Nome e Cognome: Gabriel Pedranzini Matricola: 32292A Anno: 2023 - 2024

BubbleBlast

Nel programma realizzato è presente lo sviluppo del gioco Bubble Blast in C, usando come parte grafica la console.

All'avvio del programma verrà chiesto all'utente di inserire il proprio nome. Una volta inserito verrà valorizzata la struttura Giocatore leggendo dal file *giocatori.txt* il record corrispondente al nome inserito.

Tale record è così composto:
nome;partite_giocate;partite_vinte;miglior_sequenza;punteggio;data_ultima_partita;matrice_partita;max_mosse;mosse_eff;diff_partita;

I campi: matrice_partita, max_mosse, mosse_eff, diff_partita contengono le informazioni riguardanti ad una partita ancora in corso e non conclusa dall'utente.

Nel caso non fosse presente un record corrispondente al nome passato la struttura Giocatore verrà valorizzata a valori di default.

```c
// Definizione struttura Giocatore
typedef struct {
    char *nome;                     //Contiene il nome del giocatore
    int partite_giocate;            //Contiene il totale di partite giocate
    int partite_vinte;              //Contiene il totale di partite vinte
    int miglior_sequenza;           //Contiene la miglior seguenza di partite
vinte di fila
    int punteggio;                  //Contiene il miglior punteggio
    char *data_ultima_partita;      //Contiene la data dell'ultima partita
    int matrice_partita[ROW][COL];  //Contiene la matrice della partita in corso
    int max_mosse;                  //Contiene il numero delle mosse massime
della partita in corso
    int mosse_eff;                  //Contiene il numero delle mosse effettuate
della partita in corso
    int diff_partita;               //Contiene la difficoltà della partita in
corso
} Giocatore;
```

Valorizzata la struttura verrà mostrato all'utente il menù di selezione in cui potrà svolgere diverse operazioni.

```
       B U B B L E     B L A S T

*********************************************************************************************
 Benvenuto nel MENU' Gabriel (seleziona un'opzione)

  G gioca       S statistiche    D difficolta'   N giocatore     R regole        E esci

*********************************************************************************************
```

Tutta la gestione delle operazioni scelte e le varie chiamate di funzione sono gestite nel *MAIN*. All'avvio come detto in precedenza viene chiesto il nome utente in input per andare a valorizzare la struct Giocatore. Dopodiché si è entrerà in un ciclo che gestirà tutti gli input riguardanti l'opzione scelta dall'utente. Da tale ciclo si uscirà solo nel caso in cui l'utente inserisca l'opzione "E" e terminerà quindi l'esecuzione del programma. Mentre per tutti gli altri casi il controllo dell'opzione viene fatto all'interno del ciclo attraverso un IF, tranne per l'opzione "G" per cui il controllo sarà effettuato attraverso un while per far sì che l'utente possa continuare a giocare senza tornare per forza nel menu principale.

Modalità di gioco

Inserendo il carattere “G” verrà selezionata la modalità di gioco che è composta da una matrice di dimensioni 5 * 6 con al suo interno delle bolle in stati da (0 a 3).

```
       B U B B L E     B L A S T

*********************************************************************************************

 +-----+-------+-------+-------+-------+-------+-------+
 | RxC |   1   |   2   |   3   |   4   |   5   |   6   |
 +-----+-------+-------+-------+-------+-------+-------+
 |  1  |       (3)     (1)     (2)     (2)     (3)  |
 |  2  |  (2)     (2)                  (2)          |
 |  3  |  (2)     (2)           (2)              (3)  |
 |  4  |  (3)     (3)     (3)     (1)     (3)     (1)  |
 |  5  |  (3)     (2)     (1)     (3)     (3)          |
 +-----+-------+-------+-------+-------+-------+-------+

*********************************************************************************************
 Mosse minime: 4        Mosse effettuate: 0     Difficolta': 3   E per tornare al menu'
```

La matrice può contenere bolle in diversi stati:

- Stato 0: la bolla è esplosa/non è presente
- Stato 1: la bolla è in procinto di esplodere, colpendola andrà nello stato 0 e propagherà l’esplosione
- Stato 2: la bolla è gonfia a metà, colpendola andrà nello stato in procinto di esplodere (1)
- Stato 3: la bolla è sgonfia, colpendola si gonfierà a metà (2)

All’avvio della modalità di gioco viene verificato se è già presente una partita in corso e si richiederà eventualmente al player se vuole riprendere la partita in corso.

```
       B U B B L E     B L A S T

 E' stata trovata una partita in data 23/12/2023 16:25:46 con il nome: Gabriel
 Vuoi riprendere la vecchia partita? (Y o N)
```

In caso positivo verrà caricata la vecchia partita in corso, altrimenti ne verrà creata una nuova da capo.

Inserimento bolle

Le bolle vengono inserite in stati randomici grazie alla funzione:

```
// Inserimento randomico nella matrice di gioco delle bolle da 0 a 3
void inserisciBolle(int matrice[ROW][COL]) {
    // Verifica che la matrice non contenga bolle a 0
    bool flag_vuota = true;
    srand(time(NULL));
    while (flag_vuota) {
        for(int x=0;x<ROW;x++) {
            for(int y=0;y<COL;y++) {
                matrice[x][y] = rand()%4;
                if(matrice[x][y] != 0) flag_vuota = false;
            }
        }
    }
}
```

Tale funzione inserisce in ogni cella della matrice valori da 0 a 3 in modo random, nel caso in cui tutti i valori inseriti siano a 0 e quindi la matrice sia vuota si riparte da capo a generare la matrice.

Calcolo mosse minime

Lo scopo del gioco è quello di fare esplodere tutte le bolle nel minor numero di mosse possibili.

Il numero minimo di mosse possibili viene calcolato tramite la seguente funzione ricorsiva:

```
// Calcolo delle mosse minime della matrice di gioco
void calcolaNumeroMosse(int matrice[ROW][COL], int mosse){
    // Salva la mossa passata come parametro
    int save_mosse = mosse;
    // Cicla gli stati delle bolle da cercare (si parte dalle bolle in procinto
di esplodere)
    for(int stato = 1; stato < 4; stato++){
        // Cicla tutta la matrice passata come parametro
        for(int x = 0; x < ROW; x++){
            for (int y = 0; y < COL; y++) {
                // Verifica se le mosse salvate + stato (che indica numero
minimo di altre mosse che sicuramente si andranno a fare)
                // è minore delle mosse massime
                if((save_mosse + stato) >= maxMosse) return;
                // Verifica che la cella della matrice contenga lo stato cercato
                if(matrice[x][y] == stato) {
                    // Si imposta mosse = mosse passate per parametro
                    mosse = save_mosse;
                    // Si fa copia della matrice da modificare
                    int matrice_2[ROW][COL];
                    copiamatrice(*matrice_2,*matrice);
                    // Si decrementa lo stato della cella e nel caso si fa
esplodere
                    matrice_2[x][y]--;
                    if(matrice_2[x][y] == 0) esplosione(matrice_2, x, y);

                    // Si incrementa il numero di mosse
                    mosse++;

                    // Si controlla se la matrice è tutta a 0 e si salva nella
variabile globale maxMosse le mosse
                    if(controllaVincita(matrice_2)) {
                        maxMosse = mosse;
                        return;
                    }

                    // Copia la matrice modificata
                    int matrice_copia_2[ROW][COL];
                    copiamatrice(*matrice_copia_2,*matrice_2);

                    // Chiama ricorsivamente la funzione passando come parametro
la matrice modificata
                    calcolaNumeroMosse(matrice_copia_2, mosse);
                }
            }
        }
    }
}
```

Questa funzione prende come parametri in input la matrice di gioco e il numero di mosse utilizzate.

La logica della funzione è quella di cercare nella matrice una cella avente una bolla in un determinato stato, il quale partirà da 1 e si incrementerà (fino a 3) ogni volta che avrà finito di leggere tutta la matrice.

Quando viene trovata una cella con lo stato corrispondente a quello ricercato la funzione incrementerà di 1 le mosse passate dalla chiamate della funzione e decrementerà invece di uno stato la bolla della matrice trovata, se lo stato diventerà uguale a 0 verrà richiamata la funzione esplosione() la quale propagherà l'esplosione (la vedremo meglio più avanti). A questo punto la funzione richiamerà in modo ricorsivo sé stessa passando come parametri di input la matrice modificata e il numero delle mosse aggiornate.

La funzione terminerà quando:

- la matrice sarà completamente vuota (vincita) in questo caso verrà salvato in una variabile globale il numero minimo di mosse trovate.
- il numero delle mosse (ricevute come input e non ancora incrementate) più lo stato ricercato sarà maggiore o uguale al numero minimo di mosse possibili.

N.B. la variabile globale contenente il numero minimo di mosse verrà valorizzata a 100 prima di richiamare la funzione per calcolare il numero minimo di mosse possibili, evitando così di aggiungere un controllo con numero mosse minimo != 0

In questo modo la funzione troverà prima una soluzione con un determinato numero N di mosse. Una volta trovata la prima soluzione tornerà a ritroso nelle matrici negli stati precedenti e cercherà altre nuove possibili soluzioni finché non avrà provato tutte le possibili combinazioni (si escluderanno quelle matrici che hanno un numero di mosse + lo stato ricercato maggiore uguale alle mosse minime).

Gestione esplosione

Come detto in precedenza quando viene toccata una bolla nello stato 1 questa esploderà e propagherà l'esplosione nelle 4 direzioni (sopra, sotto, destra, sinistra) fino alla prima bolla adiacente o fino al bordo della matrice.

La gestione dell'esplosione e della sua reazione a catena è gestita dalla funzione padre:

```
// Esplosione main che viene richiamata dalle altre funzioni esplosioni "figlie"
void esplosione(int matrice[ROW][COL], int row, int col) {
    esplosioneSu(matrice, row, col);          // L'esplosione si propaga in alto
    esplosioneGiu(matrice, row, col);         // L'esplosione si propaga in basso
    esplosioneDestra(matrice, row, col);      // L'esplosione si propaga a destra
    esplosioneSinistra(matrice, row, col);    // L'esplosione si propaga a
sinistra
}
```

Questa funzione richiamerà le funzioni figlie, che itereranno rispettivamente la matrice in una delle 4 direzioni finché non si incontrerà una bolla (cella diversa da 0) oppure il bordo. Quando durante l'iterazione si incontra una bolla questa decrementa il suo stato di 1 e nel caso lo stato diventi uguale a 0 viene richiamata ricorsivamente la funzione padre esplosione.

```
// Propaga l'esplosione verso l'alto
void esplosioneSu(int matrice[ROW][COL], int row, int col) {
    //Verifica che l'esplosione non sia avvenuta nella prima riga
    do {
        row--;
    } while(row > 0 && matrice[row][col] == 0);
    if(row >= 0 && matrice[row][col] != 0){ //Verifica che si è  usciti dal
ciclo perché si è trovata una riga div da 0
        matrice[row][col]--;
        if(matrice[row][col] == 0) esplosione(matrice, row, col);
    }
}
```

Gestione partita

Una volta calcolato il numero minimo di mosse possibili viene chiesto al giocatore, nel caso non sia già stata valorizzata, la difficolta della partita.

```
       B U B B L E     B L A S T

********************************************************************************************
 IMPOSTA DIFFICOLTA' (premi E per tornare al menu')

 Difficolta' attuale impostata su 3

 Seleziona una difficolta'
  1) Facile (4 mosse extra)
  2) Media (2 mosse extra)
  3) Difficile (nessuna mossa extra)

********************************************************************************************
```

Una volta impostata la difficoltà viene valorizzato il campo *max_mosse* della struttura Giocatore uguale alle mosse minime più il numero di mosse aggiuntive in base alla difficoltà scelta.

A questo punto viene richiamata la funzione partita passando come parametro il puntatore alla struct Giocatore.

```c
// Gestisce la modalità GIOCA in base agli input dell'utente
int partita(Giocatore *giocatore) {
    // Stampe a video
    stampamatriceUtente(giocatore->matrice_partita,giocatore->max_mosse,
giocatore->mosse_eff, giocatore->diff_partita); //Da inserire in input utente

    // Cicla finché la matrice non è tutta a 0 o finche il giocatore ha mosse
disponibili
    while(!controllaVincita(giocatore->matrice_partita) && giocatore->mosse_eff
< giocatore->max_mosse) {
        // Salva su file la partita in corso ad ogni mossa
        scriviGiocatoreSuFile(NOME_FILE,*giocatore);

        // Input utente
        int row, col;
        char input[10];
        do{
            printf("\n Inserisci una riga: ");
            fgets(input, sizeof(input), stdin);
            size_t length = strlen(input);
            if (input[length - 1] == '\n') {
                input[length - 1] = '\0';
            }
            if(toupper(*input) == 'E') return -1; //Torna al menu principale (la
partita ripartirà da capo)

            if (isInteger(input)) {
                row = atoi(input);
                row--;
                if(row < 0 || row > ROW) printf("\n Inserisci una riga da 1 a
%d.",ROW);
            } else {
                printf("\n Input non valido.");
            }
        } while(!isInteger(input) || row>ROW || row<0);

        do{
            printf("\n Inserisci una colonna: ");
            fgets(input, sizeof(input), stdin);
```

```
            size_t length = strlen(input);
            if (input[length - 1] == '\n') {
                input[length - 1] = '\0';
            }
            if(toupper(*input) == 'E') return -1; //Torna al menu principale (la
partita ripartirà da capo)
            if (isInteger(input)) {
                col = atoi(input);
                col--;
                if(col < 0 || col > COL) printf("\n Inserisci una colonna da 1 a
%d.",COL);
            } else {
                printf("\n Input non valido.");
            }
        } while(!isInteger(input) || col>COL || col<0);

        // Verifica che la cella inserita dal giocatore non sia vuota
        if (giocatore->matrice_partita[row][col] != 0) {
            // Stampe a video
            clear();
            printf("\tB U B B L E\t B L A S T\n");

printf("\n**************************************************************************
********************\n");
            printf(" Hai inserito R: %d e C: %d\n\n",row+1,col+1);

            // Incremento mosse effettuate giocatore
            giocatore->mosse_eff++;
            // Decremento stato bolla cella e gestione esplosione
            giocatore->matrice_partita[row][col]--;
            if (giocatore->matrice_partita[row][col] == 0) {
                esplosione(giocatore->matrice_partita,row, col);
                printf(" Hai colpito una bolla in procinto di esplodere! Pronti
all'esplosione.\n BOOOM!");
            }
            else if(giocatore->matrice_partita[row][col] == 1) printf(" Hai
colpito una bolla gonfia a meta'! Ora e' in procinto di esplodere.");
            else if(giocatore->matrice_partita[row][col] == 2) printf(" Hai
colpito una bolla sgonfia! Si gonfia a meta'");
            stampamatriceUtente(giocatore->matrice_partita,giocatore->max_mosse,
giocatore->mosse_eff, giocatore->diff_partita); //Da inserire in input utente
        }
        else printf("\n *Attenzione hai selezionato una cella vuota,
riprova!*\n");
    }

printf("\n**************************************************************************
********************\n");
    //Verifica che la partita si sia conclusa per vittoria o per perdita
    if(controllaVincita(giocatore->matrice_partita) && giocatore->mosse_eff <=
giocatore->max_mosse) {
        printf("\n Complimenti hai vinto!\t Difficolta': %d \t Mosse Minime: %d
\t Mosse utilizzate: %d ",giocatore->diff_partita,giocatore-
>max_mosse,giocatore->mosse_eff);
        return 1;
    }
    printf("\n Hai perso!\t Difficolta': %d \t Mosse Minime: %d \t Mosse
utilizzate: %d ",giocatore->diff_partita,giocatore->max_mosse,giocatore-
>mosse_eff);
    return 0;
}
```

Questa funzione gestisce la partita dell'utente chiedendo in input una riga e una colonna nel range prestabilito.

Quando l'utente inserisce la riga e la colonna di una cella avente il contenuto diverso da 0 incrementerà di uno le mosse utilizzate, si decrementerà di uno la bolla colpita ed eventualmente (in caso lo stato diventi uguale a 0) richiamerà la funzione esplosione.

La funzione partita continuerà a chiedere in input all'utente una riga e una colonna finché non si verificherà uno dei seguenti casi:

- la matrice avrà tutte le celle uguali a 0 (VINCITA)
- il numero di mosse utilizzate sarà maggiore delle mosse minime più le mosse extra in base alla difficoltà (PERDITA)
- l'utente inserisce il carattere 'E' (TORNA AL MENU')

Salvataggio partita

Ad ogni mossa dell'utente viene salvato sul file *giocatori.txt* la partita in corso in modo tale che il giocatore potrà riprendere la partita in corso. Quando la partita terminerà verranno aggiornati alcuni dati riguardanti la struct Giocatore e verranno salvati sul file separati dal ";".

Tutto ciò avviene attraverso questa funzione:

```c
// Scrive su file il nuovo Giocatore (solo se inizia una partita) o aggiorna i dati del giocatore già presente
void scriviGiocatoreSuFile(const char *nomeFile, Giocatore giocatore) {
    FILE *fileIn, *fileTemp;
    char riga[MAX_LENGTH];

    // Apre i due file in modalita' lettura e in modalità scrittura
    fileIn = fopen(nomeFile, "r");
    fileTemp = fopen("temp.txt", "w");

    if (fileIn == NULL || fileTemp == NULL) {
        perror("Errore durante l'apertura dei file");
        exit(EXIT_FAILURE);
    }

    // Verifica se il giocatore ha una partita in corso e lo scrive nella prima riganel file temp
    if(giocatore.max_mosse != 0){
        fprintf(fileTemp, "%s;%d;%d;%d;%d;%s;%s;%d;%d;%d;\n",
                giocatore.nome,
                giocatore.partite_giocate,
                giocatore.partite_vinte,
                giocatore.miglior_sequenza,
                giocatore.punteggio,
                ottieni_data(),
                matrice_toString(giocatore.matrice_partita),
                giocatore.max_mosse,
                giocatore.mosse_eff,
                giocatore.diff_partita
        );
    } else {
        fprintf(fileTemp, "%s;%d;%d;%d;%d;%s;%s;%d;%d;%d;\n",
                giocatore.nome,
                giocatore.partite_giocate,
                giocatore.partite_vinte,
                giocatore.miglior_sequenza,
                giocatore.punteggio,
```

```
                ottieni_data(),
                "",
                0,
                0,
                0
        );
    }

    // Ricopia tutti le righe dal file principale a quello temp tranne (se
esiste) quella del giocatore scritto per primo
    while (fgets(riga, MAX_LENGTH, fileIn) != NULL) {
        // Verifica se la riga contiene il dato da rimuovere
        if (strstr(riga, giocatore.nome) == NULL) {
            // Se non contiene il dato, scrive la riga nel file temporaneo
            fputs(riga, fileTemp);
        }
    }

    // Chiusura dei file
    fclose(fileIn);
    fclose(fileTemp);

    // Rimuove il file originale e rinomina il file temporaneo
    remove(nomeFile);
    rename("temp.txt", nomeFile);
    remove("temp.txt");
}
```

Questa funzione scriverà i record del giocatore nella prima riga di un file temporaneo *temp.txt* e ricopierà tutti gli altri record presenti nel file *giocatori.txt* tranne quello corrisponde al giocatore inserito nella prima riga. Scritti tutti i record rimoverà il file *giocatori.txt* e rinominerà il file temporaneo.

Statistiche di gioco

Selezionando l'opzione "S" viene mostrato al giocatore l'elenco delle sue statistiche.

```
       B U B B L E     B L A S T

*********************************************************************************************
 STATISTICHE DEL GIOCATORE (premi E per tornare al menu')

  Nome: Gabriel
  Data ultima partita: 20/12/2023 14:11:13
  Partite giocate: 5
  Partite Perse: 1
  Partite totali vinte: 4
  Miglior sequenza: 3
  Miglior punteggio: 180
  Percentuale di vittoria: 80.00%

*********************************************************************************************
```

(Il punteggio viene calcolato in base al numero di partite vinte di fila per la difficolta e moltiplicato per 10)

Valorizzazione struttura Giocatore

L'elenco di statistiche è stampato a video con i dati presenti nella struttura Giocatore, i quali ricordo vengono valorizzati all'inizio del programma leggendo i file *giocatori.txt* grazie a queste alle funzioni valorizzaGiocatore e leggiGiocatoriDaFile le quali ritornano una struct Giocatore.

```c
// Valorizza la struttura Giocatore con tutti valori di default ad eccezione del
nome
Giocatore valorizzaGiocatore(char *user) {
    Giocatore giocatore;
    giocatore.nome = strdup(user);
    giocatore.partite_giocate = 0;
    giocatore.partite_vinte = 0;
    giocatore.miglior_sequenza = 0;
    giocatore.data_ultima_partita = "";
    giocatore.matrice_partita;
    giocatore.max_mosse = 0;
    giocatore.mosse_eff = 0;
    giocatore.diff_partita = 0;
    return giocatore;
}

// Verifica la presenza di un giocatore con lo stesso nome nel file definito in
NOME_FILE ed eventualmente
// carica i dati nella struttura Giocatore
Giocatore leggiGiocatoriDaFile(const char *nomeFile, char *user) {
    // Valorizzazione giocatore di default
    Giocatore giocatore = valorizzaGiocatore(user);

    // Apertura file di lettura
    FILE *file = fopen(nomeFile, "r");

    if (file == NULL) {
        // Il file non esiste, lo creiamo
        file = fopen(nomeFile, "w"); // Apre il file in modalità scrittura
        if (file != NULL) {
           fclose(file); // Chiude il file dopo averlo creato
        } else {
            perror("Errore nella creazione del file.\n");
            return giocatore;
```

```
        }
    }

    char buffer[MAX_LENGTH]; // Buffer per salvare la riga letta
    char *token;
    char *tokens[150];
    int num_tokens = 0;
    // Legge una riga alla volta finché non raggiunge la fine del file
    bool flg_trovato = false;
    while (fgets(buffer, MAX_LENGTH, file) != NULL) {
        // Verifica se nella riga letta nella prima posizione è presente la
stringa uguale al nome del giocatore in cas valorizza il flg_trovato
        token = strtok(buffer, ";");
        if(strcmp(token, giocatore.nome) == 0) {
            flg_trovato = true;
            break;
        }
    }

    //Se è stato trovato il giocatore nel file valorizza il giocatore con i dati
trovati
    if(flg_trovato) {
        while (token != NULL && num_tokens < 100) {
            tokens[num_tokens++] = token;
            token = strtok(NULL, ";");
        }
        if(tokens[1] != NULL && isInteger(tokens[1])) giocatore.partite_giocate
= atoi(tokens[1]);
        if(tokens[2] != NULL && isInteger(tokens[2])) giocatore.partite_vinte =
atoi(tokens[2]);
        if(tokens[3] != NULL && isInteger(tokens[3])) giocatore.miglior_sequenza
= atoi(tokens[3]);
        if(tokens[4] != NULL && isInteger(tokens[4])) giocatore.punteggio =
atoi(tokens[4]);
        if(tokens[5] != NULL) giocatore.data_ultima_partita = strdup(tokens[5]);

        // Verifica se è presente la stringa della matrice corretta e la
valorizza
        bool flag_matrice = true;
        bool flag_vuota = true;
        if(tokens[6] != NULL && isInteger(tokens[6])) {
            int indice_carattere = 0;
            for (int riga = 0; riga < ROW; riga++) {
                for (int colonna = 0; colonna < COL; colonna++) {
                    if (tokens[6][indice_carattere] != '\0') {
                        giocatore.matrice_partita[riga][colonna] =
tokens[6][indice_carattere] - '0'; // Conversione da carattere a intero
                        if(giocatore.matrice_partita[riga][colonna] != 0)
flag_vuota = false;
                        indice_carattere++;
                    }
                    else flag_matrice = false;
                }
            }
        }
        // Se è presente la matrice valorizza anche gli altri campi relativi
alla partira in corso
        if(flag_matrice && !flag_vuota) {
            if(tokens[7] != NULL && isInteger(tokens[7])) giocatore.max_mosse
= atoi(tokens[7]);

            if(giocatore.max_mosse == 0) {
                printf("\n Calcolo numero mosse minime in corso...");
```

```
                maxMosse = 100;
                calcolaNumeroMosse(giocatore.matrice_partita,0);
                giocatore.max_mosse = maxMosse;
            }

            if(tokens[8] != NULL && isInteger(tokens[8])) giocatore.mosse_eff
= atoi(tokens[8]);
            if(tokens[9] != NULL && isInteger(tokens[9])) giocatore.diff_partita
= atoi(tokens[9]);
        }
    }
    // Chiusura del file
    fclose(file);

    // Ritorna la struttura del giocatore
    return giocatore;
}
```

La funzione *valorizzaGiocatore* (che verrà richiamata all'interno della seconda funzione) valorizza gli attributi della struct Giocatore a valori di default (tranne per il nome) mentre la funzione *leggiGiocatoriDaFile* cerca nel file di testo la presenza di un giocatore con lo stesso nome, se lo trova valorizza i vari campi della struct con i dati presenti nel file. In particolare, durante la lettura se si accorgerà della presenza di una stringa nel formato corretto nel campo *matrice_partita* verrà letto ad uno ad uno ogni singolo carattere della stringa, convertito in un *int* e inserito in una cella della matrice. Inoltre, qualora fosse presente una matrice valorizzata e il campo mosse minime sia uguale a 0 lo si valorizzerà richiamando la funzione per calcolare le mosse minime.

Impostazione difficoltà

Selezionando l'opzione "D" verrà data all'utente la possibilità di cambiare la difficoltà.

```
       B U B B L E     B L A S T

*********************************************************************************************
 IMPOSTA DIFFICOLTA' (premi E per tornare al menu')

 Difficolta' attuale impostata su 3

 Seleziona una difficolta'
  1) Facile (4 mosse extra)
  2) Media (2 mosse extra)
  3) Difficile (nessuna mossa extra)

*********************************************************************************************
```

Quando l'utente selezionerà una nuova difficoltà e caricherà una partita ancora in corso le mosse minime si aggiorneranno in automatico grazie a questa funzione:

```
// Aggiunge o rimuove mosse alle mosse massime in base:
// alle mosse massime (se presente), alla difficolta precedente (se presente) e
alla difficolta nuova
int impostaDifficolta(int max_mosse, int difficolta_old, int difficolta_new) {
    if(difficolta_old == difficolta_new) return 0;
    if(max_mosse == 0){ //Verifica se le mosse massime disponibili al giocatore
sono già state calcolate
        if(difficolta_new == 1) return 4; //Difficolta facile
        if(difficolta_new == 2) return 2; //Difficolta media
    }
    else {
        if(difficolta_old == 3) { //Se la vecchia difficolta' era 3 (difficile)
aggiunge 4 per diff. facile e 2 per diff. media
            if(difficolta_new == 1) return 4; //Difficolta facile
            if(difficolta_new == 2) return 2; //Difficolta media
        }
        if(difficolta_old == 2) { //Se la vecchia difficolta' era 2 (media)
aggiunge 2 per diff. facile e ne toglie 2 per diff. difficile
            if(difficolta_new == 1) return 2; //Difficolta facile
            if(difficolta_new == 3) return -2; //Difficolta difficile
        }
        if(difficolta_old == 1) { //Se la vecchia difficolta' era 1 (facile)
toglie 2 per diff. media e 4 per diff. difficile
            if(difficolta_new == 2) return -2; //Difficolta media
            if(difficolta_new == 3) return -4; //Difficolta difficile
        }
    }
    return 0;
}
```

Questa funzione prende in input le mosse minime presenti nella struct giocatore, qualora siano diversa da 0 significherà che è presente una partita in corso e quindi le mosse aggiuntive andranno ricalcolate in base alla nuova difficoltà impostata rispetto alla difficoltà presente in precedenza.

Cambio giocatore

Selezionando l'opzione "N" si può cambiare l'utente di gioco.

```
       B U B B L E     B L A S T

********************************************************************************************
 CAMBIA GIOCATORE

  Inserisci il nome del giocatore: Giovanni

  E' stato impostato il giocatore: Giovanni
********************************************************************************************
 Seleziona Y per confermare o N per annullare
```

Inserendo in input un nuovo nome verrà valorizzata la struttura Giocatore (*come spiegato precedentemente*) ma in una variabile temporanea, la quale nel caso in cui l'utente confermasse il cambio, verrà copiata nella variabile "ufficiale". Se l'utente inserirà un nome uguale a quello inserito il programma eviterà di leggere nuovamente il file *giocatori.txt* e manterrà il nome del giocatore corrente.

Stampa regole

Inserendo il carattere "R" viene mostrato a video all'utente le regole di gioco.

```
      B U B B L E     B L A S T

 Lo scopo del gioco e' semplice: fai esplodere tutte le bolle prima di terminare i tentativi.
 Appena partira' il gioco vedrai una matrice con dei numeri al suo interno.
 I tipi di bolle che puoi trovare sono 3:
        1) Bolla in procinto di esplodere;
        2) Bolla gonfia a meta';
        3) Bolla sgonfia.
 Quando colpirai una bolla sgonfia (3) la gonfierai a meta' (2).
 Quando colpirai una bolla gonfia a meta' (2) la renderai in procinto di esplodere (1).
 Quando colpirai una bolla in procinto di esplodere (1) la farai esplodere (0).
 Facendo esplodere una bolla di tipo 1 creerai un'esplosione a catena che coinvolgera' le bolle nelle quattro direzioni
rispetto a quella da te colpita!
 Vinci quando fai esplodere tutte le bolle presenti sulla matrice, perdi se non riesci nei tentativi possibili.

 INIZIAMO!

 Premi E per tornare al menu'
```