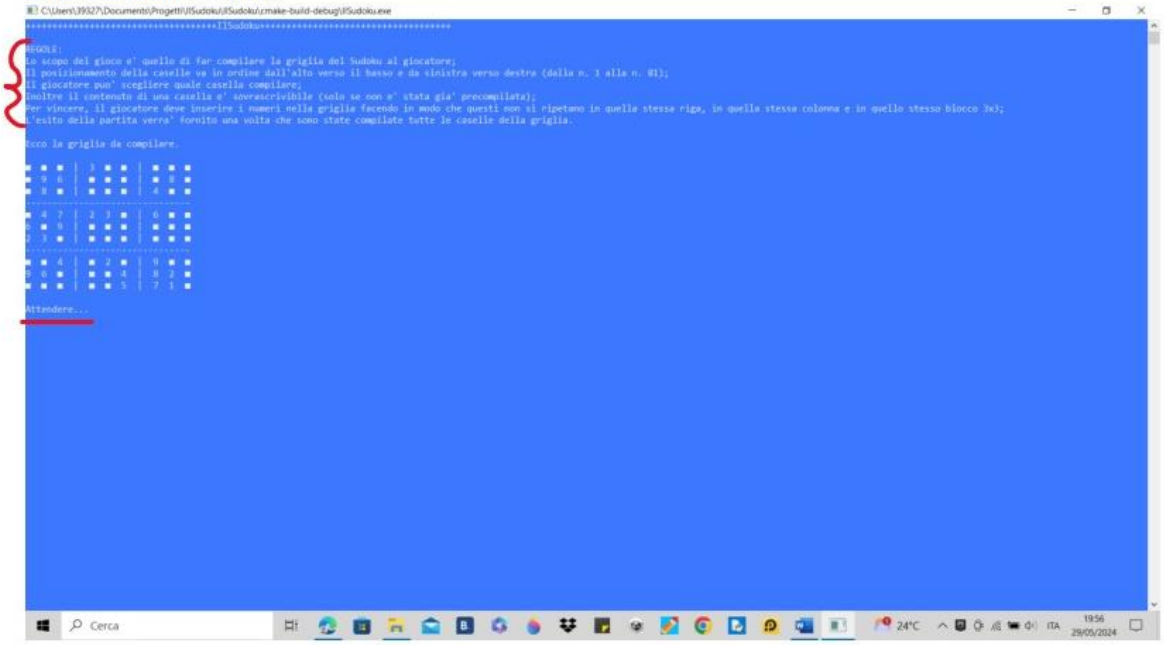
L'esito della partita verrà fornito una volta che sono state compilate tutte le caselle della griglia.

Ecco la griglia da compilare.

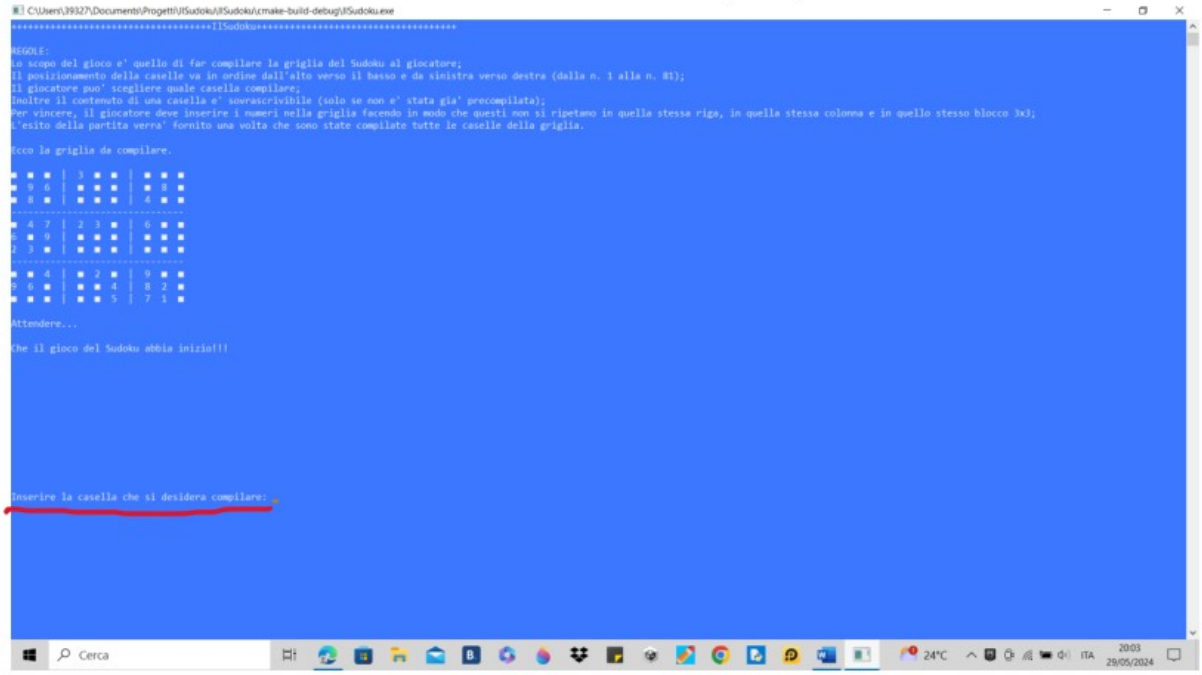


Ecco illustrato il videogioco nei suoi passaggi:

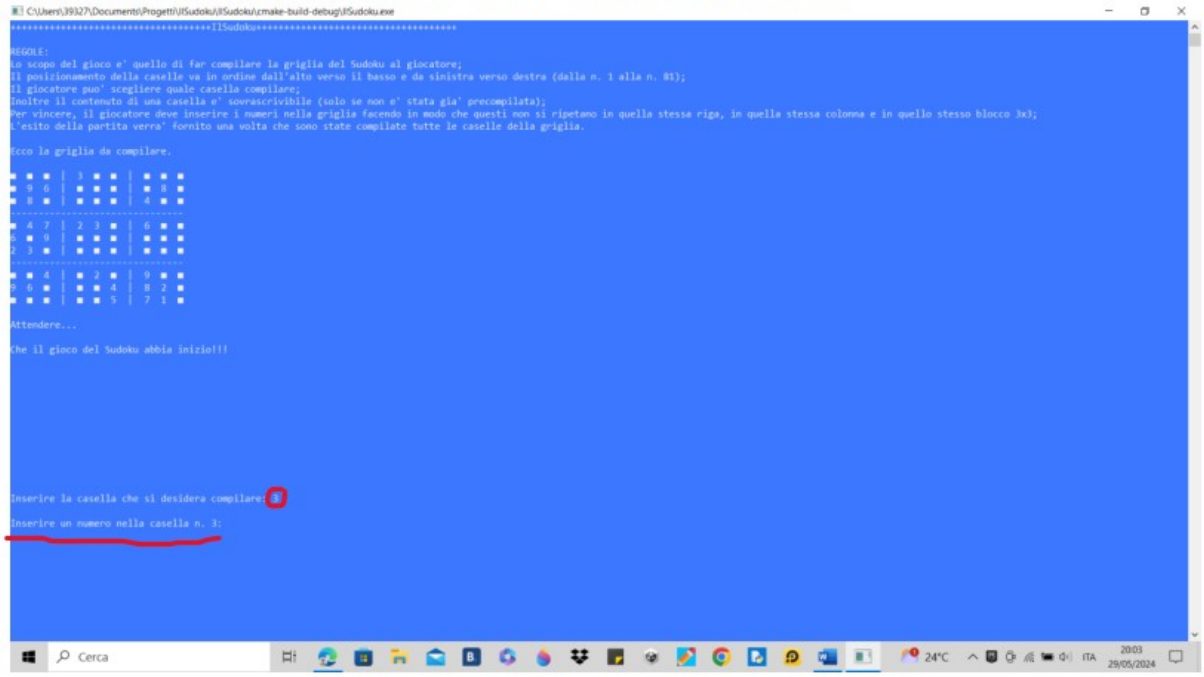
- Qui è come si mostra il software in partenza (la pausa serve al giocatore per ripassare le regole indicate sopra);



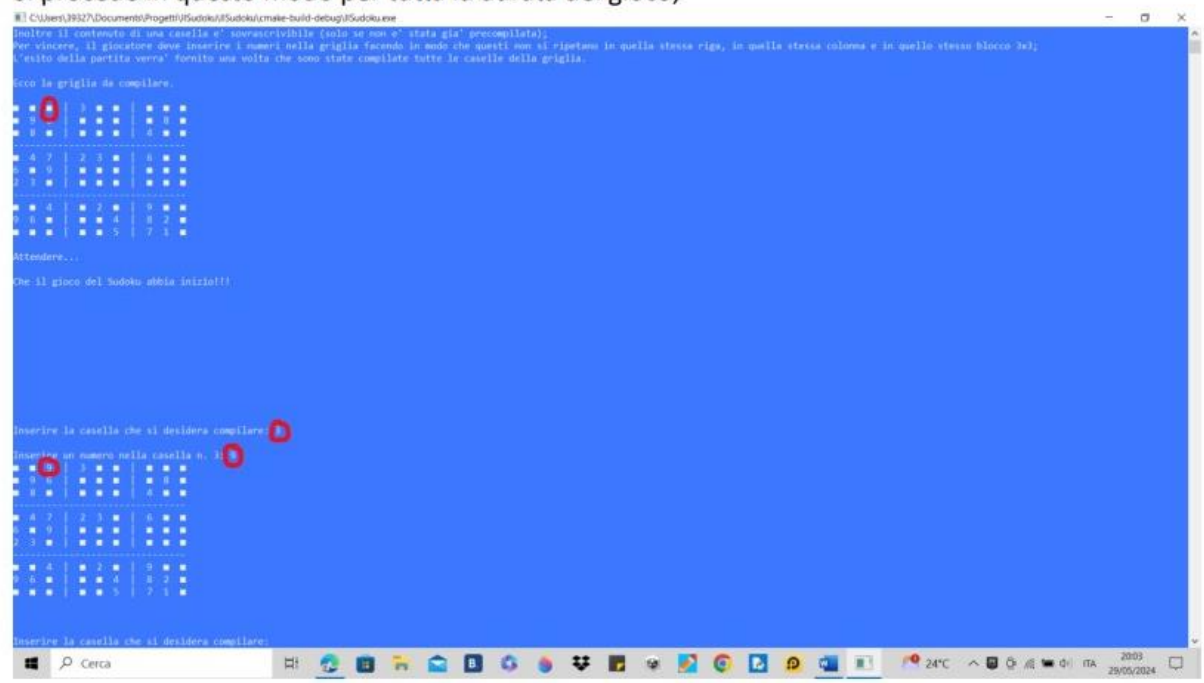

- Successivamente viene richiesta la cella che si desidera compilare;



- Qui si chiede il valore che si preferisce inserire nella cella selezionata;




- Si procede in questo modo per tutta la durata del gioco;



Ecco riportato il dizionario dati del software.

Dizionario dati:
Costanti N = 9, MAX = 9, MIN = 1;

| Nome variabile | Tipo | Utilizzo | Descrizione |
|---|---|---|---|
| blocco_1[] | Intero | Lavoro | Vettore di dimensione pari a N contenente il primo sotto-blocco di valori della griglia |
| blocco_2[] | Intero | Lavoro | Vettore di dimensione pari a N contenente il secondo sotto-blocco di valori della griglia |
| blocco_3[] | Intero | Lavoro | Vettore di dimensione pari a N contenente il terzo sotto-blocco di valori della griglia |
| blocco_4[] | Intero | Lavoro | Vettore di dimensione pari a N contenente il quarto sotto-blocco di valori della griglia |
| blocco_5[] | Intero | Lavoro | Vettore di dimensione pari a N contenente il quinto sotto-blocco di valori della griglia |
| blocco_6[] | Intero | Lavoro | Vettore di dimensione pari a N contenente il sesto sotto-blocco di valori della griglia |
| blocco_7[] | Intero | Lavoro | Vettore di dimensione pari a N contenente il settimo sotto-blocco di valori della griglia |
| blocco_8[] | Intero | Lavoro | Vettore di dimensione pari a N contenente il ottavo sotto-blocco di valori della griglia |
| blocco_9[] | Intero | Lavoro | Vettore di dimensione pari a N contenente il nono sotto-blocco di valori della griglia |
| blocco_tot_soluzione[] | Intero | Lavoro | Vettore di dimensione pari a N * N contenente tutta la griglia e quindi il contenuto dei nove sotto-blocchi caricati |
| blocco_tot_risolvere[] | Intero | Output | Vettore di dimensione pari a N * N contenente la griglia da far compilare al giocatore |
| blocco_tot_precompilato[] | Intero | Lavoro | Vettore di dimensione pari a N * N contenente la griglia di caselle gia' precompilate e non |
| casella_scelta | Intero | Input | Variabile contenente il numero della casella che il giocatore desidera compilare |
| num_inserire | Intero | Input | Variabile contenente il valore del numero da inserire in quella determinata casella |
| flag_vuoto | Intero | Lavoro | Variabile sentinella in grado di rilevare caselle vuote all'interno della griglia |
| flag_diverso | Intero | Lavoro | Variabile sentinella in grado di rilevare una disuguaglianza tra il contenuto di una casella della griglia da far compilare e il contenuto di quella stessa casella ma della griglia risolutiva |
| i_dim | Intero | Lavoro | Variabile indice del vettore blocco_tot_risolvere[] |

Si illustrino le librerie utilizzate:

- **stdio.h** (da “**St**andar**d i**nput **o**utput”): fornisce le funzioni/procedure per compiere le operazioni basiche di input/output;
- **stdlib.h** (da “**St**andar**d lib**rary”): contiene qualsiasi delle funzioni/procedure di utilità generale (es.: operazioni di allocazione di memoria);
- **time.h**: è costituita da tutte le funzioni/procedure di controllo temporale (Es.: operazioni sul tempo di elaborazione della CPU);
- **unistd.h** (da “**Uni**x **st**andar**d**”): permette l’utilizzo delle funzioni/procedure di base del sistema operativo UNIX.

```c
#include <stdio.h> // Includere la libreria standard STANDARD INPUT/OUTPUT
#include <stdlib.h> // Includere la libreria standard STANDARD LIBRARY
#include <time.h> // Includere la libreria time.h
#include <unistd.h> // Includere la libreria unistd.h
```

Si mostrino le costanti di cui si è fatto uso:

- La costante **N** si rende utile nella generazione dei 9 blocchi 3x3 costituenti la griglia;
- mentre **MAX** e **MIN** servono all’inserimento randomico di un numero compreso tra 1 e 9 nelle celle precompilate della griglia.

```c
#define N 9 // Dichiarare la costante N di valore pari a 9
#define MAX 9 // Dichiarare la costante MAX di valore pari a 9
#define MIN 1 // Dichiarare la costante MIN di valore pari a 1
```

Questi sono tutti i prototipi di funzioni/procedure create per facilitare la programmazione. Lo scopo di ognuna di queste è illustrato in basso.

```
// Dichiarazione prototipi di funzioni e/o procedure
void caricamento_blocco_1(int *);
void caricamento_blocchi_marginali(int *, int *);
void caricamento_altri_blocchi(int *, int *);
void caricamento_righe_1_blocco_tot_soluzione(int *, int *, int *, int *, int *);
void caricamento_righe_2_blocco_tot_soluzione(int *, int *, int *, int *, int *);
void caricamento_righe_3_blocco_tot_soluzione(int *, int *, int *, int *, int *);
void caricamento_blocco_tot_risolvere(int *, int *, int *);
void stampa_blocco_tot_risolvere(int *);
int verifica_vuoto(int *);
int verifica_diverso(int *, int *);
```

Si termini con la descrizione del codice sorgente del videogioco:

**int main()**:

1. Si comincia con la dichiarazione di variabili e array monodimensionali locali;

```
// Dichiarazione variabili
int i_dim, casella_scelta, num_inserire, flag_vuoto, flag_diverso;

// Dichiarazione vettori
int blocco_1[N], blocco_2[N], blocco_3[N], blocco_4[N], blocco_5[N], blocco_6[N], blocco_7[N], blocco_8[N], blocco_9[N], blocco_tot_soluzione[N * N], blocco_tot_risolvere[N * N], blocco_tot_precompilato[N * N];
```

2. Successivamente, si usufruisce della funzione srand della libreria stdlib.h per creare un “germoglio” da cui bisogna partire per la generazione di numeri pseudocasuali. A srand viene fornito come parametro il valore senza segno (unsigned) restituito dalla funzione time della libreria time.h, alla quale viene a sua volta fornito il parametro NULL. Il “germoglio” corrisponde quindi all’ora della data corrente, in modo da rendere il generatore random il più attendibile possibile;

```
// Dichiarazione di un valore Seed (Seme) da cui incominciare la generazione di numeri casuali
srand( Seed: (unsigned) time( Time: NULL));
```

3. Dopodiché, tramite l’utilizzo della funzione system della libreria stdlib.h, si cambiano i colori di background e foreground della finestra di gioco (sfondo azzurro indicato dal “9” e testo bianco indicato da “F” in questo caso);

```
// Assegnazione di un colore allo sfondo e al testo della CLI del software
system( Command: "COLOR 9F");
```

4. Qui vengono stampati il titolo e le regole del gioco;

```
// Titolo del software
printf("++++++++++++++++++++++++++++++++++++IlSudoku++++++++++++++++++++++++++++++++++++\n\n");


// Istruzioni finalizzate a garantire il corretto utilizzo del software
printf("REGOLE:\n");
printf("Lo scopo del gioco e' quello di far compilare la griglia del Sudoku al giocatore;\n");
printf("Il posizionamento della caselle va in ordine dall'alto verso il basso e da sinistra verso destra (dalla n. 1 alla n. 81);\n");
printf("Il giocatore puo' scegliere quale casella compilare;\n");
printf("Inoltre il contenuto di una casella e' sovrascrivibile (solo se non e' stata gia' precompilata);\n");
printf("Per vincere, il giocatore deve inserire i numeri nella griglia facendo in modo che questi non si ripetano in quella stessa riga, in quella stessa colonna e in quello stesso blocco 3x3;\n");
printf("L'esito della partita verra' fornito una volta che sono state compilate tutte le caselle della griglia.\n\n");
```

5. Si prosegue con il caricamento del primo sotto-blocco della griglia soluzione mediante la procedura caricamento_blocco_1;

```
// Fase di caricamento del vettore blocco_1[]
caricamento_blocco_1( pblocco_1: blocco_1);
```

6. A seguire, si va avanti per tutti gli altri blocchi della griglia soluzione con le procedure indicate sotto. La procedura caricamento_blocchi_marginali si occupa di tutti i sotto-blocchi sul margine sinistro della griglia ad eccezione del primo (cioè il quarto ed il settimo), perché (come accade anche per il primo) meritano un trattamento particolare. Per il caricamento di tutti gli altri blocchi si utilizza la procedura caricamento_altri_blocchi.

```
// Fase di caricamento del vettore blocco_2[]
caricamento_altri_blocchi( pblocco_caricare: blocco_2,  pblocco_usare_2: blocco_1);

// Fase di caricamento del vettore blocco_3[]
caricamento_altri_blocchi( pblocco_caricare: blocco_3,  pblocco_usare_2: blocco_2);

// Fase di caricamento del vettore blocco_4[]
caricamento_blocchi_marginali( pblocco_caricare: blocco_4,  pblocco_usare_1: blocco_1);

// Fase di caricamento del vettore blocco_5[]
caricamento_altri_blocchi( pblocco_caricare: blocco_5,  pblocco_usare_2: blocco_4);

// Fase di caricamento del vettore blocco_6[]
caricamento_altri_blocchi( pblocco_caricare: blocco_6,  pblocco_usare_2: blocco_5);

// Fase di caricamento del vettore blocco_7[]
caricamento_blocchi_marginali( pblocco_caricare: blocco_7,  pblocco_usare_1: blocco_4);

// Fase di caricamento del vettore blocco_8[]
caricamento_altri_blocchi( pblocco_caricare: blocco_8,  pblocco_usare_2: blocco_7);

// Fase di caricamento del vettore blocco_9[]
caricamento_altri_blocchi( pblocco_caricare: blocco_9,  pblocco_usare_2: blocco_8);
```

7. Dopo il caricamento dei nove sotto-blocchi della griglia (la griglia è il blocco totale), si procede con il caricamento di altrettanti nove vettori contenenti questi vettori appena caricati ma disposti riga per riga nella griglia. Per compiere tutto ciò, si utilizzano le procedure mostrate nell'immagine in basso;

```
i_dim = 0;

// Fase di caricamento della prima riga della griglia risolutiva del gioco
caricamento_righe_1_blocco_tot_soluzione( pi_dim: &i_dim, pblocco_tot_soluzione: blocco_tot_soluzione, pblocco_usare_1_1: blocco_1, pblocco_usare_1_2: blocco_2, pblocco_usare_1_3: blocco_3);

// Fase di caricamento della seconda riga della griglia risolutiva del gioco
caricamento_righe_2_blocco_tot_soluzione( pi_dim: &i_dim, pblocco_tot_soluzione: blocco_tot_soluzione, pblocco_usare_2_1: blocco_1, pblocco_usare_2_2: blocco_2, pblocco_usare_2_3: blocco_3);

// Fase di caricamento della terza riga della griglia risolutiva del gioco
caricamento_righe_3_blocco_tot_soluzione( pi_dim: &i_dim, pblocco_tot_soluzione: blocco_tot_soluzione, pblocco_usare_3_1: blocco_1, pblocco_usare_3_2: blocco_2, pblocco_usare_3_3: blocco_3);

// Fase di caricamento della quarta riga della griglia risolutiva del gioco
caricamento_righe_1_blocco_tot_soluzione( pi_dim: &i_dim, pblocco_tot_soluzione: blocco_tot_soluzione, pblocco_usare_1_1: blocco_4, pblocco_usare_1_2: blocco_5, pblocco_usare_1_3: blocco_6);

// Fase di caricamento della quinta riga della griglia risolutiva del gioco
caricamento_righe_2_blocco_tot_soluzione( pi_dim: &i_dim, pblocco_tot_soluzione: blocco_tot_soluzione, pblocco_usare_2_1: blocco_4, pblocco_usare_2_2: blocco_5, pblocco_usare_2_3: blocco_6);

// Fase di caricamento della sesta riga della griglia risolutiva del gioco
caricamento_righe_3_blocco_tot_soluzione( pi_dim: &i_dim, pblocco_tot_soluzione: blocco_tot_soluzione, pblocco_usare_3_1: blocco_4, pblocco_usare_3_2: blocco_5, pblocco_usare_3_3: blocco_6);

// Fase di caricamento della settima riga della griglia risolutiva del gioco
caricamento_righe_1_blocco_tot_soluzione( pi_dim: &i_dim, pblocco_tot_soluzione: blocco_tot_soluzione, pblocco_usare_1_1: blocco_7, pblocco_usare_1_2: blocco_8, pblocco_usare_1_3: blocco_9);

// Fase di caricamento dell'ottava riga della griglia risolutiva del gioco
caricamento_righe_2_blocco_tot_soluzione( pi_dim: &i_dim, pblocco_tot_soluzione: blocco_tot_soluzione, pblocco_usare_2_1: blocco_7, pblocco_usare_2_2: blocco_8, pblocco_usare_2_3: blocco_9);

// Fase di caricamento della nona riga della griglia risolutiva del gioco
caricamento_righe_3_blocco_tot_soluzione( pi_dim: &i_dim, pblocco_tot_soluzione: blocco_tot_soluzione, pblocco_usare_3_1: blocco_7, pblocco_usare_3_2: blocco_8, pblocco_usare_3_3: blocco_9);
```

8. Una volta pronto il blocco soluzione, si continua con la creazione della griglia sulla quale il giocatore dovrà lavorare. Tale griglia viene costruita dalla procedura caricamento_blocco_tot_risolvere;

```
// Fase di caricamento della griglia da far risolvere al giocatore, contenente talvolta anche delle caselle vuote, rappresentate dal valore numerico 254
caricamento_blocco_tot_risolvere( pblocco_tot_soluzione: blocco_tot_soluzione, pblocco_tot_risolvere: blocco_tot_risolvere, pblocco_tot_precompilato: blocco_tot_precompilato);
```

9. Poi viene fatta stampare a video quest'ultima griglia mediante la procedura stampa_blocco_tot_risolvere;

```
// Messaggio di introduzione ai valori contenuti nella griglia da far compilare
printf("Ecco la griglia da compilare.\n\n");

// Fase di visualizzazione della griglia da far compilare
stampa_blocco_tot_risolvere( pblocco_tot_risolvere: blocco_tot_risolvere);
```

10. Si inserisce anche una piccola pausa con la funzione sleep della libreria unistd.h per permettere al giocatore di ripassare le regole del gioco prima di iniziare effettivamente a giocarci;

```
// Messaggio finalizzato a rendere chiara l'attesa a cui il giocatore deve sottostare
printf("\n\nAttendere...");

// Pausa di 7 secondi
sleep(7);
```

11. Dopo lo svolgimento di queste operazioni di introduzione, si comincia davvero a giocare;

```
// Messaggio finalizzato a informare il giocatore che il gioco e' iniziato
printf("\n\n\aChe il gioco del Sudoku abbia inizio!!!");


printf("\n\n\n\n\n\n\n\n\n");
```

12. In questa regione di codice, il software richiede l'inserimento di un valore (da 1 a 81) che indichi la casella della griglia che si desidera compilare. La richiesta viene effettuata per tutte le volte che il giocatore inserisce un numero che fuoriesce dall'intervallo corretto (cioè 1 – 81);

```
// Inserire il numero della casella che si desidera compilare fin quando questo non e' compreso nell'intervallo [+1, +81] compresi e rispecchia una casella che e' stata gia' precompilata
do {
    printf("Inserire la casella che si desidera compilare: ");
    scanf( format: "%d", &casella_scelta);

    if (casella_scelta < 1 || casella_scelta > 81) {
        printf("\aAttenzione! Inserire solo numeri compresi tra 1 e 81 compresi.\n\n"); // Lasciare un messaggio di errore nel caso in cui la condizione di cui sopra non venisse rispettata
    } else {
        if (blocco_tot_precompilato[casella_scelta - 1] == -1) {
            printf("\aAttenzione! Compilare solo le caselle che non sono state precompilate.\n\n"); // Lasciare un diverso messaggio di errore nel caso in cui fosse vera l'altra condizione
        }
    }
} while (casella_scelta < 1 || casella_scelta > 81 || blocco_tot_precompilato[casella_scelta - 1] == -1);
```

13. Qui viene poi richiesto il numero (da 1 a 9) che si vuole inserire nella cella selezionata. Proprio come prima, la richiesta si ripete ciclicamente per tutte le volte che l'utente inserisce un numero non presente nel giusto intervallo (cioè 1 – 9);

```
// Inserire un numero da impostare in quella casella fin quando questo non e' contenuto nell'intervallo [+1, +9] compresi
do {
    printf("\nInserire un numero nella casella n. %d: ", casella_scelta);
    scanf( format: "%d", &num_inserire);

    if (num_inserire < 1 || num_inserire > 9) {
        printf("\aAttenzione! Inserire solo numeri compresi tra 1 e 9 compresi.\n\n"); // Lasciare un messaggio di errore nel caso in cui la condizione di cui sopra non venisse rispettata
    }
} while (num_inserire < 1 || num_inserire > 9);
```

14. Il numero scelto viene successivamente inserito nell'apposita casella;

```
// Inserire il valore di num_inserire nella cella del vettore di indice pari a casella_scelta - 1 (questo perche' casella_scelta indica una posizione e non un indice)
blocco_tot_risolvere[casella_scelta - 1] = num_inserire;
```

15. Si stampa a video la griglia aggiornata con la procedura stampa_blocco_tot_risolvere;

```
// Fase di visualizzazione della griglia aggiornata all'ultimo valore inserito dall'utente in quella determinata casella
stampa_blocco_tot_risolvere( pblocco_tot_risolvere: blocco_tot_risolvere);
```

16. Dopo aver verificato che la griglia non sia ancora stata completata (con la funzione verifica_vuoto), si continua ripetendo questi ultimi quattro passaggi (si riparte quindi dal punto 12);

```
    // 0: false; 1: true.
    flag_vuoto = verifica_vuoto( pblocco_tot_risolvere: blocco_tot_risolvere);
} while (flag_vuoto);
```

17. Più tardi, dopo aver completato la griglia, il gioco verifica (mediante l'utilizzo della funzione verifica_diverso) che i valori inseriti dall'utente nella sua griglia siano TUTTI uguali a quelli presenti nella griglia soluzione;

```
// 0: false; 1: true
flag_diverso = verifica_diverso( pblocco_tot_soluzione: blocco_tot_soluzione,  pblocco_tot_risolvere: blocco_tot_risolvere);


printf("\n\n\n\n");
```

18. Quindi se le due griglie sono perfettamente identiche, il giocatore ha vinto. Ha perso in caso contrario;

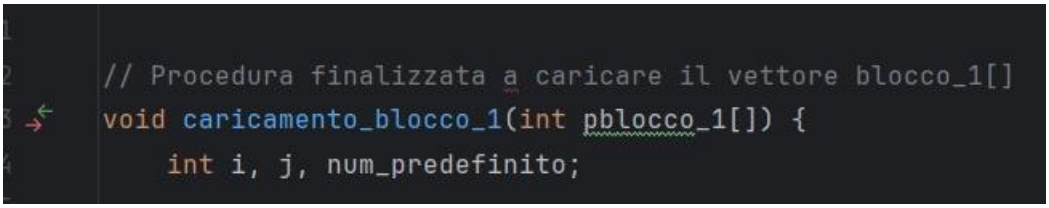
```
// Se flag_diverso = 1 (quindi se nella griglia da risolvere e' stato trovato un elemento diverso da quello contenuto nella griglia risolutiva)
if (flag_diverso) {
    printf("Esito: spiacente, ma hai perso."); // Comunicare al giocatore che ha perso

// Altrimenti
} else {
    printf("\aEsito: complimenti! Hai vinto!"); // Comunicare al giocatore che ha vinto
}
```

19. Infine il gioco va in pausa (a causa del valore "PAUSE" fornito come parametro alla funzione system) e si chiude con la pressione di un tasto.

```
    // Andare a capo per due volte
    printf("\n\n");

    // Mettere in pausa il software alla fine della sua esecuzione
    system( Command: "PAUSE");

    // Chiudere la funzione main
    return 0;
}
```

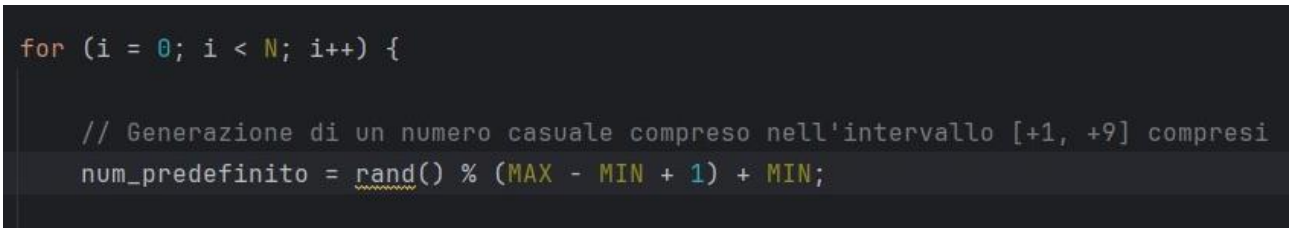
**caricamento_blocco_1(int pblocco_1[])**:

1. Si comincia con la dichiarazione di variabili locali. Il parametro intero pblocco_1[] fa riferimento al primo sotto-blocco della griglia;



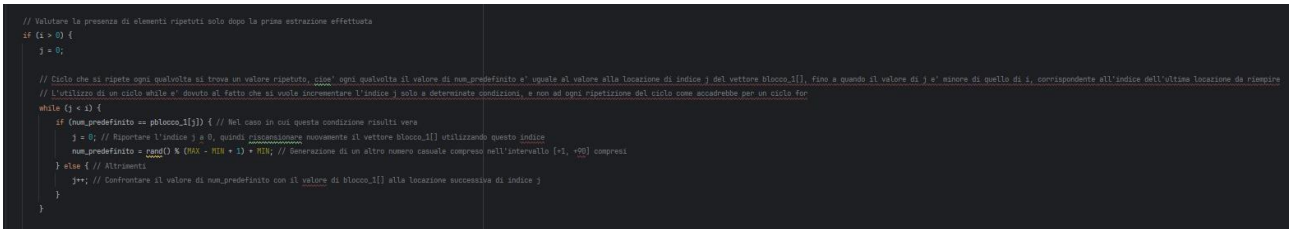
```
// Procedura finalizzata a caricare il vettore blocco_1[]
void caricamento_blocco_1(int pblocco_1[]) {
    int i, j, num_predefinito;
```

2. Si prosegue con la generazione di un numero pseudocasuale compreso tra 1 e 9;



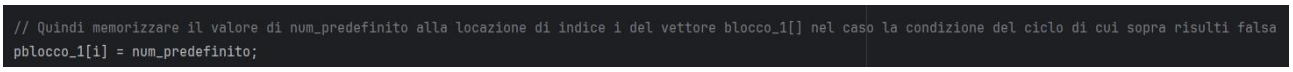
```
for (i = 0; i < N; i++) {

    // Generazione di un numero casuale compreso nell'intervallo [+1, +9] compresi
    num_predefinito = rand() % (MAX - MIN + 1) + MIN;
```

3. Prima di inserire un altro numero all'interno del sotto-blocco n. 1 (ah! Il sotto-blocco n. 1 è quello in alto a sinistra nella griglia di gioco, da come si può dedurre dalla guida), è giusto verificare che questo nuovo numero non sia già presente all'interno del sotto-blocco. Questa verifica si svolge in modo ciclico per ogni nuovo numero generato, e continua fin quando non si riempie tutto il sotto-blocco;



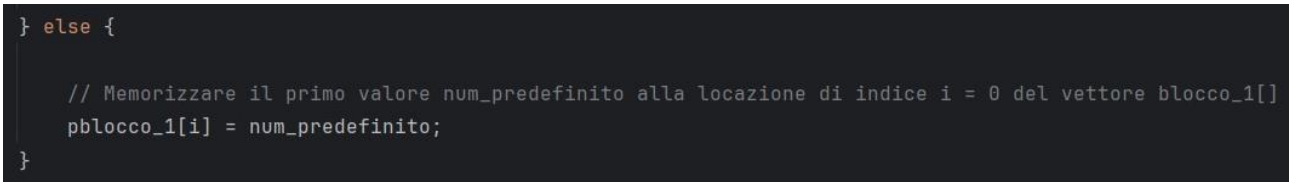
```
// Valutare la presenza di elementi ripetuti solo dopo la prima estrazione effettuata
if (i > 0) {
    j = 0;

    // Ciclo che si ripete ogni qualvolta si trova un valore ripetuto, cioe' ogni qualvolta il valore di num_predefinito e' uguale al valore alla locazione di indice j del vettore blocco_1[], fino a quando il valore di j e' minore di quello di i, corrispondente all'indice dell'ultima locazione da riempire
    // L'utilizzo di un ciclo while e' dovuto al fatto che si vuole incrementare l'indice j solo a determinate condizioni, e non ad ogni ripetizione del ciclo come accadrebbe per un ciclo for
    while (j < i) {
        if (num_predefinito == pblocco_1[j]) { // Nel caso in cui questa condizione risulti vera
            j = 0; // Riportare l'indice j a 0, quindi riscansionare nuovamente il vettore blocco_1[] utilizzando questo indice
            num_predefinito = rand() % (MAX - MIN + 1) + MIN; // Generazione di un altro numero casuale compreso nell'intervallo [+1, +99] compresi
        } else { // Altrimenti
            j++; // Confrontare il valore di num_predefinito con il valore di blocco_1[] alla locazione successiva di indice j
        }
    }
```

4. Una volta uscito dal ciclo di verifica di cui al punto n. 3, il numero viene memorizzato in quella cella di indice i;



```
// Quindi memorizzare il valore di num_predefinito alla locazione di indice i del vettore blocco_1[] nel caso la condizione del ciclo di cui sopra risulti falsa
pblocco_1[i] = num_predefinito;
```

5. Facendo ancora riferimento al punto n. 3: quel controllo ciclico non viene svolto se (ovviamente) il numero generato a random è il primo (deve occupare la prima cella della griglia) a dover ospitare il sotto-blocco.



```
} else {

    // Memorizzare il primo valore num_predefinito alla locazione di indice i = 0 del vettore blocco_1[]
    pblocco_1[i] = num_predefinito;
}
```

**caricamento_blocchi_marginali(int pblocco_caricare[], int pblocco_usare_1[])**:

1. Si comincia con la dichiarazione di variabili locali. Il parametro intero pblocco_caricare[] si riferisce al sotto-blocco che deve essere caricato facendo basandosi su pblocco_usare_1[];

```
// Procedura finalizzata a caricare tutti i blocchi che si trovano sul margine a sinistra nella griglia
void caricamento_blocchi_marginali(int pblocco_caricare[], int pblocco_usare_1[]) {
    int i;
```

2. In quest'area di codice viene effettuato il caricamento di pblocco_caricare[]. Si preferisce non aggiungere ulteriori dettagli a riguardo per evitare di fornire troppi indizi sulla soluzione del gioco.

```
    for (i = 0; i < N; i++) {
        if (i == 0 || i == 3 || i == 6) {
            pblocco_caricare[i] = pblocco_usare_1[i + 2];
        } else {
            pblocco_caricare[i] = pblocco_usare_1[i - 1];
        }
    }
}
```

**caricamento_altri_blocchi(int pblocco_caricare[], int pblocco_usare_2[])**:

1. Si comincia con la dichiarazione di variabili locali. Il parametro intero pblocco_caricare[] si riferisce al sotto-blocco che deve essere caricato facendo basandosi su pblocco_usare_2[];

```
// Procedura finalizzata a caricare tutti gli altri vettori blocco_n[]
void caricamento_altri_blocchi(int pblocco_caricare[], int pblocco_usare_2[]) {
    int i;
```

2. In quest'area di codice viene effettuato il caricamento di pblocco_caricare[]. Si ribadisce quanto detto in precedenza sui dettagli in merito a quanto presente nell'area di codice in basso.

```
    for (i = 0; i < N; i++) {
        if (i > 2) {
            pblocco_caricare[i] = pblocco_usare_2[i - 3];
        } else {
            pblocco_caricare[i] = pblocco_usare_2[i + 6];
        }
    }
}
```

**caricamento_righe_1_blocco_tot_soluzione(int *pi_dim, int pblocco_tot_soluzione[], int pblocco_usare_1_1[], int pblocco_usare_1_2[], pblocco_usare_1_3[])**:

1. Si inizia con la dichiarazione di variabili locali. il parametro intero passato per referenza pi_dim (inizializzato a 0 con il nome di i_dim nel codice della funzione main()) serve a fare da indice all'interno di pblocco_tot_soluzione[] che, per essere caricato, ha bisogno dei valori presenti nella prima riga dei tre blocchi pblocco_usare_1_1[], pblocco_usare_1_2[] e pblocco_usare_1_3[];

```c
// Procedura finalizzata a caricare la prima riga di ogni trio di blocchi
void caricamento_righe_1_blocco_tot_soluzione(int *pi_dim, int pblocco_tot_soluzione[], int pblocco_usare_1_1[], int pblocco_usare_1_2[], int pblocco_usare_1_3[]) {
    int i;
```

2. Qui viene effettuato effettivamente il caricamento della prima, della quarta e della settima riga della griglia. Si ribadisce quanto detto in precedenza sui dettagli in merito a quanto presente nell'area di codice in basso.

```c
    for (i = 0; i < N; i++) {
        if (i > 5) {
            pblocco_tot_soluzione[*pi_dim] = pblocco_usare_1_3[i - 6];
            *pi_dim = *pi_dim + 1;
        } else if (i > 2) {
            pblocco_tot_soluzione[*pi_dim] = pblocco_usare_1_2[i - 3];
            *pi_dim = *pi_dim + 1;
        } else {
            pblocco_tot_soluzione[*pi_dim] = pblocco_usare_1_1[i];
            *pi_dim = *pi_dim + 1;
        }
    }
}
```

**caricamento_righe_2_blocco_tot_soluzione(int *pi_dim, int pblocco_tot_soluzione[], int pblocco_usare_2_1[], int pblocco_usare_2_2[], pblocco_usare_2_3[])**:

1. Si inizia con la dichiarazione di variabili locali. il parametro intero passato per referenza pi_dim (inizializzato a 0 con il nome di i_dim nel codice della funzione main()) serve a fare da indice all'interno di pblocco_tot_soluzione[] che, per essere caricato, ha bisogno dei valori presenti nella prima riga dei tre blocchi pblocco_usare_2_1[], pblocco_usare_2_2[] e pblocco_usare_2_3[];

```c
// Procedura finalizzata a caricare la seconda riga di ogni trio di blocchi
void caricamento_righe_2_blocco_tot_soluzione(int *pi_dim, int pblocco_tot_soluzione[], int pblocco_usare_2_1[], int pblocco_usare_2_2[], int pblocco_usare_2_3[]) {
    int i;
```

2. Qui viene effettuato effettivamente il caricamento della seconda, della quinta e dell'ottava riga della griglia. Si ribadisce quanto detto in precedenza sui dettagli in merito a quanto presente nell'area di codice in basso.

```
    for (i = 3; i < N + 3; i++) {
        if (i > 8) {
            pblocco_tot_soluzione[*pi_dim] = pblocco_usare_2_3[i - 6];
            *pi_dim = *pi_dim + 1;
        } else if (i > 5) {
            pblocco_tot_soluzione[*pi_dim] = pblocco_usare_2_2[i - 3];
            *pi_dim = *pi_dim + 1;
        } else {
            pblocco_tot_soluzione[*pi_dim] = pblocco_usare_2_1[i];
            *pi_dim = *pi_dim + 1;
        }
    }
}
```

**caricamento_righe_3_blocco_tot_soluzione(int *pi_dim, int pblocco_tot_soluzione[], int pblocco_usare_3_1[], int pblocco_usare_3_2[], pblocco_usare_3_3[])**:

1. Si inizia con la dichiarazione di variabili locali. il parametro intero passato per referenza pi_dim (inizializzato a 0 con il nome di i_dim nel codice della funzione main()) serve a fare da indice all'interno di pblocco_tot_soluzione[] che, per essere caricato, ha bisogno dei valori presenti nella prima riga dei tre blocchi pblocco_usare_3_1[], pblocco_usare_3_2[] e pblocco_usare_3_3[];

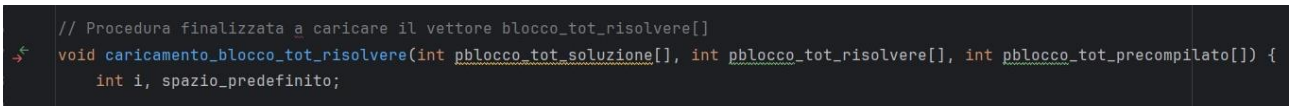
```
// Procedura finalizzata a caricare la terza riga di ogni trio di blocchi
void caricamento_righe_3_blocco_tot_soluzione(int *pi_dim, int pblocco_tot_soluzione[], int pblocco_usare_3_1[], int pblocco_usare_3_2[], int pblocco_usare_3_3[]) {
    int i;
```

2. Qui viene effettuato effettivamente il caricamento della terza, della sesta e della nona riga della griglia. Si ribadisce quanto detto in precedenza sui dettagli in merito a quanto presente nell'area di codice in basso.

```
    for (i = 6; i < N + 6; i++) {
        if (i > 11) {
            pblocco_tot_soluzione[*pi_dim] = pblocco_usare_3_3[i - 6];
            *pi_dim = *pi_dim + 1;
        } else if (i > 8) {
            pblocco_tot_soluzione[*pi_dim] = pblocco_usare_3_2[i - 3];
            *pi_dim = *pi_dim + 1;
        } else {
            pblocco_tot_soluzione[*pi_dim] = pblocco_usare_3_1[i];
            *pi_dim = *pi_dim + 1;
        }
    }
}
```

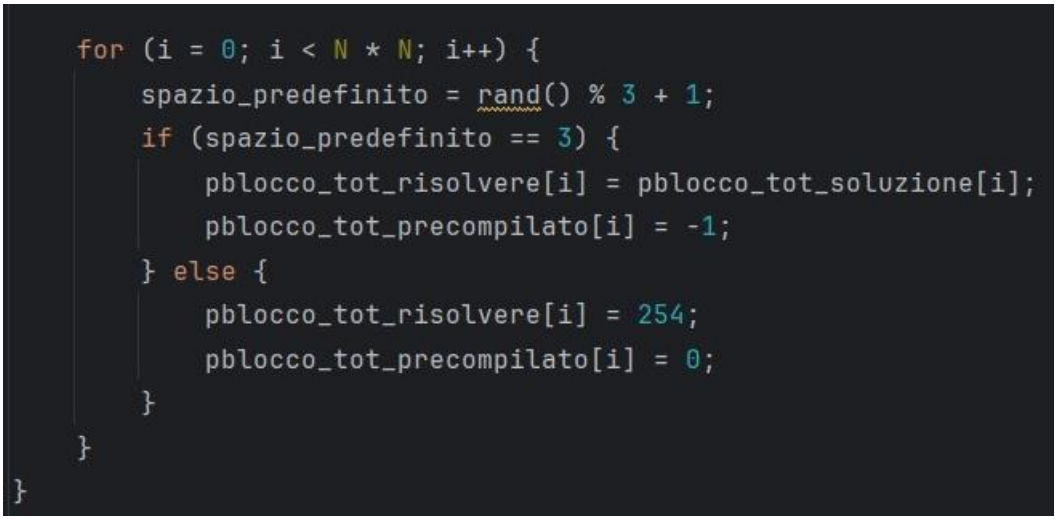
**caricamento_blocco_tot_risolvere(int pblocco_tot_soluzione[], int pblocco_tot_risolvere[], int pblocco_tot_precompilato[]):**

1. Si inizia con la dichiarazione di variabili locali. Prendendo come riferimento pblocco_tot_soluzione[] e pblocco_tot_precompilato[], si permette il caricamento della griglia da risolvere. La griglia pblocco_tot_precompilato[] contiene due soli valori: 0 (la cella deve essere compilata dall'utente e il suo contenuto è modificabile); -1 (la cella è stata precompilata dal software, quindi non è modificabile);



```
// Procedura finalizzata a caricare il vettore blocco_tot_risolvere[]
void caricamento_blocco_tot_risolvere(int pblocco_tot_soluzione[], int pblocco_tot_risolvere[], int pblocco_tot_precompilato[]) {
    int i, spazio_predefinito;
```

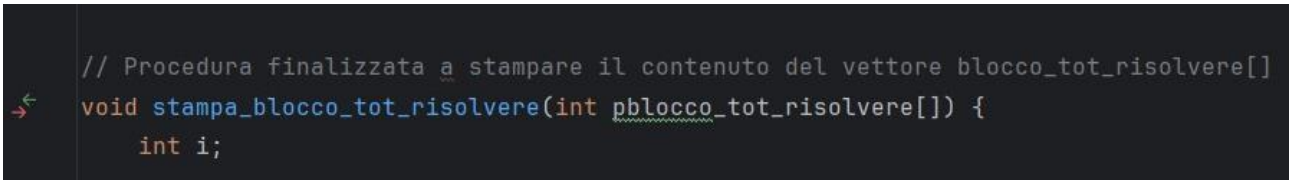
2. Qui viene effettuato effettivamente il caricamento della griglia pblocco_tot_risolvere[]. La variabile spazio_predefinito serve a capire quale cella deve rimanere vuota e quale deve contenere un valore predefinito. Il numero 254 viene interpretato come "■" dal videogioco, associandolo di conseguenza alla cella vuota. La probabilità di incontrare nella griglia una cella priva di valore è di 1/3.



```
    for (i = 0; i < N * N; i++) {
        spazio_predefinito = rand() % 3 + 1;
        if (spazio_predefinito == 3) {
            pblocco_tot_risolvere[i] = pblocco_tot_soluzione[i];
            pblocco_tot_precompilato[i] = -1;
        } else {
            pblocco_tot_risolvere[i] = 254;
            pblocco_tot_precompilato[i] = 0;
        }
    }
}
```

**stampa_blocco_tot_risolvere(int pblocco_tot_risolvere[])**:

1. Si inizia con la dichiarazione di variabili locali. Si effettui quindi la stampa a video di pblocco_tot_risolvere[], la griglia che il giocatore ha il compito di completare.



```
// Procedura finalizzata a stampare il contenuto del vettore blocco_tot_risolvere[]
void stampa_blocco_tot_risolvere(int pblocco_tot_risolvere[]) {
    int i;
```

2. In quest'area di codice, il programma inserisce le barre verticali divisorie tra un sotto-blocco e un altro adiacente;

```
for (i = 0; i < N * N; i++) {

    // Scrivere dei caratteri "|" nella griglia che riescano a dividerla in tre colonne
    if (i % 3 == 0 && i != 0 && i != 9 && i != 18 && i != 27 && i != 36 && i != 45 && i != 54 && i != 63 && i != 72) {
        printf("|  ");
    }
```

3. Successivamente, si inserisce una barra orizzontale utile a dividere un gruppo di tre sotto-blocchi da un altro sopra/sotto.

```
// Scrivere anche dei trattini in modo da simulare delle linee orizzontali che dividano, insieme ai caratteri "|", la griglia nei suoi nove sotto-blocchi
if (i % 9 == 0 && i != 0) {
    printf("\n");
    if (i % 27 == 0) {
        printf("-------------------------------\n");
    }
}
```

4. Infine, si decide se stampare "■" o un numero preimpostato in ognuna delle celle della griglia da completare.

```
        // Condizione utile a capire se bisogna scrivere un valore numerico o una casella vuota
        if (pblocco_tot_risolvere[i] == 254) {
            printf("%c  ", pblocco_tot_risolvere[i]);
        } else {
            printf("%d  ", pblocco_tot_risolvere[i]);
        }
    }
}
```

**verifica_vuoto(int pblocco_tot_risolvere[])**:

1. Si inizia con la dichiarazione di variabili locali. La variabile sentinella F serve a contenere la risposta alla verifica che si effettua successivamente sulla griglia pblocco_tot_risolvere[];

```
// Funzione finalizzata a verificare che nel vettore blocco_tot_risolvere[] ci sia il valore 254 (corrispondente alla casella vuota nel gioco)
int verifica_vuoto(int pblocco_tot_risolvere[]) {
    int i, F = 0;
```

2. Qui si verifica per davvero la presenza di eventuali vuoti nella griglia;

```
for (i = 0; i < N * N && !F; i++) {
    if (pblocco_tot_risolvere[i] == 254) {
        F = 1;
    }
}
```

3. Se ci sono ancora vuoti, viene restituito 1 e quindi il gioco procede; altrimenti viene restituito 0 e il gioco finisce;

```
    if (F) {
        return 1;
    } else {
        return 0;
    }
}
```

**verifica_diverso(int pblocco_tot_soluzione[], int pblocco_tot_risolvere[])**:

1. Si inizia con la dichiarazione di variabili locali. La variabile sentinella F serve a contenere la risposta alla verifica che si effettua successivamente mettendo a confronto la griglia soluzione e la griglia da completare;

```
// Funzione finalizzata a verificare che la griglia compilata dal giocatore coincida con quella della soluzione
int verifica_diverso(int pblocco_tot_soluzione[], int pblocco_tot_risolvere[]) {
    int i, F = 0;
```

2. Qui viene effettivamente verificata la presenza di eventuali differenze tra le due griglie;

```
// Portare a 1 il valore di flag_diverso nel caso in cui venisse trovato nella griglia da risolvere un elemento diverso da quello presente nella griglia risolutiva
for (i = 0; i < N * N && !F; i++) {
    if (pblocco_tot_soluzione[i] != pblocco_tot_risolvere[i]) {
        F = 1;
    }
}
```

3. Si restituisca quindi il valore ricevuto da F. Se tale valore è uguale a 1, il giocatore ha perso; ha vinto in caso contrario.

```
    return F;
}
```