23/02/2018

Giorgio Gandola, Samuele Pasini, Matteo Carlino

UpWords

Relazione di Progetto

SOMMARIO

[INTRODUZIONE 2](#_Toc527049945)

[ANALISI DEL PROBLEMA 2](#_Toc527049946)

[COME FUNZIONA UPWORDS 2](#_Toc527049947)

[SOLUZIONI ADOTTATE 2](#_Toc527049948)

[STRUTTURE DATI 2](#_Toc527049949)

[DIZIONARIO 2](#_Toc527049950)

[TERRENO DI GIOCO 3](#_Toc527049951)

[LIBRERIE UTILIZZATE 3](#_Toc527049952)

[CLASSI COINVOLTE E SPIEGAZIONE 3](#_Toc527049953)

[SACCHETTO 3](#_Toc527049954)

[[int] extract\_letter() 4](#_Toc527049955)

[[char] change\_letter(char c) 4](#_Toc527049956)

[[bool] this\_is\_empity() 4](#_Toc527049957)

[Costruttore 4](#_Toc527049958)

[DIZIONARIO 4](#_Toc527049959)

[[bool] exist (const std::string &word) 4](#_Toc527049960)

[Costruttore 4](#_Toc527049961)

[FILEEXPANDER 4](#_Toc527049962)

[GIOCATORE 5](#_Toc527049963)

[[char] get\_letter(unsigend int index) 5](#_Toc527049964)

[[char] remove\_letter(unsigned int index) 5](#_Toc527049965)

[[void] add\_letter(char letter) 5](#_Toc527049966)

[[bool] hand\_full() 5](#_Toc527049967)

[TERRENO 5](#_Toc527049968)

[[char] getElement(unsigned int x, unsigend int y) 5](#_Toc527049969)

[[short int] getLevel(unsigned int x, unsigend int y) 5](#_Toc527049970)

[[bool] insertChar (unsigned int x, unsigned int y, char c) 5](#_Toc527049971)

[[void] insertChar(unsigned int x, unsigned int y, char c, unsigned short int level) 6](#_Toc527049972)

[Costruttore 6](#_Toc527049973)

[REFEREE 6](#_Toc527049974)

[COMBINATORE 6](#_Toc527049975)

[[std::string] suggerimento () 6](#_Toc527049976)

[COSTANTI 6](#_Toc527049977)

# INTRODUZIONE

## ANALISI DEL PROBLEMA

Il progetto in questione riguardava la creazione di un programma che permettesse ad un numero di utenti compresi tra 2 e 4 di giocare al gioco UpWords. In particolare le specifiche fornite chiedono l’implementazione di una funzione di “suggerimento” che restituisca all’utente la migliore parola possibile nel momento della chiamata.

### COME FUNZIONA UPWORDS

Il giocatore deve formare parole più o meno complesse sfruttando le lettere che ha in mano (7) e le lettere già presenti sul campo da gioco.

Il gioco differisce del più celebre Scarabeo per la possibilità di sovrapporre una lettera ad un’altra per poter formare una nuova parola.

Le lettere in mano ad un giocatore vengono aggiunte (per un massimo di 7 lettere in mano) ad ogni inizio turno pescando casualmente da un sacchetto e la frequenza delle lettere all’interno dello stesso ci viene fornita da specifica.

### SOLUZIONI ADOTTATE

Per lo sviluppo dell’applicativo abbiamo utilizzato C++ come linguaggio di programmazione e QtCreator come IDE per la programmazione del software. Questa non è stata una nostra scelta ma derivante dalle specifiche.

Avendo noi fatto un lavoro di gruppo abbiamo adottato soluzioni di Project Planning quali Ms Project e versioni similari per mobile che ci permettessero di progettare a priori le tempistiche del progetto; abbiamo anche adottato la piattaforma GitHub per la condivisione del codice tra i membri del gruppo in modo che ogni componesse avesse sempre il codice aggiornato.

Abbiamo poi deciso di realizzare il progetto con il metodo “ad oggetti” poiché ben si sposava con le varie parti da programmare (le varie classi) e, grazie alle piattaforme sopra citate, e alla suddivisione in classi il lavoro di gruppo è stato per noi un vantaggio.

La parte di testing è stata fatta dapprima mediante il debugger di Qt, in sun secondo luogo mediante funzioni costruite per lo scopo vista la limitatezza del debugger stesso. La parte finale è stata fatta facendo giocare da solo il programma utilizzando la funzione di suggerimento e annotando i cambiamenti in un file.

# STRUTTURE DATI

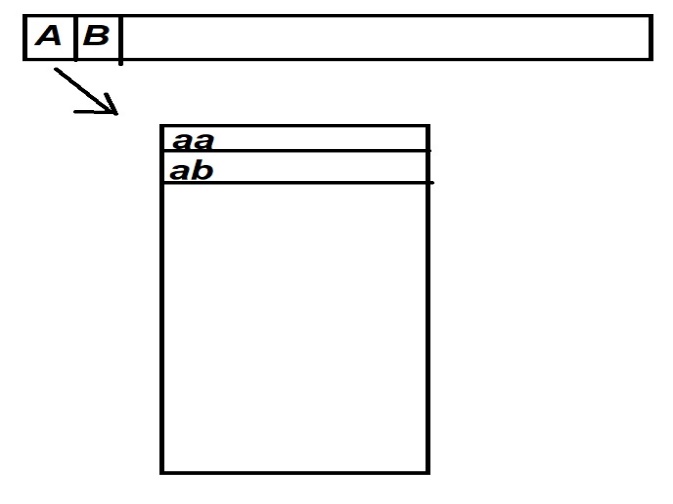
In questa sezione della relazione vengono specificate le strutture dati utilizzate.

## DIZIONARIO

La struttura del dizionario è stata motivo di dibattito all’interno del gruppo, le principali soluzioni erano 2:

* Struttura ad albero :
  + - Pro : velocità maggiore in ricerca
    - Contro: minore velocità in inserimento
* Struttura ordinata con array e vector
  + - Pro:
      * velocità di inserimento migliore (richiede meno iterazioni)
      * velocità di ricerca paragonabile mediante ricerca dicotomica
      * relativamente veloce implementazione
    - Contro:
      * Si presta meno per il suggerimento

La soluzione adottata alla fine è stata la seconda, sotto viene riportato un disegno riassuntivo ed autoesplicativo della struttura.



## TERRENO DI GIOCO

La struttura del terreno di gioco viene gestita da un’apposita classe. La struttura degli attributi interni è costituita da due matrici:

* Matrice caratteri: contiene i caratteri in campo
* Matrice livelli: contiene il livello sul quale si trova un determinato carattere

Con appositi metodi sotto spiegati le due matrici vengono messe in relazione tra loro

# LIBRERIE UTILIZZATE

Per la realizzazione dell’elaborato abbiamo sfruttato il parco di librerie che vengono fornite con Qt:

* Qt: libreria utilizzata sia per la grafica che per l’espansione del dizionario
* STD: libreria standard del C++ che comprende le librerie di io e quelle con algoritmi standard già implementati.

# CLASSI COINVOLTE E SPIEGAZIONE

## SACCHETTO

La classe sacchetto è quella che si occupa della gestione delle lettere estratte dal giocatore. La sua implementazione di basa su una struttura dati formata da un array statico di N=26 elementi (come il numero di lettere dell’alfabeto) contenente in ogni cella un intero rappresentante la frequenza della lettera in posizione I+1 nell’alfabeto.

10 qui corrisponde a 10 possibili B per esempio

I metodi più interessanti sono:

* [int] extract\_letter()
* [char] change\_letter(char c)
* [bool] this\_is\_empty()

### [int] extract\_letter()

Questo metodo restituisce una lettera random dalla struttura sopra citata e ne decrementa le possibilità di estrazione. Nel caso in cui la funzione rand restituisse una posizione con possibilità di estrazione 0 la funzione viene richiamata ricorsivamente. Ovviamente nel caso in cui il sacchetto sia vuoto viene restituito il carattere EMPTY\_FIELD dichiarato tra le costanti.

### [char] change\_letter(char c)

Questo metodo richiede in input una lettera, la riaggiunge alla struttura sopra citata e richiama extract\_letter per restituire una nuova lettera.

### [bool] this\_is\_empity()

Questo metodo restituisce true se sono state estratte tutte le lettere possibile e quindi non è piu possibile estrarre lettere. Viene utilizzata come funzione di controllo

### Costruttore

Nel costruttore vengono inizializzate le frequenze delle lettere

## DIZIONARIO

Classe che permette il caricamento e l’utilizzo del dizionario durante il gioco di UpWords.

La struttura è stata definita come sopra citato.

Nel caso il dizionario non esista come DATA.dat (un file da noi definito) viene chiamato il costruttore della classe FILEEXPANDER che si occupa della generazione ti tale file mediante espansione del dizionario datoci da specifiche.

Il metodo fondamentale, nonché l’unico pubblico è il metodo [bool] exist (const std::string &word)

### [bool] exist (const std::string &word)

Metodo che restituisce se una stringa esiste in dizionario.

Vista la struttura che abbiamo scelto di utilizzare la ricerca viene fatta spostandosi nella struttura esterna (array dim 26) all’indice della prima lettera della parola data come parametro. Successivamente viene eseguita una ricerca dicotomica e viene restituito il risultato true se la parola viene trovata.

### Costruttore

Il costruttore del dizionario si occupa di chiamare tutti i metodi privati per il caricamento delle parole all’interno della struttura dati riportata nell’apposita sezione della documentazione.

Il dizionario viene caricato mediante uno scorrimento riga per riga di un apposito file Data.dat.

Nel caso questo file non sia presente viene istanziata la classe fileexpander che provvederà alla creazione ed espansione del dizionario stesso per poi procedere al caricamento.

### FILEEXPANDER

Come sopra riportato la classe si occupa dell’espansione del dizionario.

Essa non riceve input poiché i file da espandere vengono definiti da costante.

La classe basa l’espansione delle parole sulla ricerca tramite espressioni regolari all’interno del dizionario di partenza e all’interno degli affissi fornitici con tale dizionario.

Come output viene generato il file DATA.dat che contiene il dizionario espanso e ordinato per lettera.

Grazie a questa struttura dati siamo in grado di caricare dati velocemente nella struttura dizionario.

## GIOCATORE

Classe che contiene informazioni riguardanti il singolo giocatore quali nome, punteggio e lettere in mano.

I metodi più significativi, oltre a quelli di set e get sono:

* [char] get\_letter(unsigned int index)
* [char] remove\_letter(unsigned int index)
* [void] add\_letter(char letter)
* [bool] hand\_full()

### [char] get\_letter(unsigend int index)

Il metodo restituisce la lettera in posizione index all’interno della mano del giocatore ma NON la rimuove dalla mano stessa. Viene utilizzato per mostrare a video la mano del giocatore.

### [char] remove\_letter(unsigned int index)

Il metodo funziona esattamente come il precedente ma in più setta la lettera in posizione index a EMPTY\_FIELD.

### [void] add\_letter(char letter)

Il metodo, se possibile, aggiunge una lettera alla mano del giocatore

### [bool] hand\_full()

Il metodo restituisce true se nessuna cella dell’array contenente la mano del giocatore contiene il carattere EMPTY\_FIELD

## TERRENO

La classe terreno contiene lo stato del terreno di gioco.

Le strutture in gioco sono due matrici che contengono la prima i caratteri in campo e la seconda il livello dei caratteri in campo.

I metodi più importanti di questa classe sono:

* [char] getElement(unsigned int x, unsigend int y)
* [short int] getLevel(unsigned int x, unsigend int y)
* [bool] insertChar (unsigned int x, unsigned int y, char c)
* [void] insertChar(unsigned int x, unsigned int y, char c, unsigned short int level)

### [char] getElement(unsigned int x, unsigend int y)

Metodo che restituisce il carattere in posizione xy

### [short int] getLevel(unsigned int x, unsigend int y)

Metodo che restituisce su che livello si trova il carattere in posizione xy

### [bool] insertChar (unsigned int x, unsigned int y, char c)

Metodo che permette l’inserimento di un carattere c in posizione xy.

Ritorna true solo se l’inserimento ha esito positivo e quindi solo se il livello del carattere è minore o uguale al livello massimo consentito dal gioco

### [void] insertChar(unsigned int x, unsigned int y, char c, unsigned short int level)

Metodo che permette l’inserimento di un carattere senza controlli specificando xyz del carattere stesso.

Viene utilizzato per la comunicazione con classi che modificano il terreno radicalmente

### Costruttore

Il costruttore di questa classe inizializza il campo da gioco con il carattere EMPTY\_FIELD e porta i livelli a 0

## REFEREE

## COMBINATORE

La classe combinatore si occupa di trovare il migliore suggerimento possibile in base alla situazione del campo da gioco e in base alle lettere nella mano del giocatore.

Per ottenere un suggerimento in tempi accettabili abbiamo deciso di eliminare alcune casistiche in quanto hanno probabilità di verifica.

Il cuore del suggerimento si basa sulla ricerca della parola più lunga possibile utilizzando le lettere nella mano di un giocatore e/o le lettere sul campo da gioco.

Vengono generati tutti i possibili anagrammi e poi confrontati in dizionario per verificare la validità della parola. Nel caso non sia possibile ottenere un anagramma di lunghezza maggiore della parola precedentemente generata, esso viene restituito come migliore suggerimento (ovviamente se valido in dizionario). Vengono automaticamente scartati gli anagrammi di dimensione inferiore a quello attuale tramite chiamata ricorsiva.

Oltre al costruttore che inizializza il l’oggetto il metodo più importante resta

* [std::string] suggerimento ()

### [std::string] suggerimento ()

Metodo che non riceve parametri in input poiché essi vengono inizializzati mediante il costruttore dell’oggetto o mediante gli appositi metodi di get e set.

Restituisce una stringa che contiene la migliore parola possibile al momento della chiamata.

E’ importante precisare che in caso in cui il campo da gioco sia vuoto il metodo differenzia il procedimento di ricerca della parola in modo da ottimizzare le chiamate ricorsive.

## COSTANTI

Classe che permette di unificare le costanti di tutto il programma In modo da non dover modificare il codice ma soltanto le costanti statiche all’interno della classe costanti.