UPWORDS

Corso di Ingegneria Informatica

Litkovskyy Valeriy, Kazepov Alexander

July 14, 2018

Contents

1 Introduzione

Si vuole realizzare un'applicazione che permette agli utenti di giocare al gioco Upwords, una variante dello Scarabeo classico. Qui di seguito verranno riportate le specifiche per la creazione del programma:

- Numero dei giocatori: da 2 a 4
- Dimensione griglia richiesta: 10x10
- Dimensione *mano* giocatore: 7 caselle

A differenza dello scarabeo classico le lettere possono essere sovrapposte verticalmente, fino a 5 livelli.

Le parole verranno formate dai giocatori con le lettere presenti nella loro *mano*, le quali verranno estratte da un *sacchetto*.

Le lettere sono presenti in queste quantità:

Lettere	Quantità
O	15
A	14
I	12
E	11
C R S T	6
L N M U	5
B D F P V	3
G H Z	2
Q	1

Il programma deve controllare la validità delle parole di lunghezza superiore a 3 verificandone la presenza in un dizionario della lingua italiana (rappresentato da un file in formato `txt`).

1.1 Regole principali

- La prima parola inserita deve passare per il centro della griglia

- Ogni giocatore deve poter compiere una delle seguenti azioni:
  - Inserire una parola (valida)
  - Passare il turno
  - Cambiare una lettera
  - Chiedere un eventuale suggerimento da parte del programma
- Le parole possono essere inserite solo orizzontalmente o verticalmente
- Si possono modificare le parole presenti sulla griglia per crearne una nuova
- La partita termina se: tutti i giocatori passano il turno consecutivamente o se sono state usate tutte le lettere nel *sacchetto*.

2 Progettazione della soluzione

Per la realizzazione del progetto il complesso problema del gioco è stato diviso in sotto-problemi.

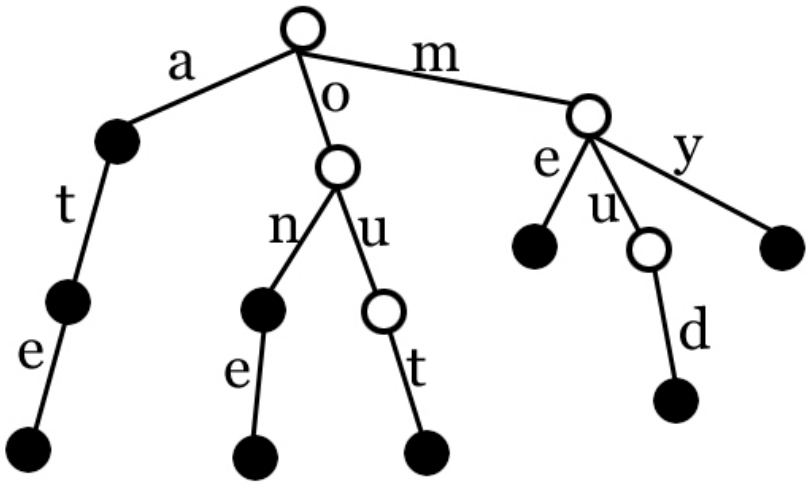
2.1 La scelta delle strutture dati

Il primo passo, ovviamente è stato la scelta delle strutture dati destinate al contenimento dei *dati principali* del gioco. Ognuna di queste strutture è stata definita nella libreria `data_structs_n_constants.h` per rendere il codice sempre consistente.

2.1.1 I Trie e il vocabolario

Data la grandezza del file contenente il vocabolario (contenente anche suffissi ecc) si è optato per i Trie.

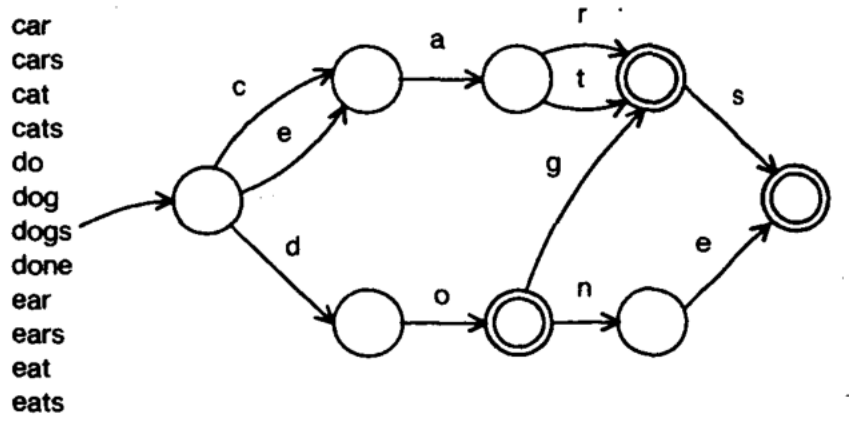
Un Trie è un tipo di struttura ad albero n-ario ordinato usato per rappresentare un set di stringhe ove ogni nodo contiene una lettera.



Dove ogni nodo è definito come:

```
struct Tnode {
    char letter;                // lettera del nodo
    bool is_end;                // nodo terminale?
    vector <Tnode*> Tchildren;  // figli del nodo
};
```

Con la scelta dei Trie si è optato per un ottimo rapporto tra ottimizzazione e semplicità del codice. Il Trie infatti non è la struttura migliore in questo caso (la struttura dati più efficente sarebbe il Dawg - Direct Acyclic Word Graph) ma la sua gestione rende il codice molto più semplice da capire rispetto a quello di altre strutture dati. Senza contare che il guadagno sarebbe solo dal punto di vista della memoria usata (circa 1/3).



Per la gestione della struttura dati è stata creata un'apposita libreria nel file `trie_manager.cpp` che verrà descritto in seguito.

2.1.2 Altre strutture dati e variabili secondarie

```
#define HORIZONTAL true
#define VERTICAL false
```

Costanti necessarie per cambiare la direzione dell'inserimento

```
int BOARD_SIZE = 10,
    PLAYER_HAND = 7;
```

Variabili necessarie per rendere disponibile la possibilità di cambiare la dimensione del bordo (e della *mano*) senza andare a modificare il codice. Infatti, ad ogni funzione che necessitava di questi dati, sono state passate queste due variabili, che possono essere soprascritte attraverso le impostazioni presenti nel menù del programma.

```
struct Player {
    string name;                // Nome del giocatore
    vector <char> letters;      // Lettere che ha in "mano"
    int points;                 // Punteggio attuale
    bool passed;                // Il giocatore ha passato il turno
};
```

La struttura che definisce un giocatore.

2.2 La scelta dell'interfaccia grafica

La scelta dell'interfaccia è stata tutt'altro che semplice, ma alla fine abbiamo optato per una TUI (Text - Based User Interface). la scelta è stata influenzata dai sistemi operativi utilizzati da noi sviluppatori (Manjaro Linux e Mac Os, entrambi basati su Unix). Questa scelta rende possibile la completa esecuzione del programma (e della sua interfaccia) attraverso il terminale utilizzato. Ciò non rende solamente il gioco utilizzabile anche per chi non ha un ambiente grafico ma diminuisce anche notevolmente il carico computazionale del programma. Mantenendo un interfaccia intuitiva e *geeky* per l'utente. Per la realizzazione è stata usata interamente la libreria ncurses (new curses) che attraverso delle API permette al programmatore di scrivere TUI indipendenti dal terminale di esecuzione. Alcuni dei programmi più famosi che implementano ncurses sono:

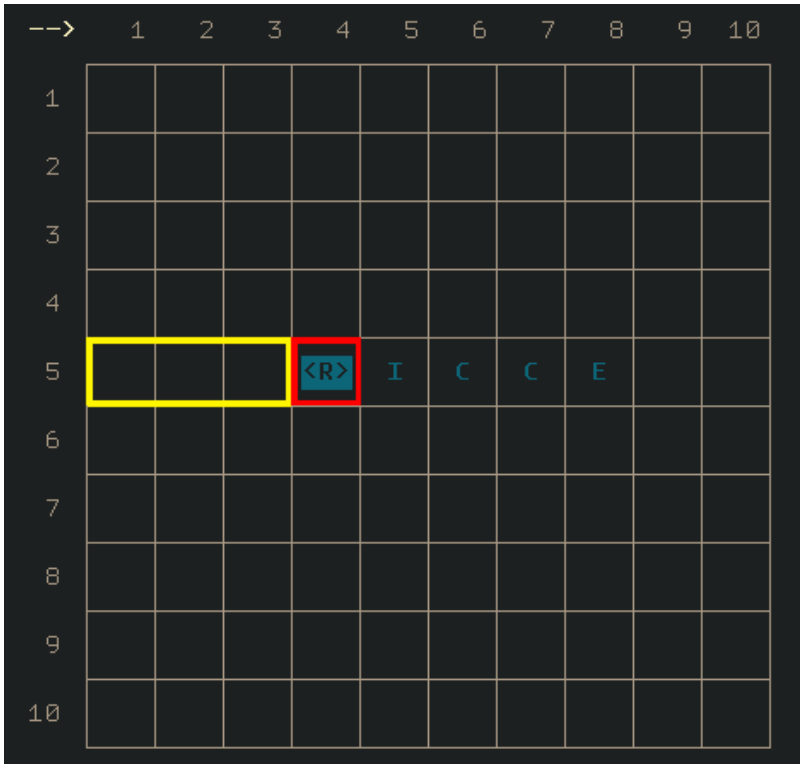
- Midnight commander
- Htop
- Ranger

Ncurses non fa altro che creare monitor fittizi i cui elementi sono interamente modificabili. Per questo motivo è possibile personalizzare molto di più l'interfaccia rispetto a una normale GUI. Alcuni esempi sono riportati nella sezione **4**.

2.3 La scelta dell'algoritmo per il suggerimento

Per capire meglio i seguenti algoritmi è necessario introdurre delle nozioni:

- controlli incrociati (cross-checks): per ogni casella vegnono precomputate tutte le lettere che possono essere inserite in questa casella. Esempio: Se sopra e sotto una casella non ci sono delle lettere, possiamo inserire qualsiasi lettera dell'alfabeto. Invece, se ci sono delle lettere, possiamo inserire solo quelle che sono presenti nel ramo della Trie.
- ancora (anchor): un'ancora è la casella che contiene la lettera più a sinistra di una parola (rosso in figura)
- limite (limit): ogni limite è associato ad un ancora ed indica le caselle vuote alla sua sinistra (giallo in figura). Rappresenta lo spazio del quale può essere estesa la parola a sinistra.



Siccome le parole possono essere inserito solo andando verso il basso o verso la destra, si può ridurre il problema fino ad una sola dimensione usando la trasposta del bordo. Quindi, computando prima tutti i cross-checks per una riga, possiamo cercare tutte lo parole incrociate in quella riga senza considerare i contenuti del resto del bordo. La ricerca del suggerimento si riduce al seguente problema unidimensionale: date le lettere della mano del giocatore, i contenuti della riga del bordo, i cross-checks della riga e le ancore, generare tutte le mosse valide per quella riga.

Per generare le mosse si usa un algoritmo composto da due parti. Per ogni ancora, si generano tutte le mosse valide in seguente modo:

1. Trovare tutte le possibili parti a sinistra dell'ancora
2. Per ogni parte sinistra trovata, trovare tutte le parti a destra valide
3. Specificare le parele trovate come "valide"

Per ognuno di questi passi si sviluppa una funzione

```
    ParteSinistra(‘ParolaParziale, nodo ‘N in Trie, ‘limite) =
      EstendiDestra(‘ParolaParziale, ‘N, ‘ancora)
      Se ‘limite > 0 allora
        Per ogni figlio ‘E del nodo ‘N
          Se la lettera ‘l rappresentata dal figlio ‘E è
              presente nella nostra "mano" allora
            Togliere la lettera ‘l dalla "mano"
            Sia ‘N’ il nodo raggiunto dal figlio ‘E
            ParteSinistra(‘ParolaParziale . ‘l, ‘N’, ‘limit - 1)
            Rimetti la lettera ‘l indietro nella "mano"

    EstendiDestra(‘ParolaParziale, nodo ‘N nella Trie, ‘casella) =
      Se ‘casella è vuota allora
        Se ‘N è un nodo terminale allora
          ParolaValida(‘ParolaParziale)
Label:  Per ogni figlio ‘E del nodo ‘N
          Se la lettera ‘l rappresentata dal figlio ‘E è
              presente nella nostra "mano"
              e
              ‘l è presente nel ‘cross-check della ‘casella allora
            Togliere la lettera ‘l dalla "mano"
            Sia ‘N’ il nodo raggiunto dal figlio ‘E
            Sia ‘prossima-casella la casella a destra dalla ‘casella
            EstendiDestra(‘ParolaParziale . ‘l, ‘N’, ‘prossima-casella)
            Rimetti la lettera ‘l indietro nella "mano"
      altrimenti
        Sia ‘l la lettera che occupa la ‘casella
        Se ‘N ha un figlio rappresentato dalla lettera ‘l
            che porta ad un nodo ‘N’ allora
          Sia ‘prossima-casella la casella a destra della ‘casella
          EstendiDestra(‘ParolaParziale . ‘l, ‘N’, ‘prossima-casella)
          Goto Label: // Caso upword
```

Ovviamente queste due funzioni trovano i possibili suggerimenti solo per un ancora. Quindi serve una funzione che, per ogni direzione chiami punteggio massimo, e lo stampiamo con coordinate e direzione.

È un algoritmo molto veloce perchè i controlli vengono eseguiti solo sulle parole che sono valide, senza considerare tutto il resto delle parole generate casualmente.

3 Note sulla realizzazione

3.1 Strumenti utilizzati

3.1.1 Editor di testo/IDE

Gli editor di testo utilizzati sono stati: Vim ed Emacs

3.1.2 Compilatore

Siccome il programma è stato scritto in un'ambiente basato su Unix, si è usato il **gcc** (Gnu Compiler Collection). Sono state eseguite delle prove solo sotto i sistemi operativi Linux e MacOS.

3.1.3 Build system e project management

È stato usato semplicemente il **make**. Si è scritto un **Makefile** con i seguenti comandi:

```
# compilare il programma con delle informazioni di debug gdb-friendly
# e senza ottimizzazioni. Utile per il dubug
make debug
# compilare il programma con ottimizzazioni per il l'uso finale
make release
# pulire gli eseguibili
make clean
```

3.2 Argomenti di riga di comando

Il programma accetta degli argomenti da terminale:

3.2.1 `-h` flag

Visualizza un messaggio d'aiuto suigli argomenti di riga di comando

3.2.2 `-d` flag

Se l'utente vuole utilizzare un dizionario diverso da quello fornito (`dictionary.txt`) si può usare `-d` flag per specificare il file di dizionario

3.3 Struttura del codice

Il codice è strutturato in modo tale da suddividere ogni funzionalità principale in un proprio file. Abbiamo cercato di seguire uno stile simile a quello funzionale per rendere il codice ancora più elegante.

Segue una lista e breve descrizione di ogni file e alcune delle sue proprietà.

3.3.1 `main.cpp`

Il main contiene la funzionalità principale del programma. Si occupa principalmente:

- del menù principale
- dell'inizializzazione delle strutture dati

  ```
  make_board()
  make_bucket()
  intialize_players()
  make_dictionary()
  ```

- dell'interazione base tra l'utente e l'interfaccia (attraverso delle librerie create)

  ```
  check_n_insert()
  ask_word_insertion()
  accept_players()
  ```

- gestire gli argementi della riga di comando

  ```
  parse_arguments()
  ```

- gestire la struttura del gioco

  ```
  game_loop()
  start_game()
  player_play()
  ```

3.3.2 `game_manager.cpp`

Il file `game_manager.cpp` contiene varie funzioni di supporto del gioco

- gestione del tavolo di gioco (board)

  ```
  make_board()
  destory_board()
  ```

- gestione della direzione dell'inserimento

  ```
  toggle_w_direction()
  ```

- gestione del sacchetto (bucket)

  ```
  make_bucket()
  destroy_bucket()
  ```

- gestione di ogni giocatore

  ```
  make_players()
  destroy_players()
  ```

- gestione delle lettere di ogni giocatore

  ```
  get_letters()
  exchange_letter()
  ```

- controllo della parola inserita

  ```
  check_word()
  insert_word_to_board()
  ```

- controllo delle condizioni di uscita ($\text{game}_{\text{over}}$)

  ```
  is_game_over()
  ```

3.3.3 `data_structs_n_constants.h`

Libreria già descritta in precedenza

3.3.4 `trie_manager.cpp`

Questo file si occupa interamente della gestione del Trie e della creazione del dizionario.

- inserire ogni elemento del dizionario nel Trie

  ```
  create_trie()
  insert_word()
  ```

- cercare le parole all'interno del Trie

  ```
  search_word()
  check_prefix()
  ```

- distruggere il Trie

  ```
  delete_trie()
  ```

3.3.5 `tui_manager.cpp`

Il file si occupa della gestione dell'interfaccia utente. Si occupa principalmente del calcolo delle dimensioni ottimali delle finestre (in base agli elementi che le compongono) rendendole adattive. Gran parte delle sue funzioni serve a gestire la dimensione ottimale del terminale.

Le funzioni relative all'UI non verranno descritte in questo documenti in quanto si basano quasi interamente su calcoli e non algoritmi interessanti.

Inoltre contiene le tre funzioni di interazione con l'utente

```
// Per visualizzare i messaggi in un messagebox
show_message()
// Una piccola finestra che permette all'utente di inserire i date
get_input()
// Permette all'utente di scegliere un elemento di una lista
// interattiva
show_menu()
```

3.3.6 `tui_helper.cpp`

Il file contiene delle funzioni d'appoggio per `tui_manager.cpp`. Si occupa semplicemente della stampatura a schermo degli aggiornamenti delle finestre all'interno della TUI

3.3.7 `suggestions.cpp`

In questo file saranno raccolte tutte le funzioni relative alla ricerca del suggerimento con punteggio maggiore.

3.4 Altri punti degni di nota

Una delle caratteristiche di questo programma è il vocabolario contenente i suffissi. Per semplificare i calcoli da parte del programma si è generato il vocabolario (fornito dai file della cartella su beep) con l'aiuto del comando:

```
# unmunch fa parte del paccketto hunspell
unmunch it_IT.dic it_IT.aff | grep -E -e "^[a-zA-Z]+$" > our_new_dict.txt
```

Un'altra funzione degna di nota di questo programma è:

```
transpose()
```

Questa funzione si occupa semplicemente di fare la trasposta del tavolo di gioco, invertendo le i con le j. Ciò semplifica notevolmente gli algoritmi di controllo e sopratutto di selezione del suggerimento.

Un'altra funzione importante è quella che si occupa del controllo di una parola e del calcolo del suo punteggio:

```
insert_word_to_board()
```

È senza dubbio la funzione più complessa di tutto il programma. Vengono effettuati i seguenti controlli e le rispettive azioni:

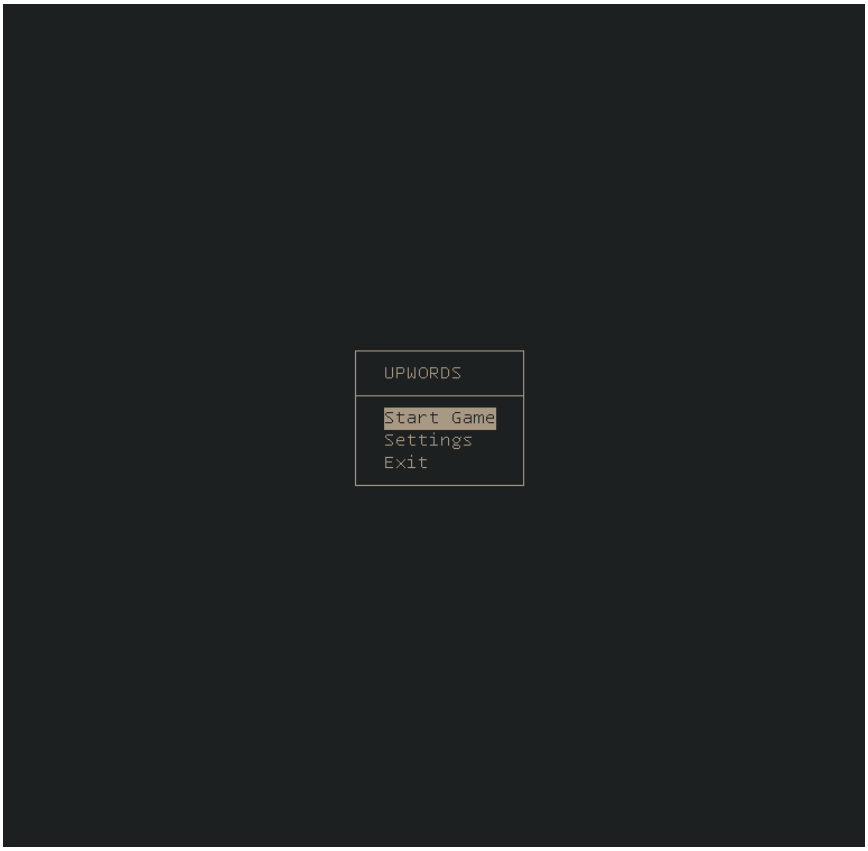
Controllo	Azione
ho la lettera e la casella è vuota	metti la lettera
ho la lettera e sulla casella c'è la stessa lettera	non cambia niente
ho la lettera e sulla casella c'è una lettera diversa e il livella è < 5	metti la lettera e aumenta il livello
non ho la lettera ed essa è presente sulla casella	niente cambia
sia io che la casella siamo senza la lettera richiesta	non posso mettere la parola
non ho la lettera e sulla casella c'è una lettera diversa	non posso mettere la parola
ho la lettera e sulla casella c'è una lettera diversa con livello $>= 5$	non posso mettere la lettera
le caselle adiacenti non sono vuote	effettuare un cross-check
se la parola incrociata esiste	metti la parola
se la parola incrociata non esiste	non metti la parola

La funzione si occupa anche dell'assegnamento dei punti. I punti vengono assegnati secondo i seguenti criteri:

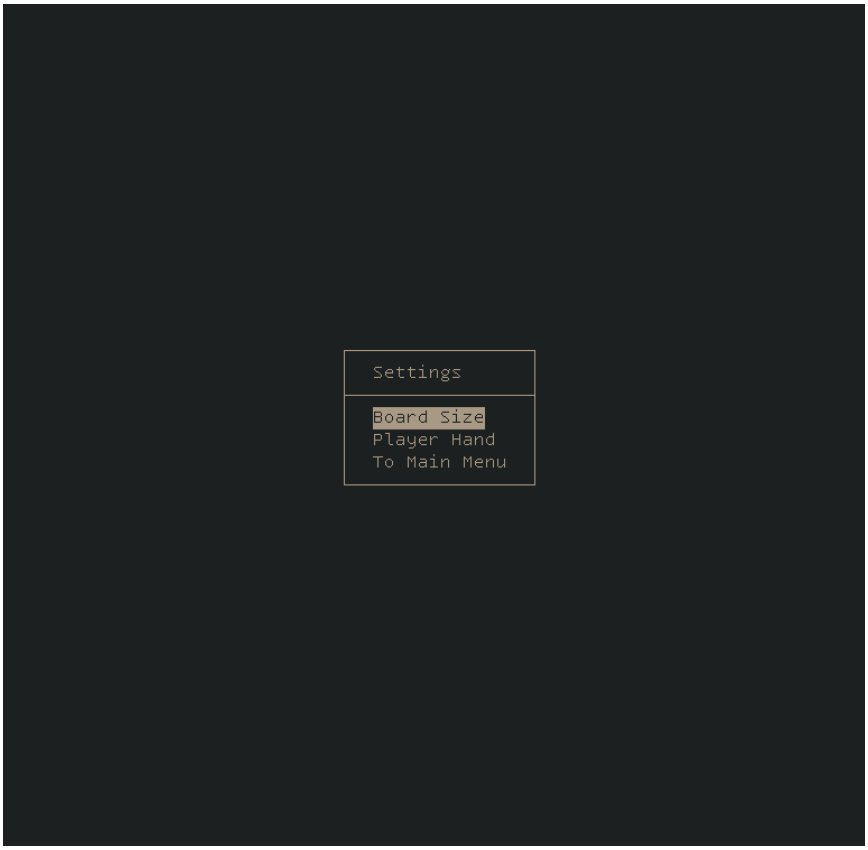
Criterio	Punteggio
ogni lettera normale in condizioni normali	+2
ogni incrocio con lettera normale	+3
ogni upword	+1
ogni upword con incrocio	+2
se tutte le lettera in mano vengono usate	+20

4 Prove di test (immagini)

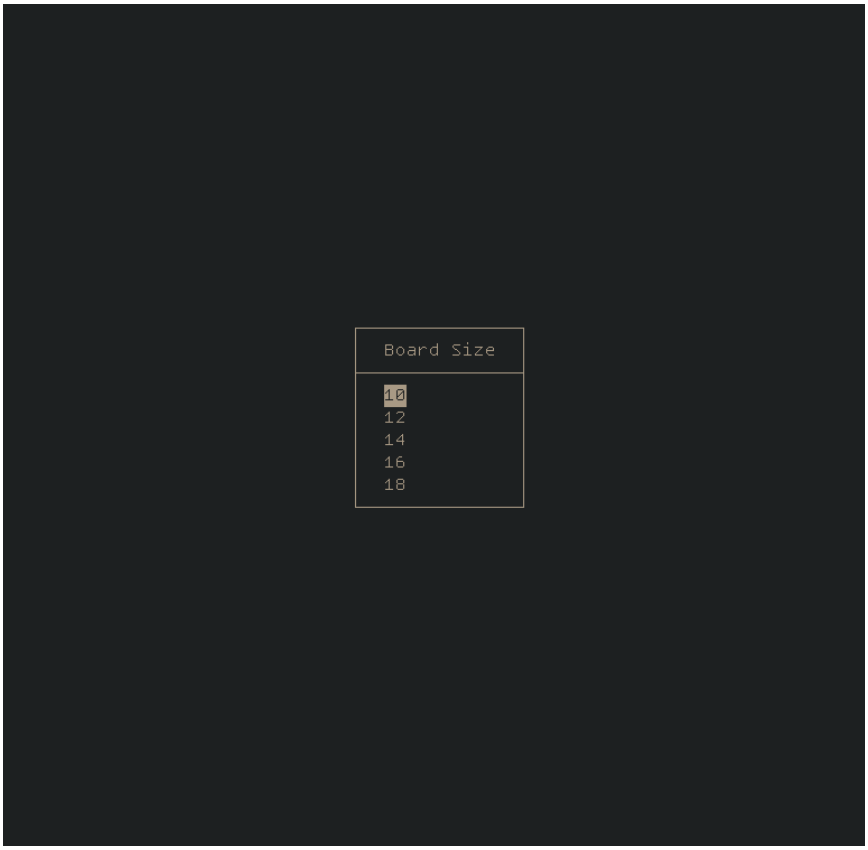
4.1 Menu iniziale



Menù principale

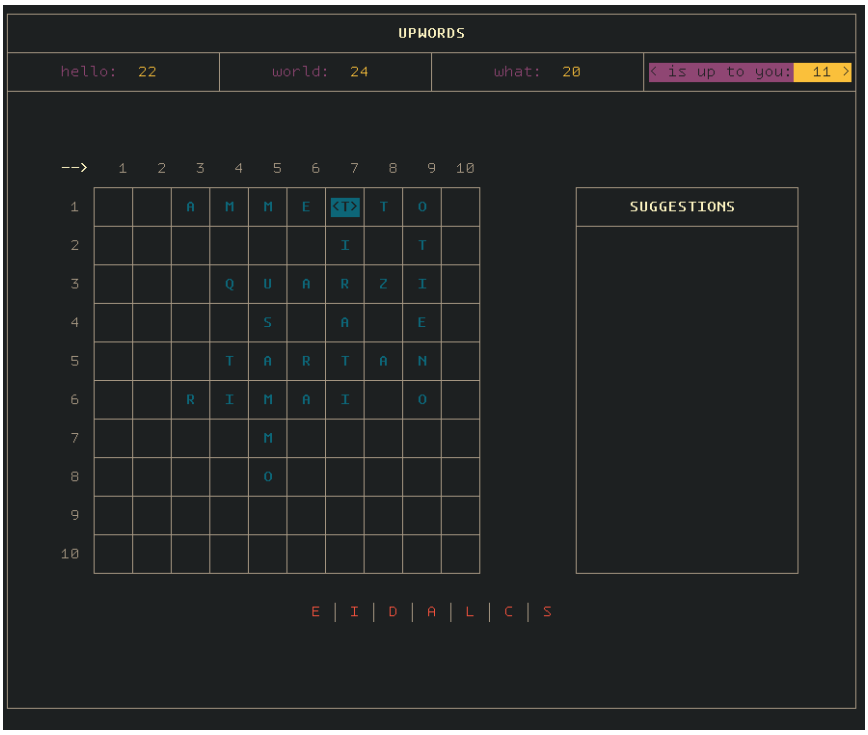


Impostazioni che permettono di cambiare alcune variabili del gioco

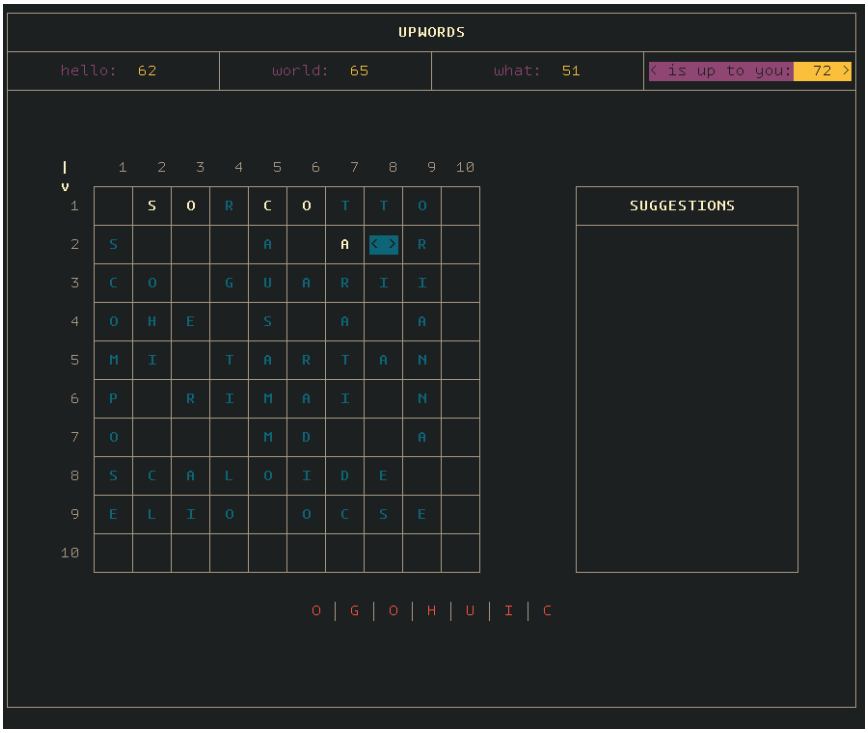


Cambiare dimensione del bordo

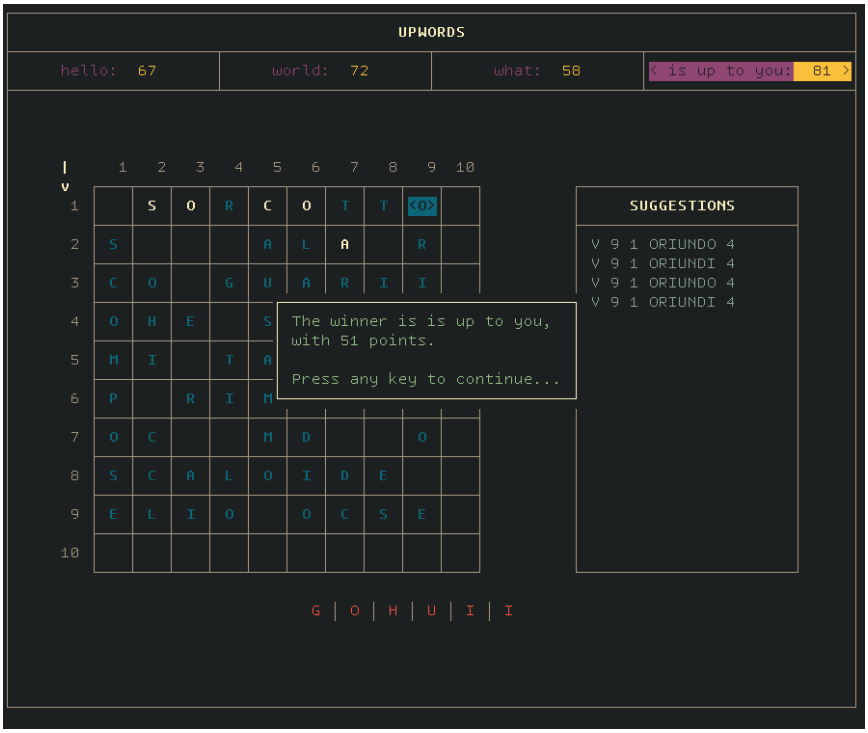
4.2 Tavolo di gioco



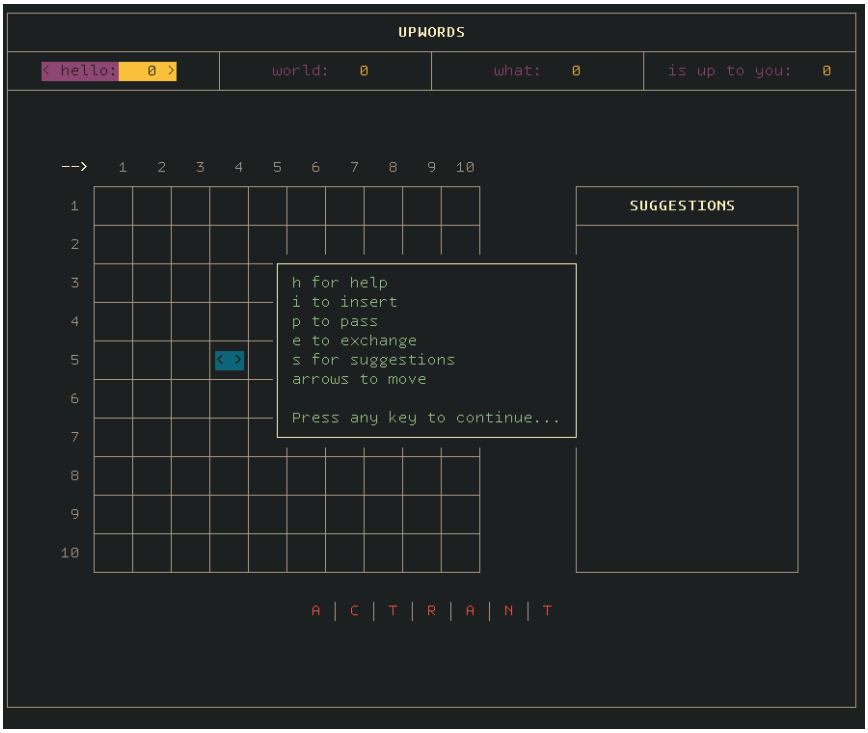
Dimostrazione gioco



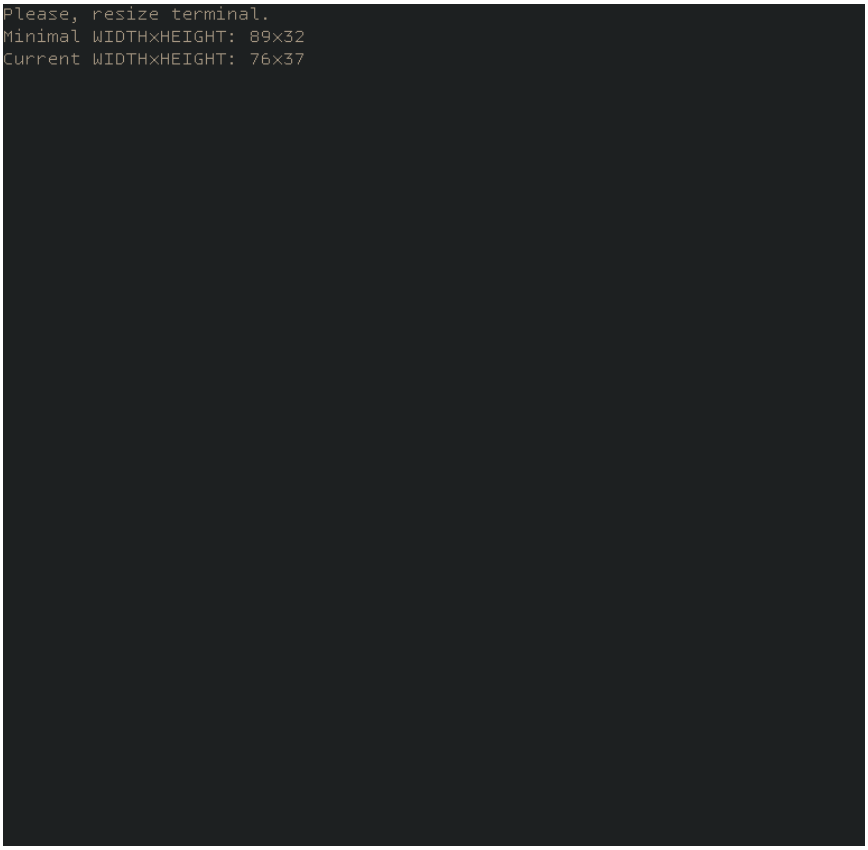
Dimostrazione parole upword



Esempio di vittoria



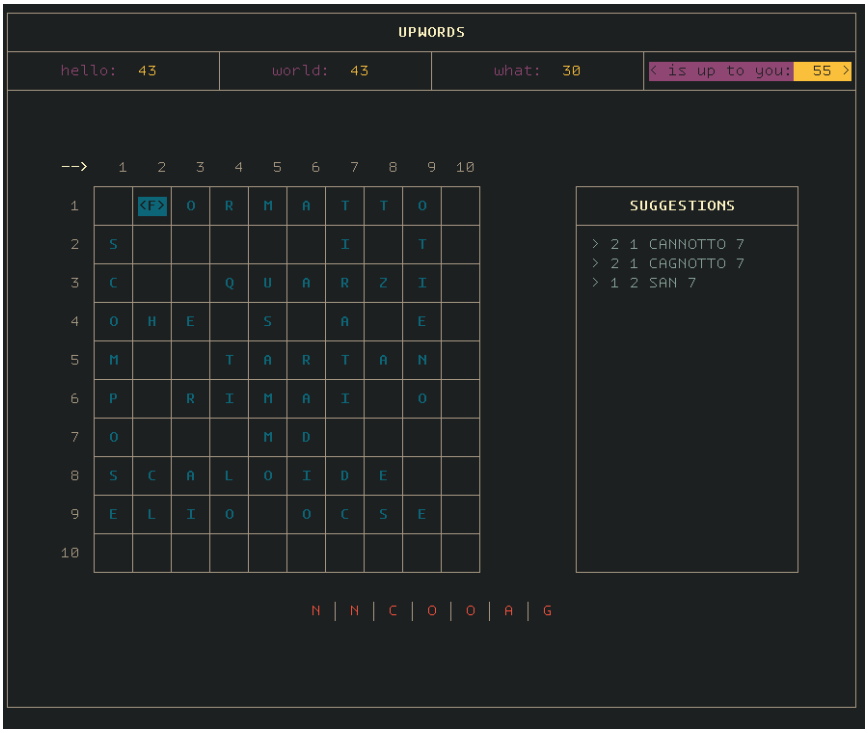
Messaggio di aiuto (premere ’h’)



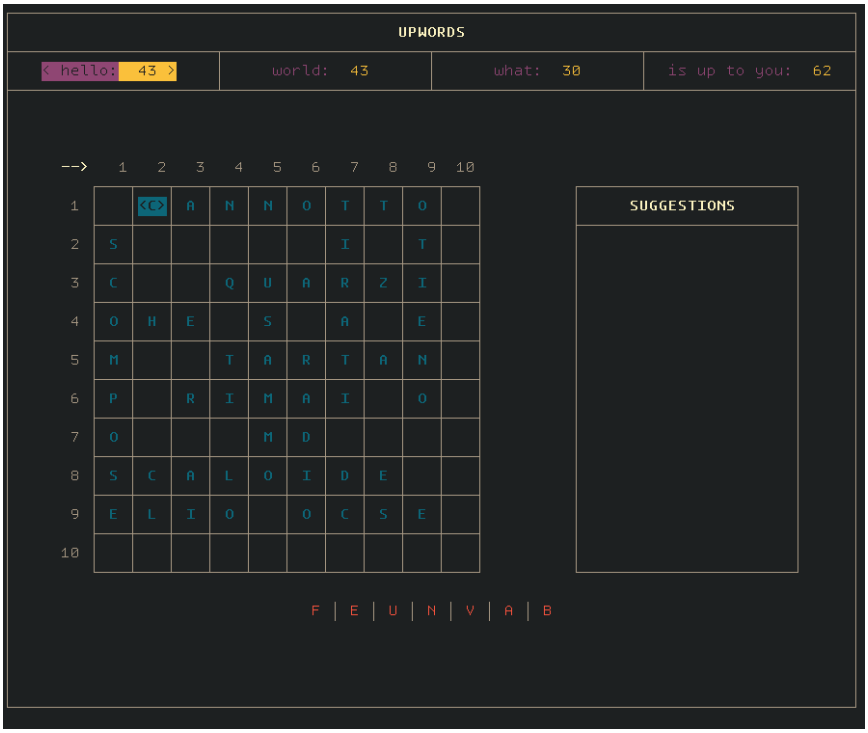
Messaggio di errore di dimensione. Il gioco ci comunica che la dimensione del terminale non è sufficiente per disegnare il tavolo da gioco.

4.3 Suggerimento

Esempio di suggerimenti



Suggerimento con upword



Esecuzione del suggerimento con upword