NICOLÒ PINCIROLI

SCARABEO

DOCUMENTAZIONE

POLIMI INGEGNERIA INFORMATICA A.A. 2016 - 2017

Nicolò Pinciroli,
Scarabeo
Documentazione

E-MAIL:
nicolopinci.1997@gmail.com

Documentazione realizzata con LaTeX.

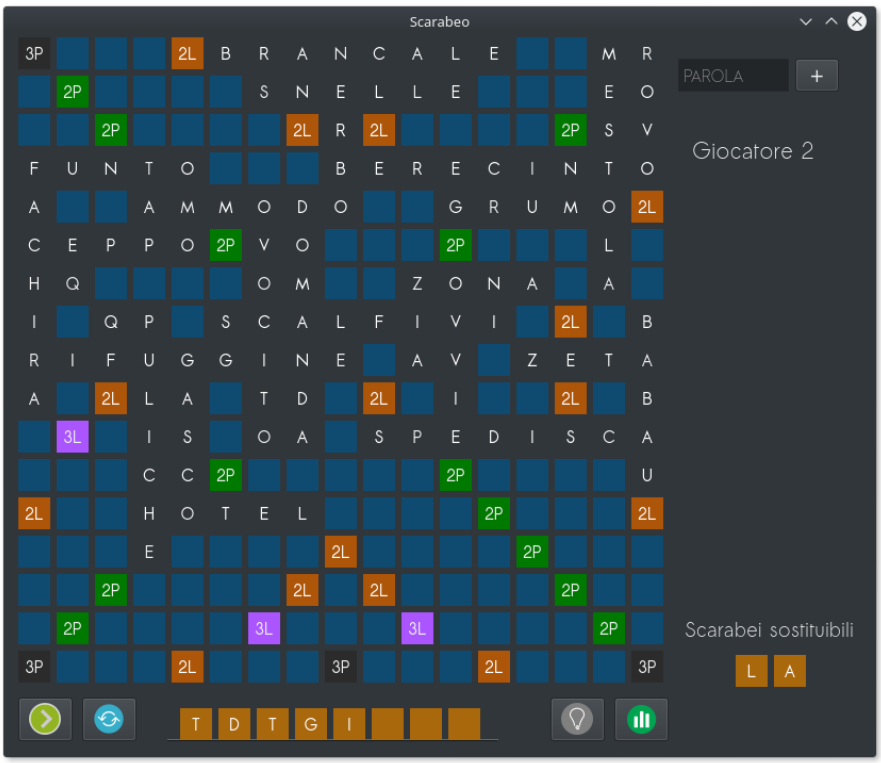


Figura 1: Schermata con alcune parole ottenute dai suggerimenti

INDICE

1 ANALISI DEI REQUISITI

1.1 INTRODUZIONE ALLE REGOLE DEL GIOCO

Il gioco *Scarabeo* può essere giocato da 2 a 4 giocatori. Il gioco si svolge su una griglia quadrata, di dimensione 17x17.

Ogni giocatore ha a disposizione un leggio, sul quale può posizionare otto lettere pescate da un sacchetto, che inizialmente ne contiene 130.

Al proprio turno il giocatore può comporre delle parole che si incrocino con quelle già presenti sul tabellone (tranne nel caso della prima, che però deve necessariamente passare per il centro del tabellone) usando le lettere presenti sul proprio leggio.

Se il giocatore di turno non può (o non vuole) comporre parole ha la possibilità di passare il turno, sia cambiando tutte le lettere sul proprio leggio sia mantenendo le lettere correnti. Ogni giocatore, dopo aver composto la propria parola, deve passare il turno al giocatore successivo.

Si ha un maggior numero di vocali (12 A, E, I, O ma solo 4 U) e poi a scalare le consonanti, sempre a seconda della frequenza (si avranno quindi 7 C, R, S, T; 6 L, M, N; 4 B, D, F, G, P, V; 2 H, Q e Z). Ci sono inoltre 2 tessere jolly, con la figura di uno scarabeo (che per questo sono anche chiamate *scarabei*).

Per quanto riguarda i jolly, questi possono essere sostituiti a qualsiasi lettera, assumendone il valore. Se un giocatore possiede la lettera corrispondente ad un jolly ha la possibilità di sostituirlo (una volta arrivato il suo turno).

Il punteggio è calcolato in base al numero presente su ogni tessera. Le lettere più frequenti hanno un punteggio più basso rispetto a quelle più rare. Per ogni parola composta, il giocatore calcola la somma dei punteggi delle lettere. Il punteggio di ogni lettera è il seguente:

- 1 punto per ogni A, C, E, I, O, R, S, T;
- 2 punti per L, M, N
- 3 punti per P;
- 4 punti per B, D, F, G, U, V;
- 8 punti per H e Z;
- 10 punti per Q.

Sulla griglia sono presenti delle particolari caselle che permettono di raddoppiare o triplicare il punteggio della lettera posta su di esse (indicate rispettivamente con 2L e 3L) oppure di raddoppiare o triplicare il punteggio ottenuto con l'intera parola (2P e 3P).

Altri punteggi bonus sono:

- 10 punti bonus se si compone una parola di 6 lettere;
- 30 punti bonus se la parola è di 7 lettere;
- 50 punti se è formata da tutte e 8 le lettere;
- 100 punti bonus se si riesce a scrivere la parola "scarabeo" o "scarabei".

A questi punteggi vanno sommati ulteriori 10 punti se non si sono utilizzati jolly nel comporre le parole.

Lo scopo del gioco è ottenere il maggior punteggio possibile una volta finite le lettere disponibili. L'ultimo giocatore ad aver composto una parola aggiunge al suo punteggio anche la somma di tutte le lettere presenti sui leggii degli altri giocatori e non ancora utilizzate.

1.2 SCOPO DEL PROGRAMMA

Il programma *Scarabeo* ha come scopo quello di permettere a più giocatori (da 2 a 4) di competere fra loro giocando una partita completa.

I giocatori hanno la possibilità di effettuare tutte le mosse della versione originale del gioco, quindi possono utilizzare le proprie lettere per comporre una parola, passare il turno senza cambiare le lettere presenti sul proprio leggio, passare il turno cambiando invece le lettere e sostituire gli eventuali jolly presenti sul tabellone.

Una prima funzionalità aggiuntiva del programma consiste nel controllo della validità della parola inserita (quindi controlla se la parola è presente all'interno del dizionario e allo stesso modo controlla anche le sue eventuali intersezioni).

Il giocatore di turno, inoltre, ha la possibilità di chiedere al programma la parola migliore in quel momento, se presente, ovvero la parola che permette di totalizzare più punti. Il programma considera valide tutte le combinazioni composte da due lettere.

1.3 INIZIO DELLA PARTITA

La partita inizia scegliendo il numero dei giocatori, che può variare da 2 a 4. Per compiere questa scelta si apre una finestra modale. Contemporaneamente il programma comincia il caricamento del dizionario, che naturalmente è indipendente dalla scelta effettuata.

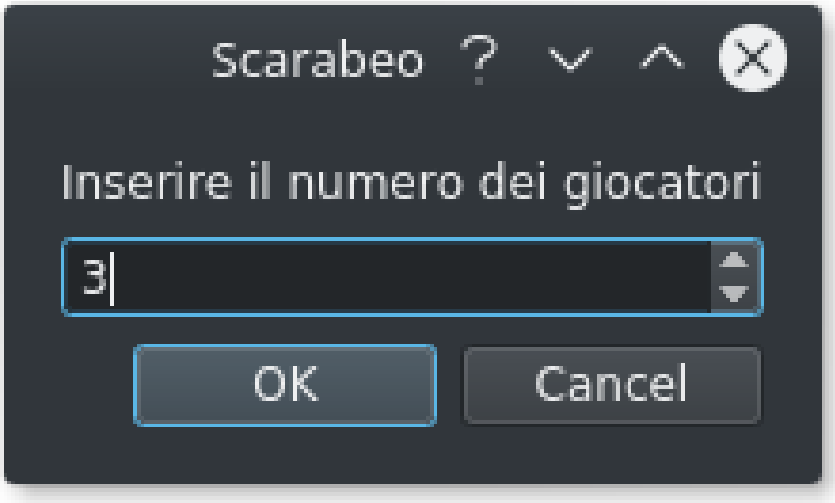


Figura 2: Impostazione del numero di giocatori

Affinché questa operazione sia possibile ho pensato di ricorrere ai thread [VV17f].

Terminato il caricamento del dizionario (e l'applicazione delle regole relative alle singole parole) si apre la finestra principale. In questa finestra vengono visualizzati il tabellone, il leggio del primo giocatore, i bottoni per la richiesta di suggerimento e per il passaggio del turno, la casella di testo per inserire la propria parola e il bottone per visualizzare i punteggi.

1.4 INSERIMENTO DI UNA PAROLA

Al proprio turno un giocatore può decidere di inserire una parola utilizzando le lettere che ha a disposizione e incrociando una parola già presente nel tabellone (o passando dalla casella centrale se la parola è la prima della partita).

Per inserire una parola può digitarla nella casella di testo in alto a destra, fare click sulla casella di partenza e premere sul bottone **+** di fianco alla casella di testo. A questo punto il programma controllerà se la parola esiste e, in caso di esito positivo, chiederà se si desidera inserire la parola in orizzontale o in verticale, altrimenti restituirà un messaggio di errore.

A questo punto il programma verifica che tutte le lettere necessarie siano in possesso del giocatore di turno e posiziona eventuali jolly utilizzati nella sezione chiamata *Scarabei disponibili*. I jolly sono indicati dal simbolo #. La parola non verrà inserita se non si interseca correttamente con quelle già presenti sul tabellone. Anche in questo caso verrà visualizzato un messaggio di errore (diverso dal precedente).

Il giocatore ha anche la possibilità di chiedere un suggerimento al

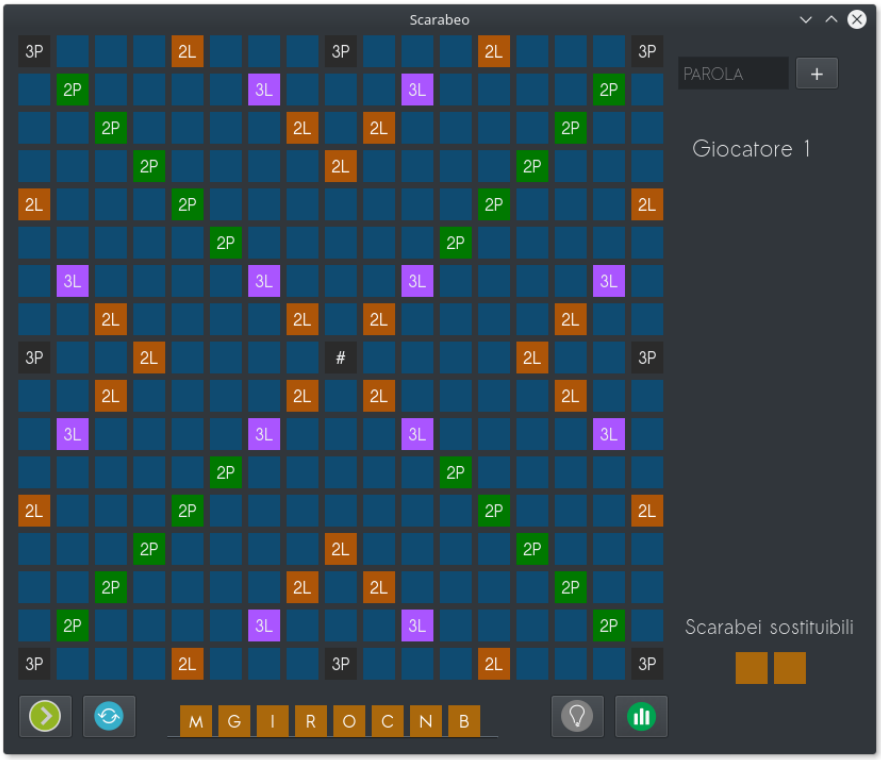


Figura 3: Schermata iniziale

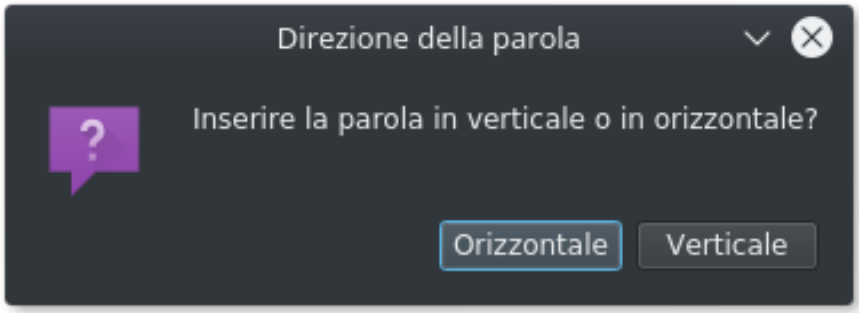


Figura 4: Direzione dell'inserimento

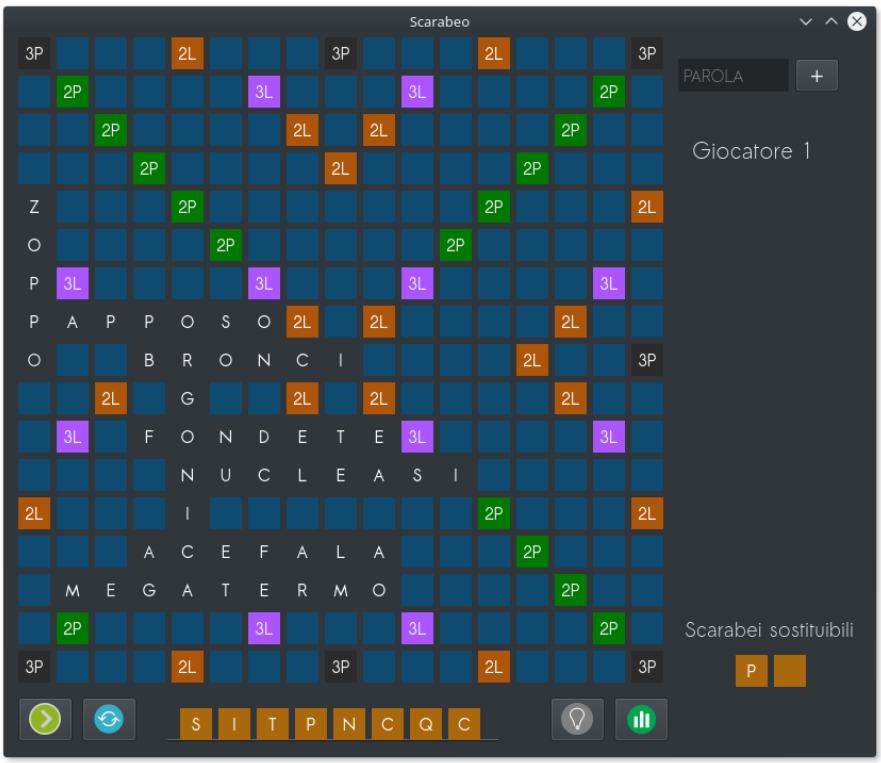


Figura 5: Schermata con alcune parole inserite

programma. Nel caso della prima parola l'algoritmo utilizzato è diverso rispetto a quello utilizzato per le parole successive. Il programma, infatti, a partire dalle lettere presenti sul leggio, genera tutte le combinazioni possibili e vede quali di queste sono parole valide.

In questa situazione, se sul leggio non sono presenti jolly, il programma valuterà fino a $8! + 7! + ... + 1!$ combinazioni, ovvero 46233 possibili parole. Se dovesse essere presente un jolly le combinazioni diventano al più $21 \cdot (7! + 6! + ... + 1!)$, ovvero 124173 e, infine, con due jolly diventano al più $21 \cdot 21 \cdot (6! + 5! + ... + 1!)$, ovvero 384993.

I suggerimenti per le parole successive alla prima, invece, considerano tutte le possibili combinazioni di spazi e lettere presenti sul tabellone, verificando se le lettere presenti sul leggio del giocatore corrente possono essere inserite in queste strutture formando delle parole (e delle intersezioni) valide.

La ricerca della parola migliore, in questa situazione, nella versione iniziale del programma poteva avvenire in due modalità: la scansione completa (ovvero sul dizionario contenente le parole di base - circa 60000 - a cui vengono aggiunte quelle generate attraverso le regole relative al dizionario) e la scansione parziale (che opera solo e soltanto sulle parole contenute nel dizionario di base).

Naturalmente il tempo di esecuzione richiesto sarà maggiore nel primo dei due casi. Nel secondo caso, invece, la ricerca impiega mediamente qualche secondo.

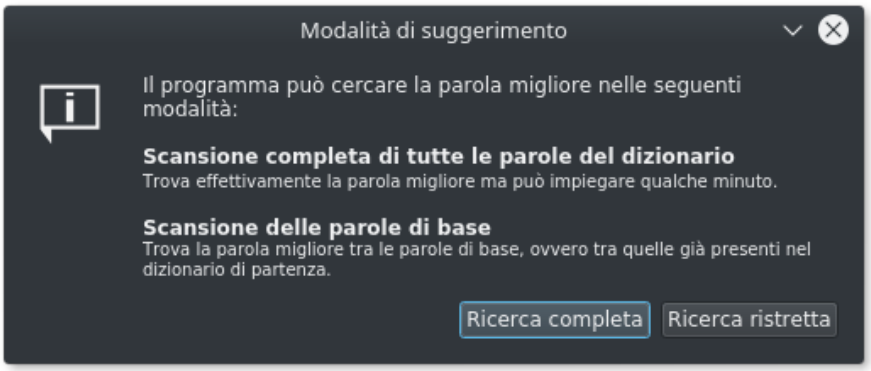


Figura 6: Richiesta di suggerimento nella prima versione del programma

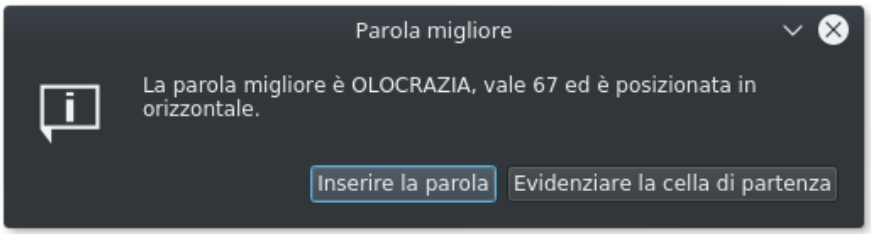


Figura 7: La parola migliore

La versione finale, invece, prevede di utilizzare soltanto il dizionario completo, infatti ho pensato di memorizzare i dati in modo più efficiente, rendendo il programma più rapido.

1.5 FINE DELLA PARTITA

La partita termina quando uno dei giocatori finisce le proprie lettere e il sacchetto è vuoto. In quel caso il giocatore che ha finito le lettere aggiunge al suo punteggio quello corrispondente alle lettere che gli altri giocatori hanno ancora sui propri leggii.

Una volta svolto questo calcolo, il programma è in grado di decretare il vincitore (o i vincitori ex-aequo), che avrà il punteggio maggiore. Naturalmente non è detto che il vincitore sia chi finisce per primo le lettere.

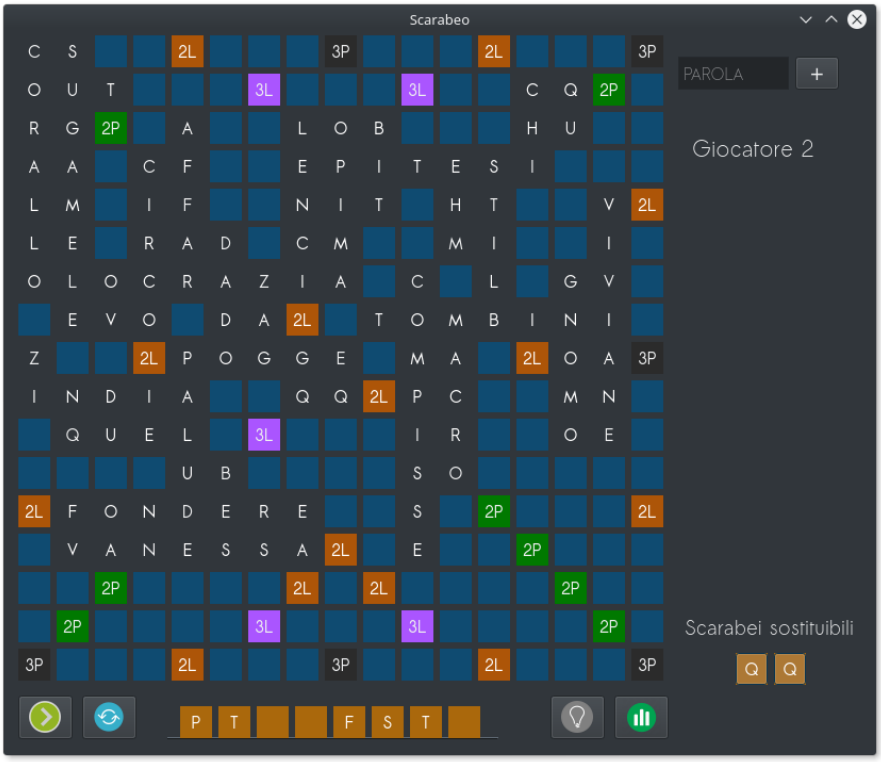


Figura 8: Partita quasi finita

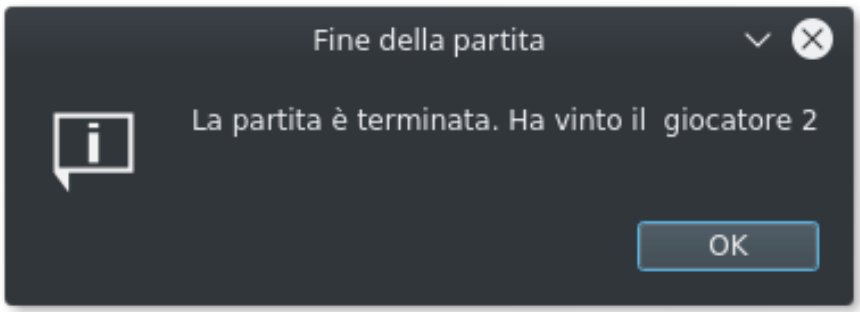


Figura 9: Vincitore della partita

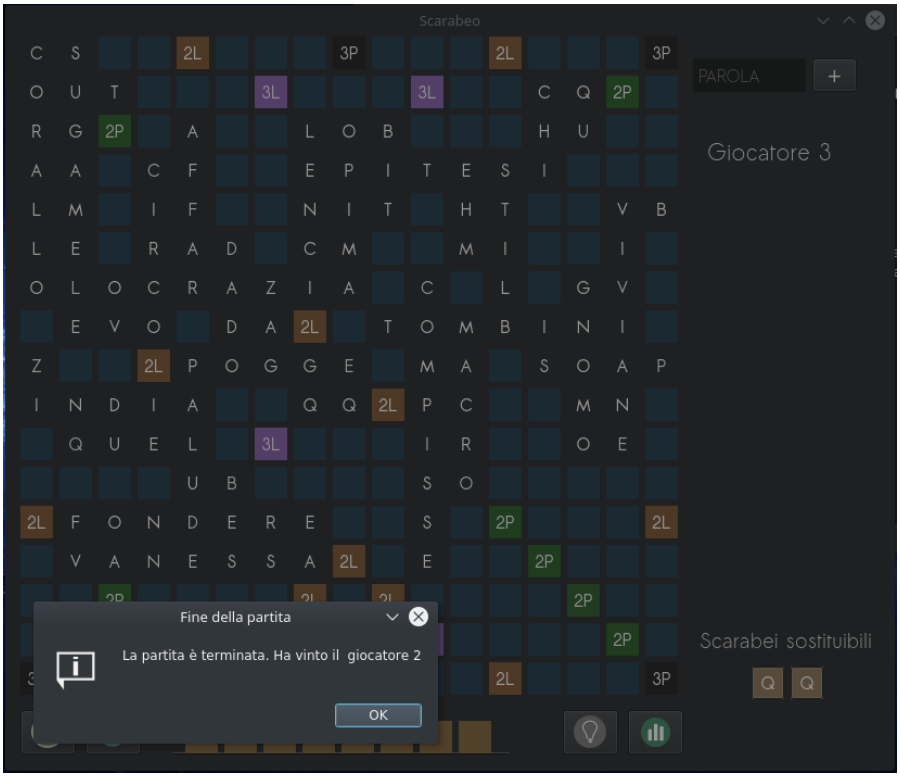


Figura 10: Schermata finale di una partita con tre giocatori

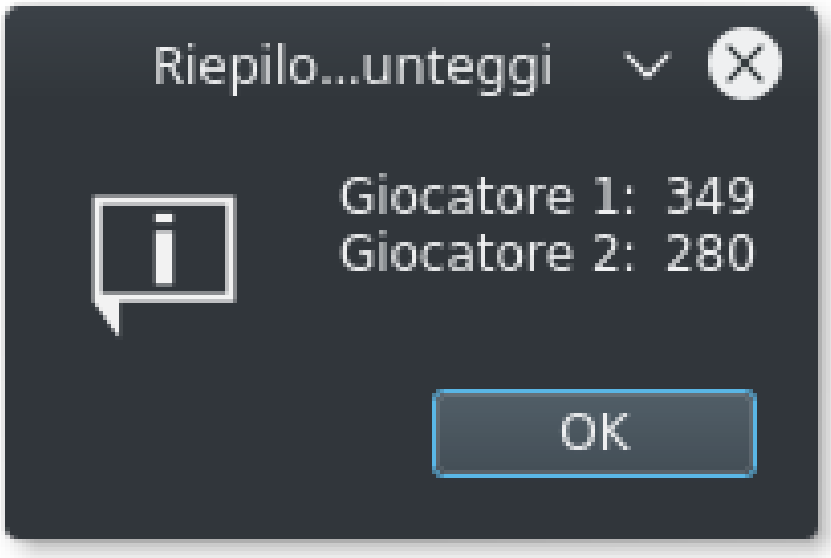


Figura 11: Riepilogo dei punteggi dei due giocatori

2 PROGETTAZIONE DELLA SOLUZIONE

2.1 PRINCIPALI STRUTTURE DATI

Il programma fa uso di alcune variabili globali, utilizzate nel caso in cui si riferiscano effettivamente a dati concettualmente "globali", ovvero riferiti a tutto il programma.

In particolare sono presenti alcuni dati riferiti ai **giocatori**. Si tratta dei punteggi (variabili intere), della variabile contente il giocatore corrente (ancora una volta si tratta di un intero che varia da 1 a 4) e dei leggii (di tipo stringa).

Per quanto riguarda il tabellone di gioco, questo è rappresentato da una matrice di caratteri 17x17x3. I livelli della matrice (0, 1, 2) servono per contenere, rispettivamente, le lettere, i moltiplicatori di lettera e i moltiplicatori di parola.

Inizialmente il livello 0 contiene solamente spazi bianchi, mentre i livelli 1 e 2 sono generati seguendo l'impostazione del tabellone descritta dalle regole del gioco.

Un altro dato globale è costituito dai jolly sostituibili. Questi ultimi sono memorizzati in una variabile di tipo array di caratteri e inizialmente hanno come valore uno spazio bianco.

Ho deciso di mantenere queste variabili globali perché la loro vita è effettivamente estesa a tutto il programma e per migliorare la leggibilità dello stesso.

Le strutture dati fondamentali per il funzionamento del programma sono le `map` [VV17a] e le `multimap` [VV17b], utilizzate per memorizzare il dizionario e le regole. Per utilizzare queste strutture dati bisogna includere la libreria **`map`**.

Le mappe permettono di indicizzare un elenco di coppie di elementi ordinati secondo il primo elemento della coppia.

Nel caso del dizionario le coppie sono formate da stringhe. La prima stringa corrisponde alla parola, mentre la seconda corrisponde alle eventuali regole collegate alla parola, lette dal file .dic fornito.

Per quanto riguarda la mappa delle regole, invece, questa contiene coppie formate da una regola (carattere) e da una `struct` (che contiene le informazioni su quella regola, ottenute dal file .aff, adattato per generare solo e soltanto le parole effettivamente ammesse dal gioco).

La struttura delle caratteristiche di una regola è la seguente:

```
typedef struct {
    string presuf;
    string togli;
```

```
    string metti;
    string dove;
} regola;
```

Nella seconda versione, più efficiente rispetto alla prima, ho introdotto un nuovo dato, ovvero una matrice di `vector` di `iterator` di `map` di coppie di `string`. Questa struttura serve per memorizzare i puntatori alle parole che hanno una lettera x in posizione y. Tali puntatori verranno memorizzati nel vector in posizione [y][pos_x]. In questo modo la scansione delle parole, durante i suggerimenti, si limita a considerare circa 1/21 del dizionario[1]

Durante la ricerca delle parole viene utilizzata anche una `struct` (chiamata `DatiParola`), in modo da memorizzare i dati riguardanti la parola migliore. Visto che la ricerca è separata su quattro thread, al termine del processo il programma valuta quale delle quattro parole ha punteggio maggiore.

La struttura in questione è definita nel modo seguente:

```
typedef struct
{
    string parola;
    int punteggio;
    string LeggioR;
    int riga;
    int colonna;
    bool maxdirvert; // true per parole verticali
} DatiParola;
```

2.2 ESTRAZIONE DELLE LETTERE DAL SACCHETTO

Ho pensato di creare il sacchetto delle lettere come **`vector`** di `elemSacc`, dove `elemSacc` è un tipo definito attraverso una `struct`:

```
typedef struct
{
    char Lettera;
    int num;
    int val;
} elemSacc;
```

1 Ad esempio, il puntatore alla parola *scarabeo* verrà memorizzata in posizione [0][18], infatti la lettera s (la diciottesima dell'alfabeto partendo da 0) si trova in posizione 0 all'interno della parola. Il puntatore, però, verrà memorizzato anche in posizione [1][2], infatti la seconda lettera è la c. Il programma prosegue fino alla nona lettera della parola, infatti sicuramente la prima lettera di una struttura candidata a formare una parola sarà in una posizione compresa fra 0 e 9, altrimenti le lettere sul leggio sarebbero insufficienti.

In questo tipo `Lettera` indica la lettera considerata, `num` indica la quantità di quella lettera nel sacchetto e val indica il suo valore. Ho assegnato al jolly un valore arbitrario pari a 0, visto che assumerà un valore diverso in base alla lettera sostituita.

Il sacchetto è vuoto quando la somma dei `num` di tutte le lettere è pari a 0.

Nel momento dell'estrazione della lettera, se il sacchetto non è ancora vuoto, viene estratto un numero casuale compreso fra 0 e il numero di lettere ancora presenti (ovvero la somma di tutti i campi `num`).

Successivamente il programma calcola, per ogni lettera, l'intervallo di valori a cui appartiene in base alla quantità delle lettere precedenti. Ad esempio, se la somma delle quantità delle lettere precedenti è k e una lettera è presente m volte nel sacchetto, allora verrà estratta se e solo se il numero estratto è compreso fra k e k+m, con k ed m in $\mathbb{N}$.

In questo modo si riesce a simulare in modo migliore l'estrazione "reale" delle lettere: una lettera presente in quantità maggiore ha più probabilità di essere estratta.

Una volta estratta la lettera, la quantità rimanente all'interno dell'elemento `elemSacc` corrispondente viene decrementata di un'unità.

2.3 GESTIONE DEI TURNI E DEI LEGGII

2.3.1 Gestione dei turni

La gestione dei turni è stata realizzata attraverso delle funzioni che permettono di cambiare il valore della variabile `Giocatore`. Ad ogni giocatore viene associato un leggio, quindi ho realizzato una funzione che permetta di stabilire il leggio corrente per ogni turno in base al numero di giocatore.

Questa parte del programma probabilmente sarebbe risultata molto leggibile con l'utilizzo delle classi, infatti la classe `Giocatore` avrebbe potuto avere come campi il numero di giocatore, il leggio associato e il punteggio, per esempio.

Nel mio programma, invece, ho deciso di mantenere queste variabili separate e di utilizzare le classi soltanto per la realizzazione dell'interfaccia grafica con Qt.

Il cambio di giocatore, di conseguenza, è gestito in questo modo:

```
void GiocatoreSuccessivo()
{
    if(Giocatore==1){ Giocatore=2;}

    else if(Giocatore==2)
    {
        if(numgioc==2){ Giocatore=1;}
```

```
        else{ Giocatore=3;}
    }

    else if(Giocatore==3)
    {
        if(numgioc==3){ Giocatore=1;}
        else{ Giocatore=4;}
    }

    else if(Giocatore==4){ Giocatore=1;}
}
```

2.3.2 Gestione dei leggii

Durante l'elaborazione dei suggerimenti il programma deve controllare se una determinata parola esiste e se le lettere presenti sul leggio sono sufficienti per formarla.

I leggii, di conseguenza, vengono copiati durante la fase dei suggerimenti, in modo che le lettere richieste per formare una parola vengano eliminate di volta in volta (ma solo dalla copia dal leggio). Questo permette di operare in modo semplice anche su parole con lettere ripetute.

2.4 GESTIONE DEL DIZIONARIO

2.4.1 Esistenza di una parola

Per verificare l'esistenza di una parola il programma, al posto di scorrere l'intero dizionario, utilizza la funzione find delle mappe [VV17a].

Questa funzione permette di effettuare una ricerca logaritmica all'interno del dizionario. Su circa 600000 di parole, quindi, il programma ne controllerà in media 10 (ovvero $\frac{1}{2}\log_2 600000$), mentre con la scansione lineare ne controllerebbe mediamente 300000).

L'implementazione dell'algoritmo, quindi, diventa molto semplice:

```
bool ParolaEsiste(string parola, map<string, string>& Diz)
{
    for(auto &c:parola)
    {
        c=tolower(c);
    }
    return (Diz.find(parola) != Diz.end());
}
```

2.4.2 Importazione del dizionario di base

Il dizionario di base è contenuto all'interno del file it_IT.dic ed è formato da circa 60000 parole (vengono infatti esclusi i nomi propri, come da regolamento), ognuna delle quali occupa una riga. Ogni parola può essere seguita dal simbolo / e da un insieme di caratteri, che indicano il nome delle regole che possono essere applicate a quella parola.

Il dizionario viene importato in una struttura chiamata `map` [VV17a], che permette di mantenere le parole ordinate. Per verificare se esiste una determinata parola, quindi, è sufficiente una ricerca dicotomica, che ha complessità pari a $\log_2 n$, dove n è il numero di parole.

2.4.3 Generazione del dizionario completo

Il dizionario completo è ottenuto dal dizionario di base in seguito all'applicazione delle regole appena descritte. La struttura delle regole è contenuta in un file .aff.

Ai fini della gestione e dell'applicazione delle regole dello scarabeo ho modificato il file .aff fornito, in modo da eliminare tutte quelle regole che avrebbero generato parole non valide, come quelle con apostrofi e accenti. Ho mantenuto solamente le regole vere e proprie, ignorando tutto il resto.

Ogni regola viene descritta in almeno una riga del file. In particolare, data una regola identificata da un nome, ne possono esistere varie versioni, che sono descritte in righe diverse del file.

Ogni riga ha la stessa struttura e si compone di vari campi, separati da spazi [VV17c]:

- `SFX` (suffisso) o `PFX` (prefisso): indica se la regola riguarda un prefisso o un suffisso della parola.
- Nome della regola (un carattere)
- Caratteri da togliere dalla fine della parola (`0` indica che non bisogna togliere caratteri) nel caso in cui a regola sia di tipo `SFX`
- Caratteri da aggiungere alla fine della parola (`0` indica che non bisogna aggiungerne) nel caso in cui la regola sia di tipo `SFX`
- Condizioni relative all'utilizzo della regola (ovvero in quali situazioni è applicabile e in quali non lo è)

Le condizioni relative alla regola possono essere composte. Questo avviene quando vengono indicate più lettere racchiuse tra parentesi quadre. In tal caso la parola è generabile se finisce con una delle lettere indicate.

Il simbolo ^, invece, indica la negazione, quindi la lettera o le lettere precedute da quel simbolo sono le lettere finali per le quali quella regola non è applicabile.

Il dizionario ottenuto - che si limita all'applicazione di alcune regole - è composto da circa 600000 di parole.

2.5 SUGGERIMENTI

2.5.1 Prima parola

L'algoritmo di ricerca del suggerimento per la prima parola è molto semplice, ma diverso rispetto a quello utilizzato per le parole successive.

In pratica, a partire dalle lettere presenti sul leggio, genera tutte le possibili combinazioni di lettere (considerando gli eventuali scarabei).

Successivamente valuta quali di queste formano delle parole valide e calcola quella con il punteggio maggiore in base alle diverse celle di partenza che possono avere, considerando il fatto che la prima parola deve necessariamente passare per il centro. Visto che la griglia è simmetrica, la prima parola viene sempre valutata in orizzontale: se venisse valutata anche la migliore parola verticale risulterebbero gli stessi punteggi e si tratterebbe di un calcolo superfluo.

2.5.2 Parole diverse dalla prima

L'algoritmo di ricerca della parola migliore in questo caso parte dalle parole già presenti sul tabellone.

Il tabellone è formato da 289 celle. L'algoritmo scorre tutte le celle che possono essere le iniziali di una parola, ovvero tutte le celle tali che gli 8 spazi a destra e gli 8 spazi in basso non siano tutti vuoti.

A partire da queste celle l'algoritmo genera delle strutture, ovvero delle stringhe, che iniziano con il dato contenuto nella cella di partenza (al livello 0). L'algoritmo aggiunge qualcosa alla struttura finché non ha raggiunto un totale di k spazi (anche non consecutivi) o la fine della matrice. Una volta raggiunti i k spazi la struttura viene allungata soltanto se le celle successive esistono e non sono degli spazi bianchi.

Il numero k di spazi è variabile, infatti, data una cella, vengono originate strutture con 1, 2, ..., 8 spazi (o meno se si raggiunge il limite della matrice) sia in orizzontale che in verticale.

Il programma verifica, quindi, che la struttura generata non sia formata da soli spazi o da sole lettere. Se questo non succede comincia a scorrere tutte le parole della porzione di dizionario compatibile (come descritto in precedenza) e vede se possono essere inserite nella struttura[2].

2 Nella versione originale venivano considerate tutte le parole del dizionario, rendendo i suggerimenti più lenti

In particolare, viene confrontata ogni lettera della struttura con ogni lettera della parola. Il confronto non avviene per le posizioni della struttura che contengono spazi bianchi.

Non appena un carattere non è compatibile con la struttura la parola viene scartata. Naturalmente l'algoritmo, per ogni lettera da inserire (ovvero dove la struttura contiene degli spazi bianchi), controlla se la lettera è presente nel leggio. Se questo non accade la parola viene scartata.

A questo punto, se la parola è valida, il programma verifica se esistono tutte le intersezioni che quella parola formerebbe con le parole già esistenti. In questo caso la ricerca è di complessità logaritmica, visto che avviene su un insieme di dati ordinati.

Se anche le intersezioni sono valide allora il programma calcola il punteggio della parola (considerando anche le intersezioni) e se questo è maggiore del punteggio più alto ottenuto fino a quel momento considera la nuova parola come migliore, per il momento.

Data una parola orizzontale, per esempio, il programma valuta tutte le intersezioni verticali. Questo significa che l'algoritmo, a partire da una delle lettere della parola orizzontale, scorre la griglia verso l'alto - mantenendo fissa la colonna di partenza - e si ferma quando trova uno spazio.

Successivamente legge l'intera parola verticale dall'inizio alla fine e ne calcola il punteggio, se questa è formata da più di una lettera.

Un algoritmo analogo funziona per le parole verticali che hanno, al contrario, intersezioni orizzontali.

Il programma fa uso di 4 thread separati [VV17f] per cercare in parallelo le parole. Due thread considerano le parole orizzontali, mentre altri due considerano le parole verticali. A loro volta i thread che cercano parole nella stessa direzione considerano parti diverse del tabellone di gioco. Le parti sono calcolate in base all'area occupata effettivamente da parole.

2.5.3 Mancanza di parole valide

In alcune situazioni particolari (e molto rare) non è possibile formare alcuna parola con le lettere a disposizione. Questa situazione si può verificare solo per parole successive alla prima, che al massimo può essere una combinazione di due lettere, sempre valida.

In questi casi particolari la ricerca della parola restituirà come punteggio massimo 0, di conseguenza il programma comunica all'utente attraverso una finestra modale l'impossibilità di comporre una parola con le lettere presenti sul leggio in quella determinata situazione.

3 NOTE SULLA REALIZZAZIONE

3.1 LIBRERIE UTILIZZATE

Per la realizzazione del programma ho utilizzato innanzitutto la libreria standard del C++ **iostream**, necessaria ad esempio per l'input e l'output del programma. Per l'importazione dei file del dizionario e delle regole, invece, ho utilizzato la libreria **fstream**.

Le strutture dati che ho utilizzato talvolta hanno richiesto l'utilizzo di ulteriori librerie, come nel caso di **vector** e di **map**.

L'estrazione casuale delle lettere ha richiesto invece l'utilizzo delle librerie **cstdlib** e **time.h**, in modo che la casualità sia basata sul tempo [VV17d].

Per alcune funzioni del programma (come la determinazione del massimo o la generazione di permutazioni) ho utilizzato la libreria **algorithm** e, infine, per la gestione dei thread ho utilizzato la libreria **thread**.

La gestione delle stringhe ha richiesto anche che alcune di queste fossero rese maiuscole o minuscole o che fossero riconosciute le parole con iniziali maiuscole nel dizionario, in modo da non considerarle, essendo nomi propri. Per svolgere queste operazioni ho utilizzato la librerie **cctype**.

In alcune situazioni, soprattutto per debuggare il programma, ho utilizzato delle funzioni per la stampa di dati (come nel caso della stampa del dizionario). In queste situazioni ho utilizzato la libreria **iomanip** per poter utilizzare la funzione *setw*.

Le librerie utilizzate fino ad ora quindi sono:

```
#include <iostream>
#include <fstream>
#include <vector>
#include <cstdlib>
#include <string>
#include <cctype>
#include <thread>
#include <iomanip>
#include <map>
#include <algorithm>
#include <time.h>
```

Sono state necessarie anche alcune librerie di Qt per la gestione dell'interfaccia grafica:

```
#include <QMessageBox>
#include <QPlainTextEdit>
#include <QInputDialog>
#include <QString>
```

3.2 ASPETTI DELLA CODIFICA DI RILIEVO

Il programma è stato suddiviso in varie parti, contenute in file di intestazione diversi.

Le classi sono state utilizzate soltanto per la gestione dell'interfaccia grafica in Qt.

3.2.1 Variabili globali

Il file di intestazione `Globali.h` contiene tutte le variabili globali, le definizioni di tipo e le istruzioni al preprocessore (`define` e `include`).

Ho deciso di utilizzare un unico file per raccogliere le variabili globali per garantire una migliore leggibilità.

All'interno del file ho cercato di raggruppare le variabili con funzioni simili (o utilizzate nella stessa parte del programma) per la stessa ragione.

3.2.2 Debug

La parte di **debug**, che non viene effettivamente utilizzata nel programma finale, è contenuta nel file `Debug.h`.

Le funzioni realizzate servono principalmente per la visualizzazione dei leggii, delle parole, dei punteggi, dei dizionari e delle regole.

Si tratta di funzioni che sono state utilizzate per verificare la correttezza dei dati presi in input e della loro elaborazione.

Alcune funzioni erano utilizzate per l'interazione con l'utente nella versione senza interfaccia grafica.

3.2.3 Dizionario

Il file di intestazione `Dizionario.h` contiene la maggior parte delle funzioni relative alla gestione dei dizionari e delle regole.

In particolare, le funzioni più importanti sono quelle per la lettura del dizionario, la lettura delle regole, l'elaborazione delle regole e la loro applicazione al dizionario di base.

Quasi tutte le funzioni indicate vengono richiamate non appena il programma si avvia, contemporaneamente alla richiesta del numero di giocatori.

Il tempo impiegato per la lettura del dizionario e le elaborazioni successive è nell'ordine di qualche secondo.

A differenza della prima versione del programma, inoltre, nella fase iniziale viene generata anche la matrice di vettori di iteratori di mappe di coppie di stringhe descritta in precedenza. Durante la fase di avvio, quindi, il programma impiega uno o due secondi in più rispetto alla versione precedente, ma questo garantisce la presenza di suggerimenti molto più rapidi.

3.2.4 Griglia

Il file di intestazione `Griglia.h` contiene tutto ciò che riguarda direttamente la gestione della griglia di parole, come la sua inizializzazione.

In questo file è contenuta anche la funzione per visualizzare graficamente i jolly. Questa funzione è relativa alla classe `FinestraPrincipale`, che si riferisce a tutte le caratteristiche dell'interfaccia grafica in Qt.

L'impostazione dei livelli della griglia non è manuale, ma segue un algoritmo. In questo modo la posizione dei moltiplicatori di lettera e di parola viene stabilita in rapporto alla grandezza di ogni livello della matrice, le cui dimensioni sono definite attraverso due variabili costanti, uno per la riga e l'altro per la colonna.

Ho pensato di realizzare la griglia in questo modo nel caso in cui in futuro dovesse servire realizzare una versione del gioco con dimensioni diverse rispetto a quelle standard (in tal caso, comunque, l'interfaccia grafica andrebbe ricostruita, almeno in parte).

3.2.5 Parole

Il file di intestazione `Parole.h` contiene alcune funzioni riguardanti i leggii (in particolare il loro riempimento e la copia su un leggio provvisorio, che serve durante i suggerimenti), una funzione che permette di verificare se una parola esiste (la ricerca diventa logaritmica, e quindi più efficiente rispetto a quella lineare), una funzione che restituisce il valore di una lettera, altre funzioni per il calcolo del punteggio di una singola parola (e delle sue intersezioni) e infine delle funzioni per l'inserimento di una parola sul tabellone di gioco[1].

1 In questa situazione l'utilizzo delle classi mi avrebbe permesso di avere una struttura più compatta e probabilmente più leggibile.

3.2.6 Sacchetto

Il file di intestazione `Sacchetto.h` si occupa della gestione del sacchetto delle lettere, ovvero della lettura delle lettere, della loro estrazione, del loro conteggio e del loro eventuale ordinamento.

Il sacchetto è di tipo **`vector di elemSacc`**, dove `elemSacc` è definito mediante una `struct` [sez. 2.2].

3.2.7 Turni

Il file di intestazione `Turni.h` contiene la maggior parte delle funzioni che riguardano la gestione dei leggii, dei jolly e della fine del gioco.

La funzione `FineGioco`, in particolare, verifica se il leggio corrente (ottenuto grazie ad un altra funzione) è formato da 8 spazi bianchi.

In tal caso viene applicato il regolamento e decretato il vincitore. La funzione comunica il risultato attraverso una finestra modale, il cui messaggio varia nel caso in cui vi siano più vincitori ex-aequo.

La funzione `FineGioco` viene richiamata al termine di ogni mossa del giocatore, ma solo dopo che il leggio è stato riempito (in caso contrario considererebbe il gioco finito anche nel caso in cui fossero presenti delle lettere nel sacchetto).

Dovendo mostrare una finestra modale questa funzione è definita all'interno della classe `FinestraPrincipale`.

3.2.8 Suggerimenti

Il file di intestazione `Suggerimenti.h` contiene tutte le funzioni che servono per la generazione di suggerimenti, sia nel caso in cui questi siano riferiti alla prima parola che nel caso in cui questi si riferiscano ad una parola successiva.

Le funzioni sviluppate in questo file si basano innanzitutto sull'utilizzo di `map` e `multimap`, oltre che della matrice di vettori di iteratori descritta in precedenza.

In particolare, esistono due funzioni distinte per il calcolo dei suggerimenti verticali e orizzontali. Ho fatto questa scelta per poter utilizzare quattro thread in parallelo su due aree distinte della matrice.

Nel momento in cui queste funzioni vengono richiamate da un'altra funzione, chiamata `SuggerimentiGenerici`, non è necessario copiare il dizionario considerato (che sia quello di base o quello completo) per quattro volte, infatti questo viene solamente letto [Sta13]. Analogamente, anche la griglia non deve essere copiata quattro volte.

Vi sono poi quattro variabili di tipo `DatiParola` [sez. 2.1] che vengono utilizzate rispettivamente per ognuno dei quattro thread, infatti queste vengono modificate all'interno dei thread.

La ricerca avviene suddividendo la griglia in regioni della stessa ampiezza. Le regioni considerate vengono calcolate in base alla posi-

zione delle lettere già inserite, in modo che le quattro regioni non contengano celle tali da essere contemporaneamente su righe e colonne vuote.

Naturalmente l'utilizzo dei thread non riduce la complessità dell'algoritmo, ma permette di calcolare contemporaneamente più dati, distribuendoli su quattro thread (e non più su uno solo).

La definizione dei thread all'interno della funzione è la seguente:

```
thread OP(SuggerimentiGenericiOrizzontali, Leggio, rigamin,
    rigamed, std::ref(OrP));
thread OS(SuggerimentiGenericiOrizzontali, Leggio, rigamed,
    rigamax+1, std::ref(OrS));
thread VP(SuggerimentiGenericiVerticali, Leggio, colonnamin,
     colonnamed, std::ref(VerP));
thread VS(SuggerimentiGenericiVerticali, Leggio, colonnamed,
     colonnamax+1, std::ref(VerS));

OP.join(); OS.join(); VP.join(); VS.join();
```

`OrP`, `OrS`, `VerP` e `VerS` sono dei dati di tipo `DatiParola` e servono per memorizzare la parola col punteggio maggiore per ogni thread.

`SuggerimentiGenericiOrizzontali` e `SuggerimentiGenericiVerticali` sono i nomi delle funzioni considerate, mentre l'utilizzo di `std::ref` è necessario per passare le variabili per riferimento (nel caso in cui vengano utilizzati i thread).

3.2.9 Altre funzioni e interfaccia grafica

All'interno del programma vi sono poi altre funzioni (le principali sono descritte nella *Progettazione della soluzione*) legate all'interfaccia grafica.

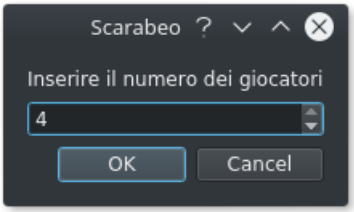
In particolare sono presenti delle funzioni per la visualizzazione degli scarabei, la gestione delle finestre modali per l'interazione con l'utente e l'interazione tra l'interfaccia grafica e le funzioni descritte fino ad ora.

L'interfaccia grafica è stata realizzata utilizzando Qt.

4 PROVE DI TEST

4.1 NUMERO DI GIOCATORI

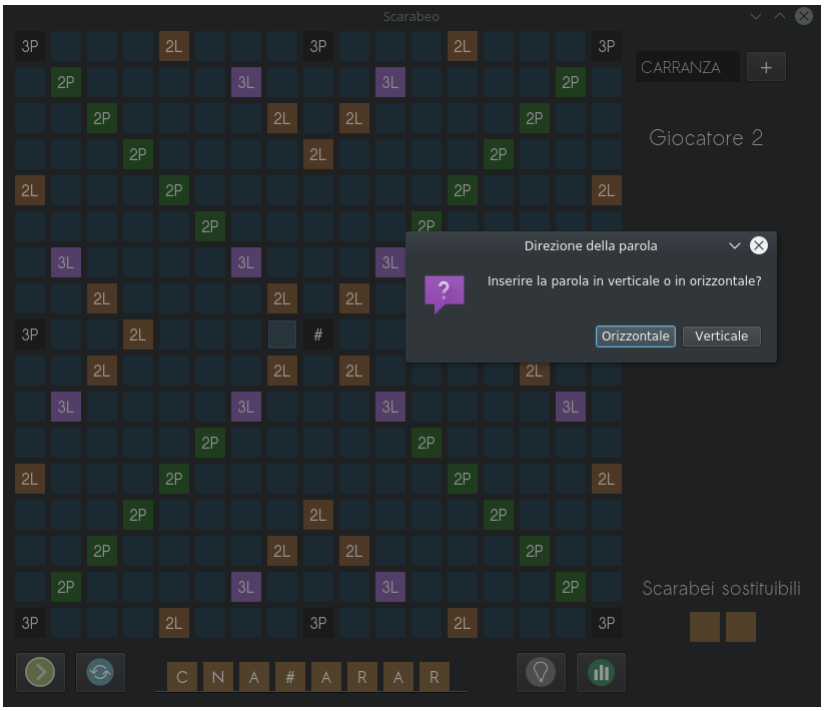
All'inizio del gioco è possibile scegliere il numero di giocatori, variabile tra 2 e 4, come mostrato nella sez. 1.3.



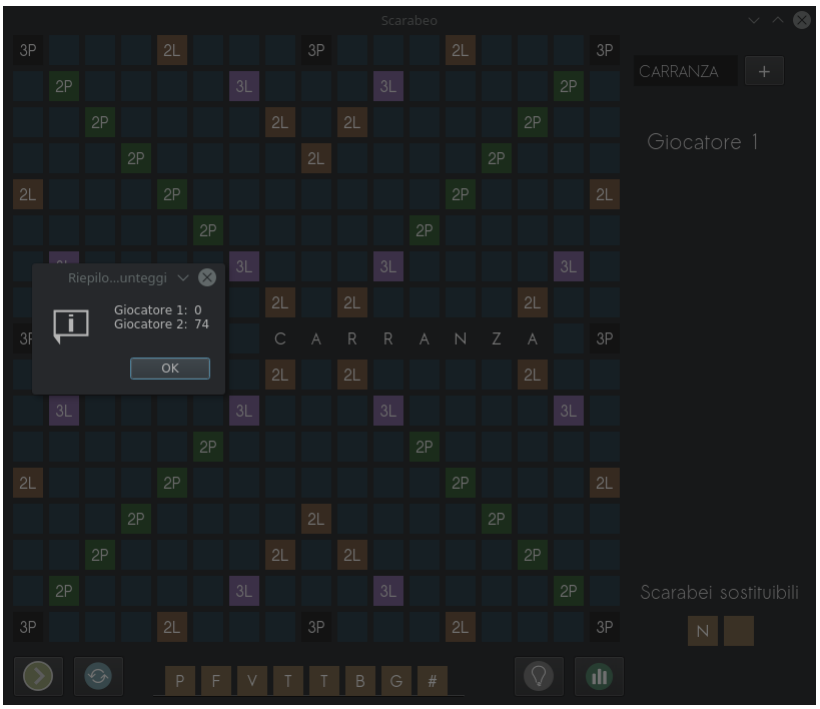
4.2 INSERIMENTO DELLA PRIMA PAROLA

L'inserimento della prima parola può avvenire manualmente o chiedendo un suggerimento [sez. 2.5.1].

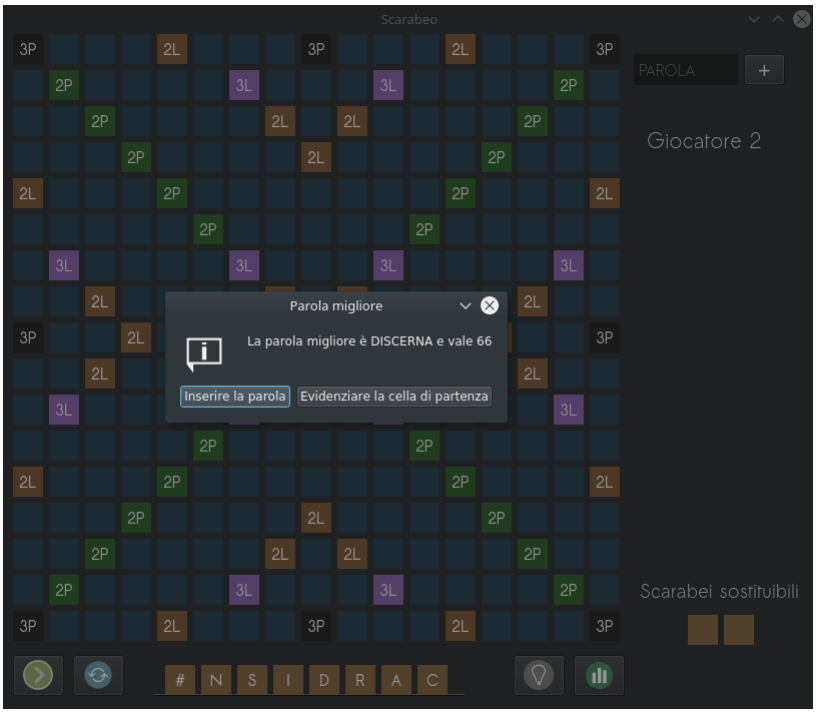
Nel primo caso la schermata risultante sarà la seguente:



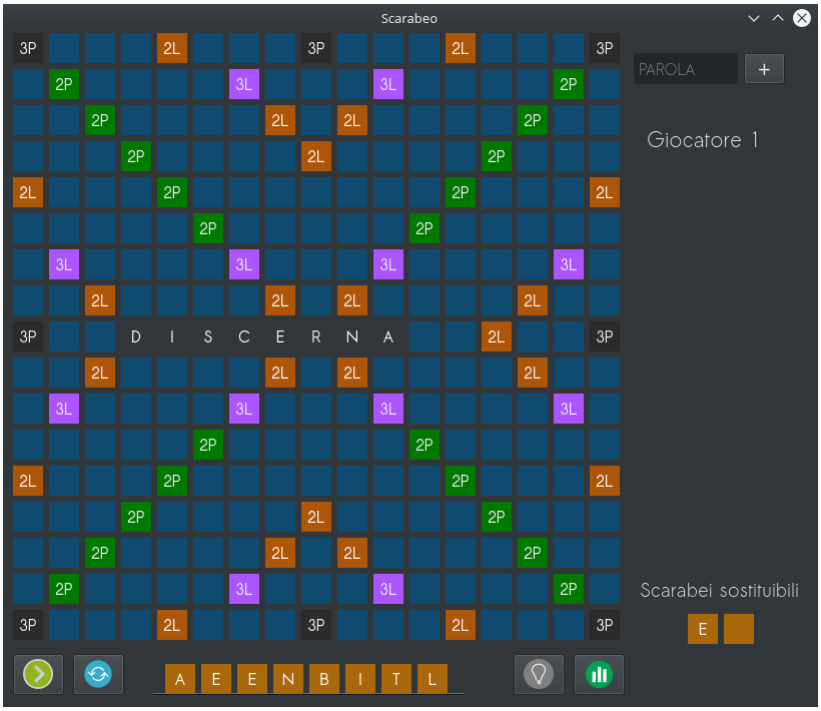
In questo caso la parola utilizza anche un jolly, che verrà aggiunto alla lista degli scarabei sostituibili.



L'inserimento automatico, invece, avviene utilizzando il tasto con l'icona della lampadina o con la combinazione Ctrl+H. La ricerca avviene direttamente nel dizionario completo [sez. 2.5.1]. Anche in questo caso il leggio contiene un jolly.

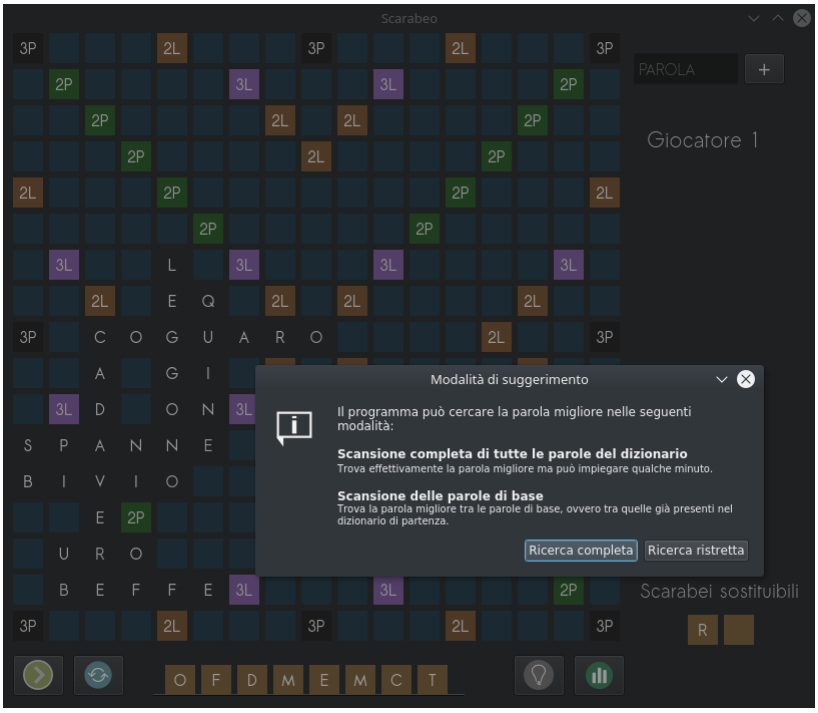


Scegliendo di inserire la parola il risultato è il seguente:



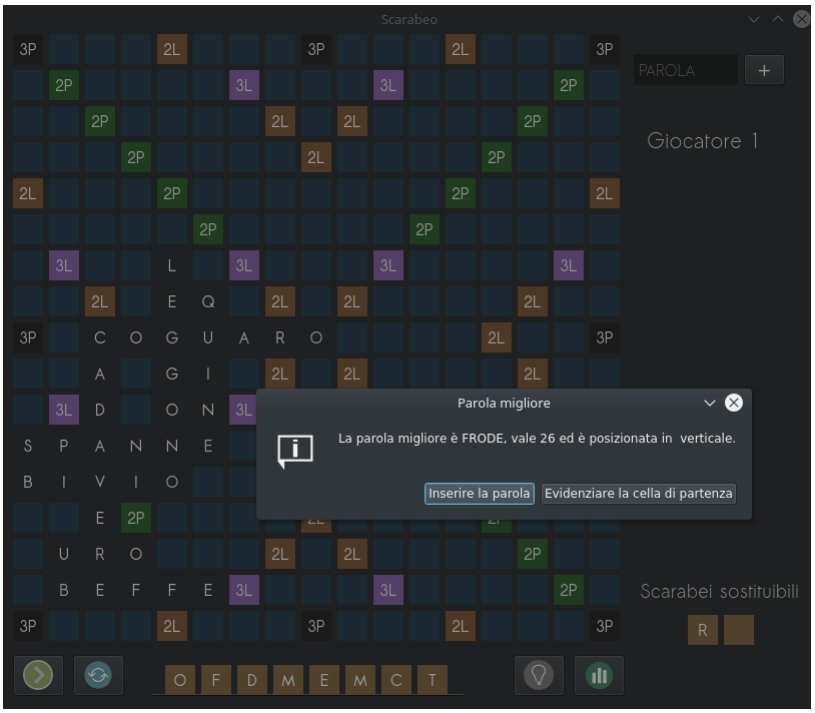
4.3 INSERIMENTO DI ALTRE PAROLE

Anche in questo caso l'inserimento può essere o meno manuale. Il primo esempio mostra la situazione in cui non lo è.

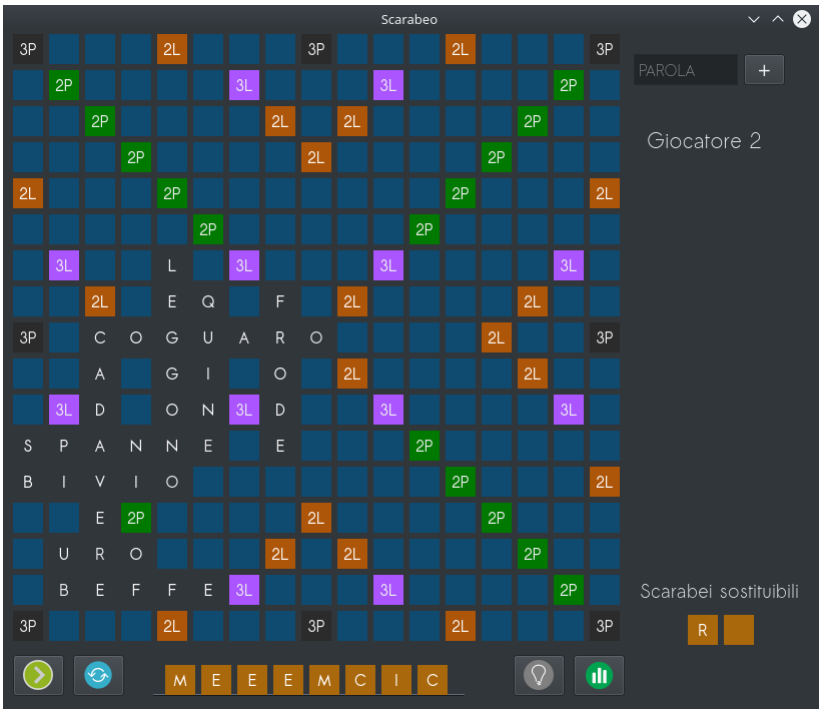


La finestra riporta delle informazioni per permettere all'utente di distinguere tra i due tipi di ricerca.

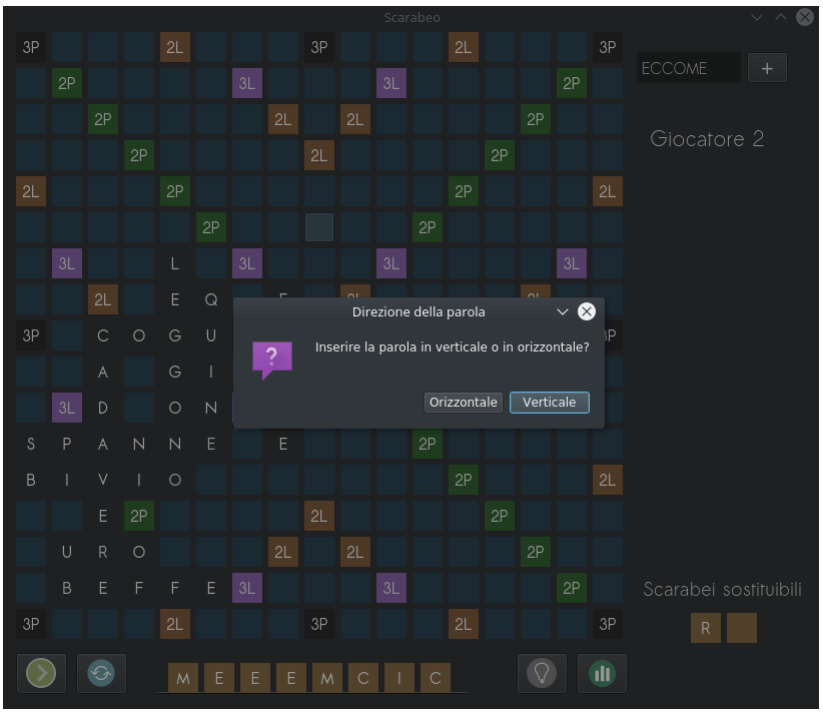
Scegliendo la ricerca completa la parola trovata viene visualizzata come nel caso della prima parola:



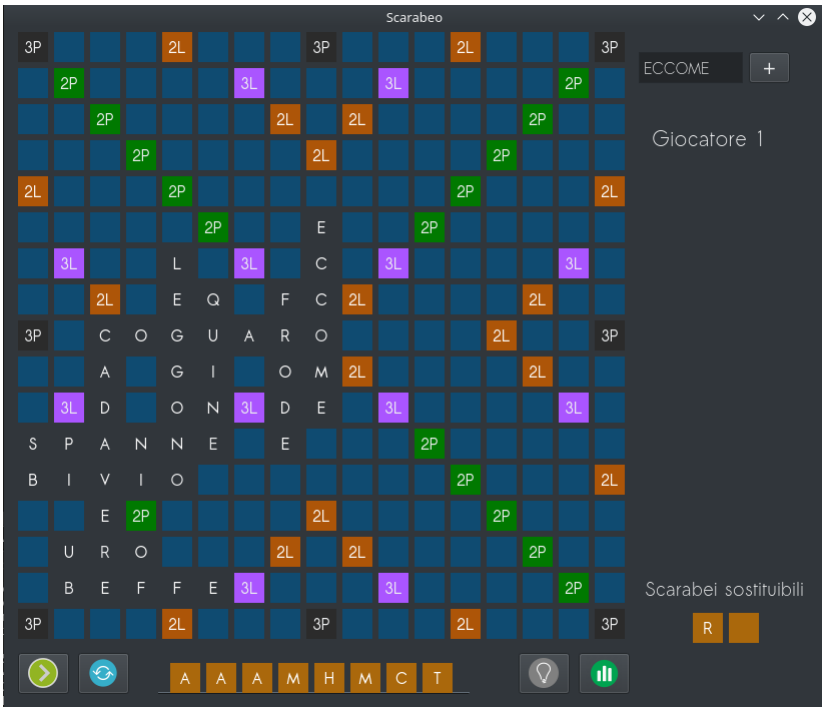
Anche in questo caso è possibile inserire la parola indicata:



L'inserimento manuale avviene analogamente a quello mostrato per la prima parola inserita.

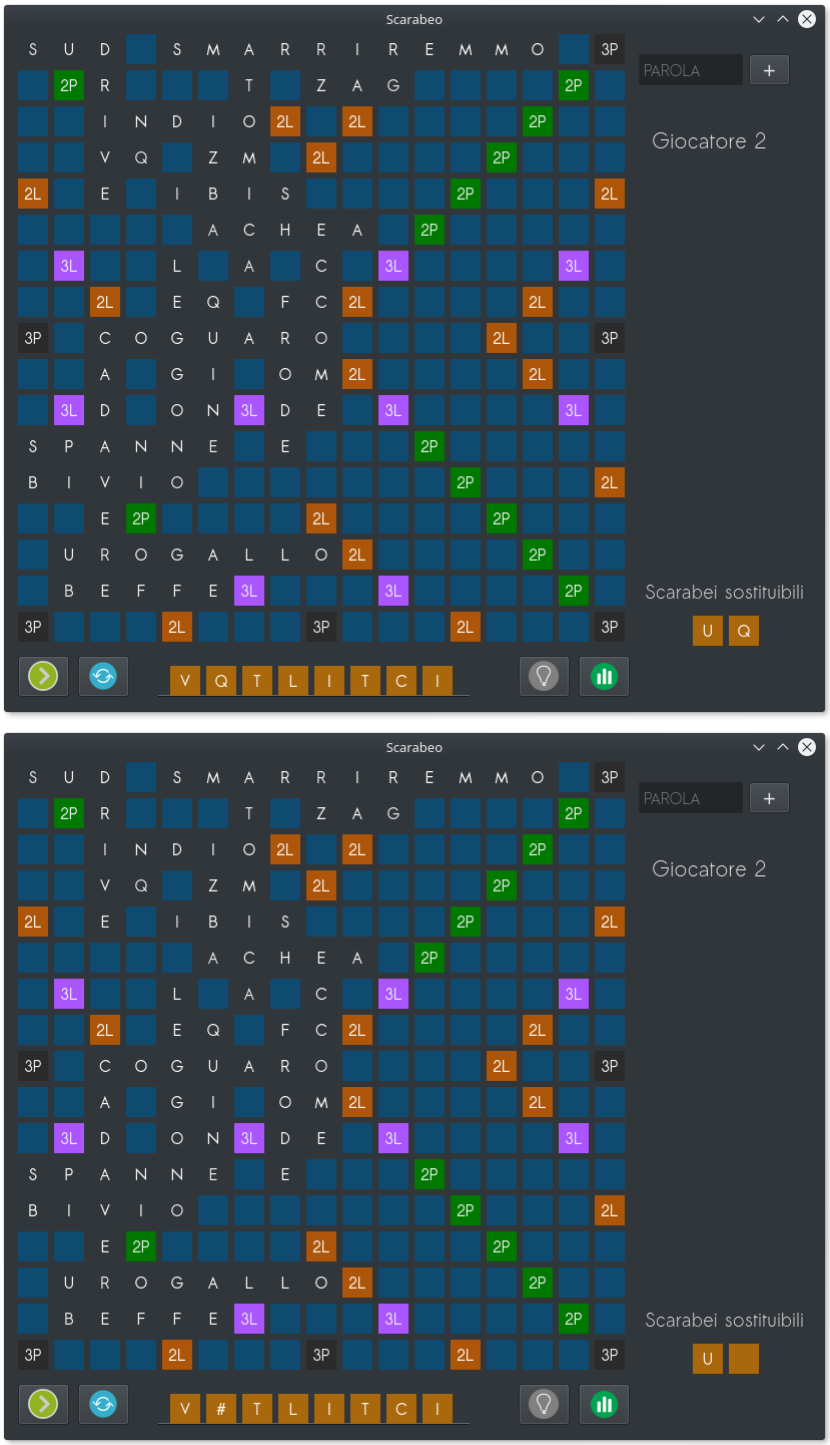


Scegliendo di inserire la parola, quindi, risulta:



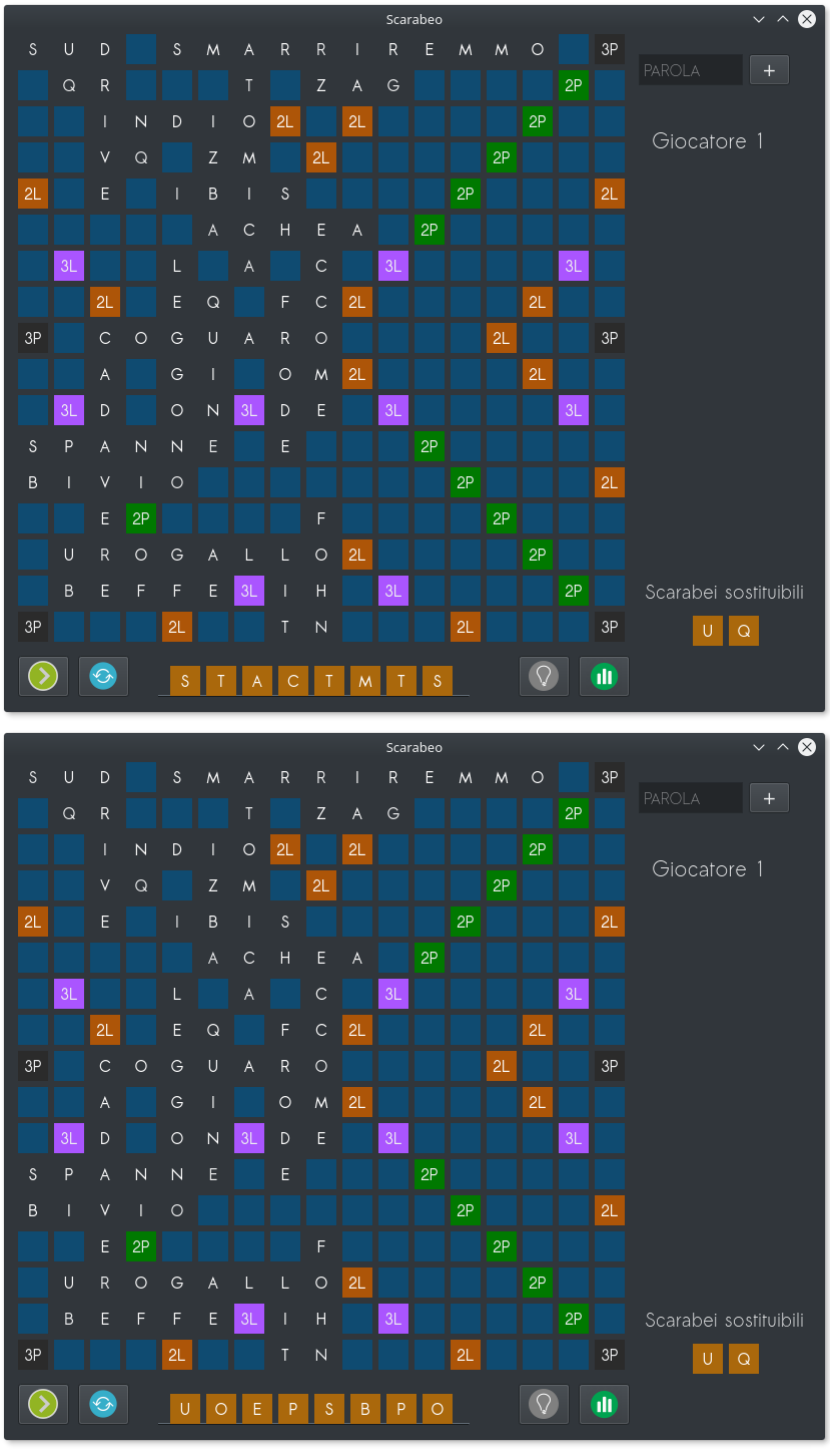
4.4 SOSTITUZIONE DEI JOLLY

Se il giocatore di turno ha una lettera che corrisponde ad uno dei jolly (indicati nella sezione *Scarabei sostituibili*) ha la possibilità di sostituire la propria lettera al jolly facendo click sul jolly. Nell'esempio seguente viene sostituita la lettera Q.



4.5 CAMBIO DI TUTTE LE LETTERE

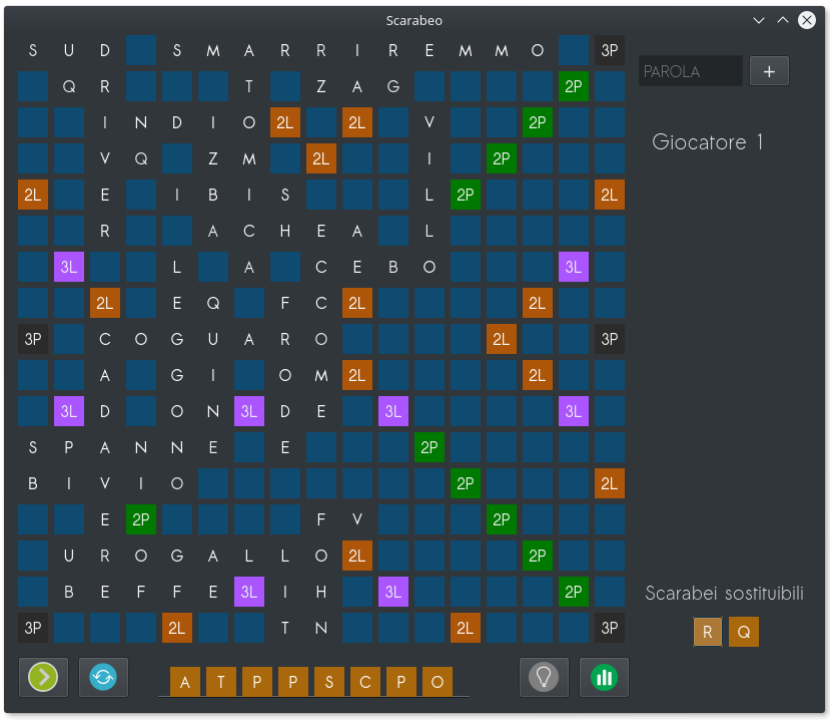
Il giocatore di turno ha la possibilità di cambiare tutte le proprie lettere utilizzando il secondo pulsante da sinistra. In questo caso il turno viene passato al giocatore successivo[1].

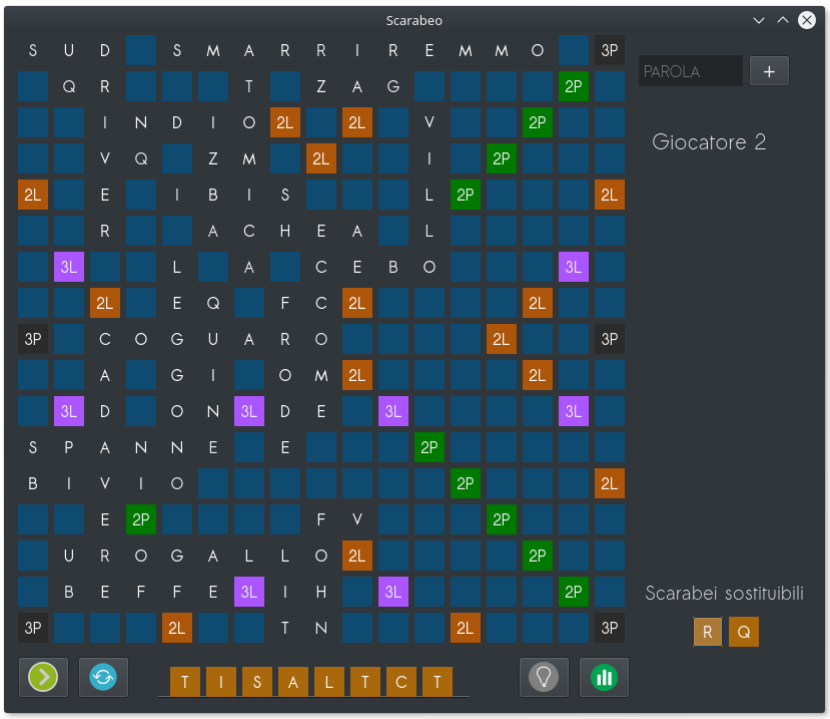


1 Nell'esempio viene mostrato il risultato del cambio di lettere prima che avvenga il passaggio al giocatore successivo

4.6 PASSAGGIO DEL TURNO

Il giocatore di turno ha la possibilità di passare il turno senza cambiare le proprie lettere.

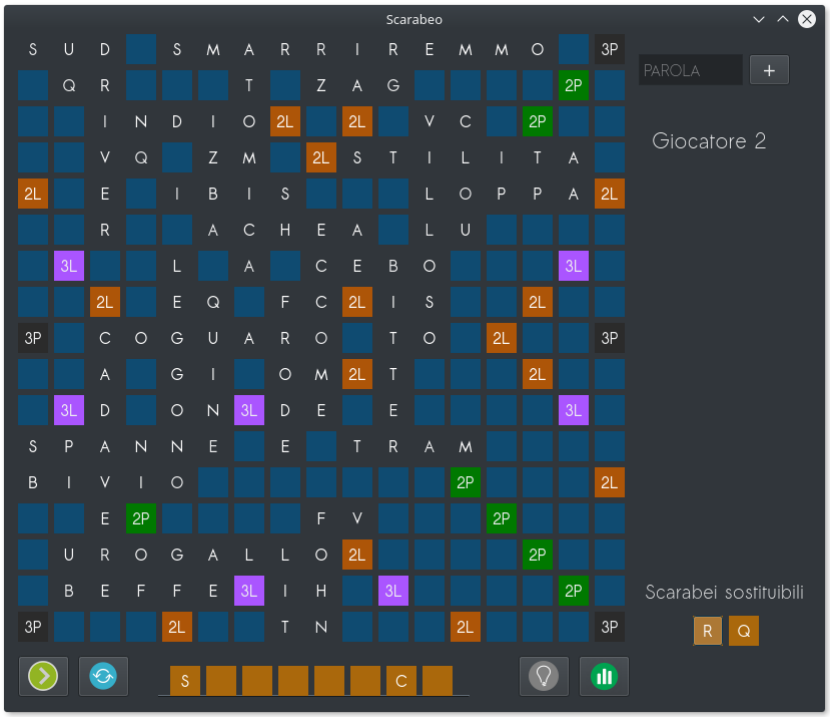


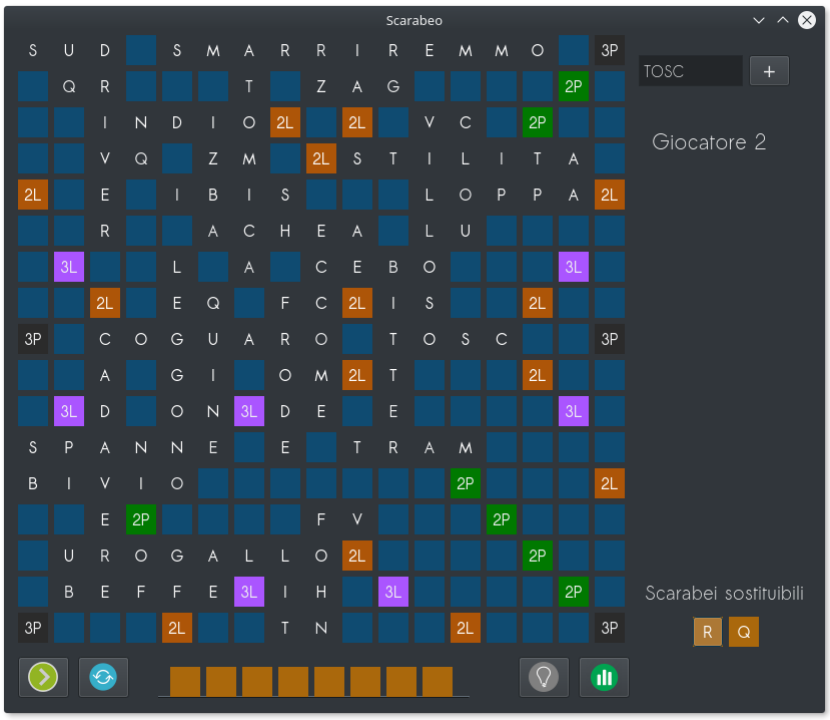


Il passaggio avviene utilizzando il primo pulsante da sinistra.

4.7 FINE DELLA PARTITA

La partita termina quando uno dei giocatori rimane senza lettere sul proprio leggio e il sacchetto è vuoto. Il giocatore che finisce le proprie lettere aggiunge al suo punteggio la somma dei punti delle lettere che gli altri giocatori hanno sui propri leggii.





BIBLIOGRAFIA

[LLM13] Stanley Lippman, Josée Lajoie e Barbara Moo. *C++ Primer, 5th edition*. 2013.

[Sta10] Comunità di Stackoverflow. *Conversione da int a QString*. 2010. URL: http://stackoverflow.com/questions/3211771/how-to-convert-int-to-qstring.

[Sta13] Comunità di StackOverflow. *Is it wise to access read-only data from multiple threads simultaneously?* 2013. URL: http://stackoverflow.com/questions/5643060/is-it-wise-to-access-read-only-data-from-multiple-threads-simultaneously.

[Str98] Bjarne Stroustrup. *The C++ programming language, third edition*. 1998.

[VV17a] AA. VV. *C++ Reference: mappe*. 2017. URL: http://www.cplusplus.com/reference/map/map/.

[VV17b] AA. VV. *C++ Reference: multimappe*. 2017. URL: http://www.cplusplus.com/reference/map/multimap/.

[VV17c] AA. VV. *Converting Affix Files: Understanding the Affix File Format*. 2017. URL: http://www.openoffice.org/lingucomponent/affix.readme.

[VV17d] AA. VV. *Generate random numbers*. 2017. URL: http://www.cplusplus.com/reference/cstdlib/rand/.

[VV17e] AA. VV. *QString Class*. 2017. URL: http://doc.qt.io/qt-5/qstring.html.

[VV17f] AA. VV. *Thread*. 2017. URL: http://www.cplusplus.com/reference/thread/thread/.