

Progetto Fondamenti di Informatica

Davide De Marco



**POLITECNICO**  
**MILANO 1863**

Made with L<sup>A</sup>T<sub>E</sub>X

# Contents

<b>1 Richieste elaborato</b>	<b>3</b>
<b>2 Introduzione</b>	<b>3</b>
2.1 Il gioco . . . . .	3
2.2 Per cominciare . . . . .	3
2.3 Assegnazione punti . . . . .	7
<b>3 Strutture Dati</b>	<b>8</b>
3.1 Tile . . . . .	8
3.2 tileList . . . . .	8
3.3 Bag . . . . .	9
3.4 Player . . . . .	9
3.5 Dictionary e dNode . . . . .	9
3.6 Board . . . . .	10
<b>4 Interfaccia grafica e SFML</b>	<b>11</b>
<b>5 Main</b>	<b>11</b>
<b>6 Algoritmi principali</b>	<b>12</b>
6.1 Ricerca nel dizionario . . . . .	12
6.2 Controllo del tabellone di gioco . . . . .	12
6.3 Calcolo del punteggio . . . . .	13
6.4 Suggerimento . . . . .	15
<b>7 Appendice</b>	<b>17</b>

## 1 Richieste elaborato

Viene richiesta la programmazione del gioco "Scarabeo" utilizzando il linguaggio di programmazione c++. Si chiede inoltre di implementare un algoritmo di suggerimento in grado di formare la parola migliore possibile utilizzando la mano del giocatore di turno

## 2 Introduzione

### 2.1 Il gioco

componenti necessari:

- tabellone 17 x 17
- Lettere dell'alfabeto italiano (130 lettere tasselli)
- sacchetto per contenere i tasselli

Scarabeo può essere giocato da 2 a 4 giocatori.

Ciascun giocatore all'inizio del gioco pesca 8 lettere casualmente dal sacchetto, chiameremo queste lettere "mano del giocatore"

A turno i giocatori dovranno tentare di formare sul tabellone delle parole di senso compiuto, utilizzando le lettere presenti nella propria mano.

Formando parole i giocatori ottengono punti, e l'obiettivo del gioco è quello di ottenere il maggior numero di punti.

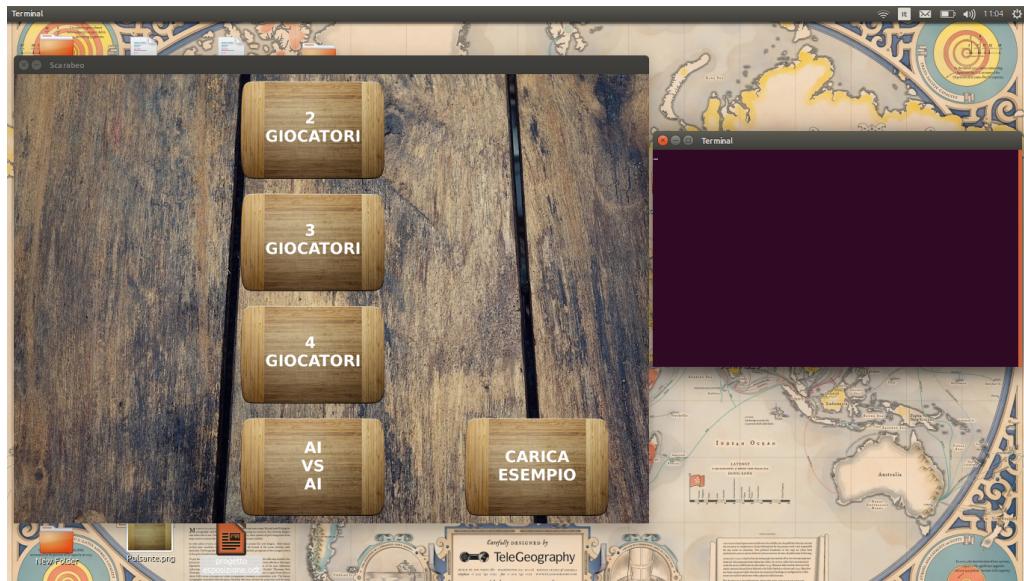
Vince chi alla fine della partita ha totalizzato più punti.

La partita si conclude quando sono esaurite tutte le lettere presenti nel sacchetto ed un giocatore finisce tutte le lettere della propria mano.

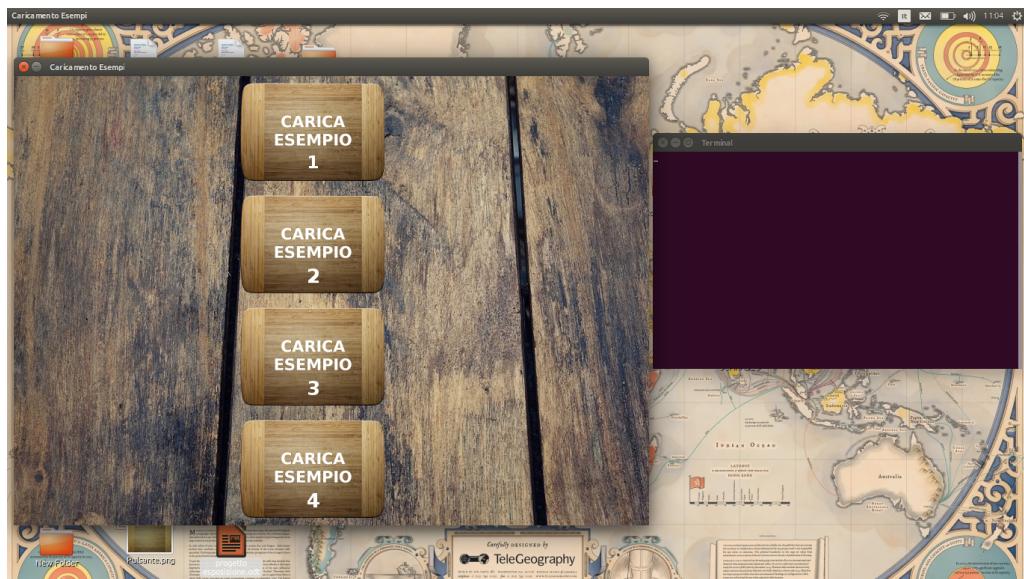
Il giocatore che ha giocato l'ultima parola ottiene in aggiunta i punti delle lettere presenti nelle mani degli altri giocatori.

### 2.2 Per cominciare

Il gioco si presenta con una schermata che permette di scegliere la modalità di gioco.



È possibile scegliere il numero di giocatori oppure di veder giocare tra loro due intelligenze artificiali. È inoltre disponibile l'opzione di caricamento di alcuni esempi particolari.



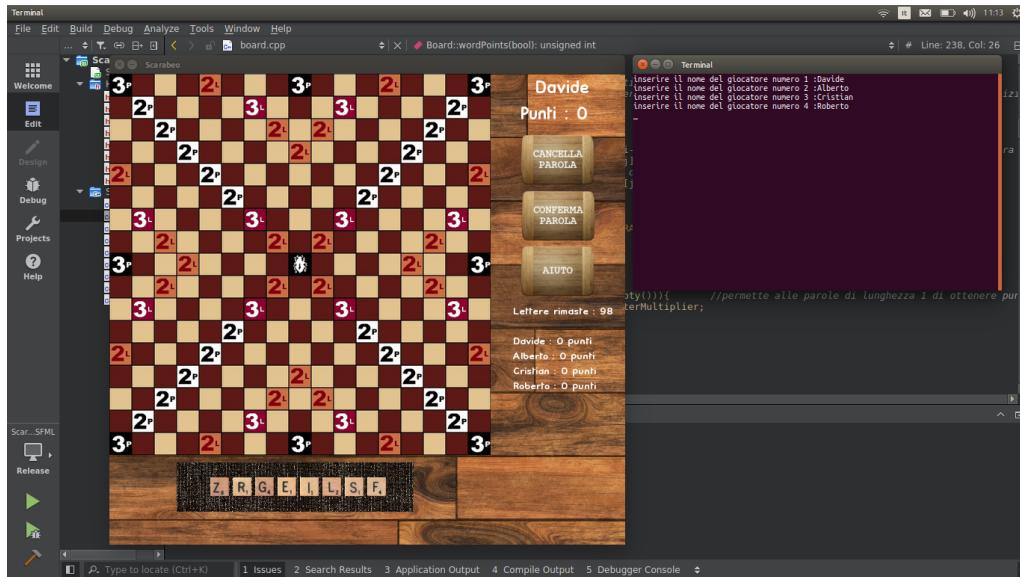
Selezionando il numero di giocatori verrà inizialmente chiesto di inserire il

nome di ogni giocatore

Successivamente apparirà la schermata di gioco, con il tabellone, pulsanti per permettere la conferma della parola, la richiesta di suggerimento e di annullare l'ultimo inserimento.

Il nome del giocatore di turno è rappresentato in alto a destra nella schermata, con il relativo punteggio, mentre in basso a destra sono rappresentati i punteggi di tutti i giocatori con i relativi nomi.

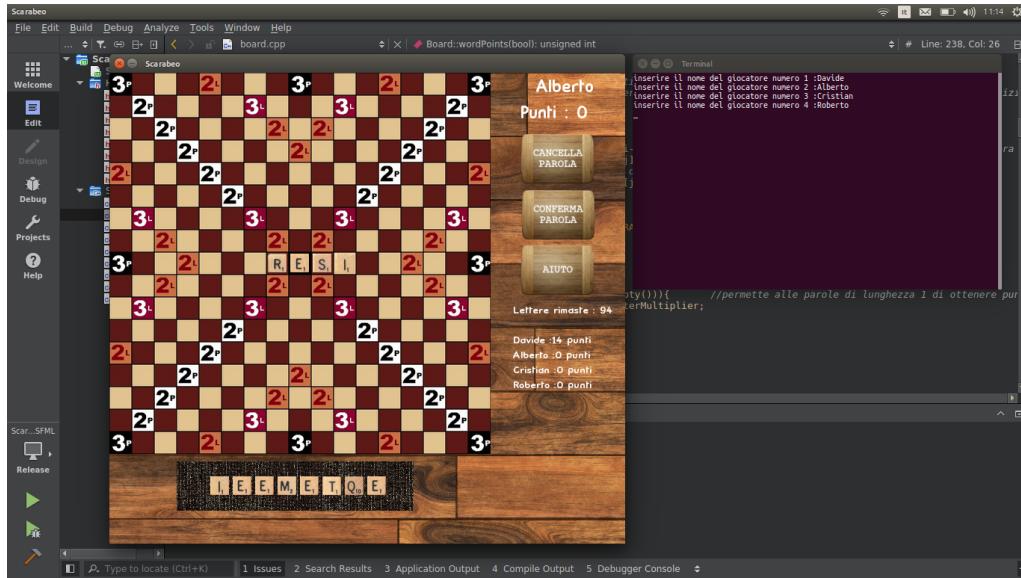
Da questa schermata è possibile, utilizzando il click sinistro del mouse, trascinare le lettere sul tabellone.



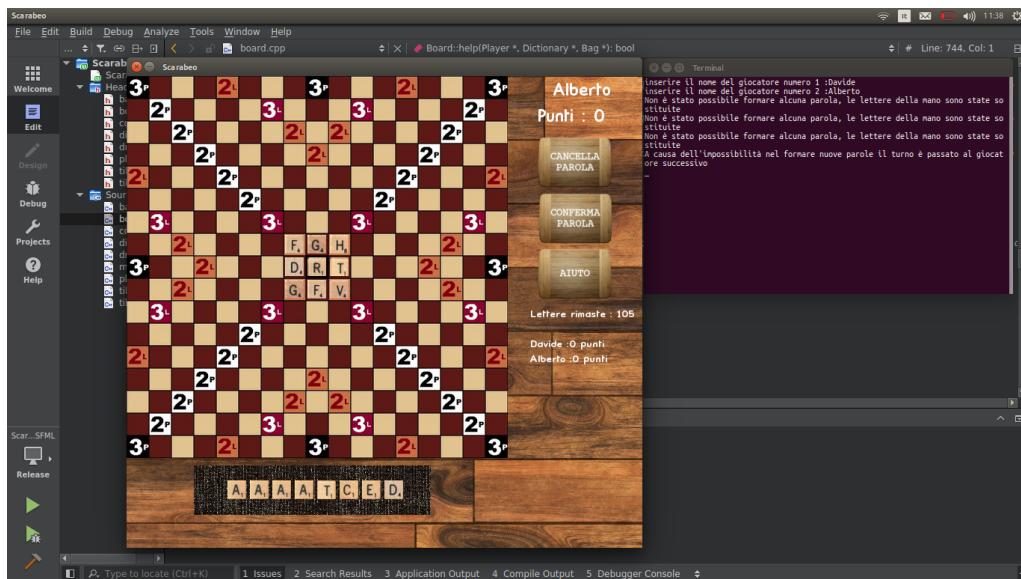
La posizione della lettera verrà aggiustata automaticamente per centrare la casella dopo il rilascio del click sinistro del mouse.

Una volta confermato l'inserimento il tabellone di gioco verrà analizzato per capire se non vi sono parole errate sul tabellone e verranno assegnati i punti relativi alla parola. Nel caso in cui la parola inserita fosse insensata allora le lettere ritorneranno in mano al giocatore.

Premendo il pulsante *aiuto* il programma provvederà autonomamente a calcolare la parola migliore da giocare e a posizionarla sul tabellone di gioco, aggiornando il punteggio del giocatore.



Nel caso in cui non fosse possibile formare alcuna parola tramite il suggerimento, allora verrà cambiata la mano del giocatore fino ad un massimo di 3 volte. Dopo 3 tentativi falliti il programma provvederà a cambiare giocatore e si comporterà allo stesso modo.



Nel caso in cui nessuno giocatore fosse in grado di formare una parola verrebbe dichiarato lo stallo e la partita si interromperebbe proclamando vincitore il giocatore con il punteggio più alto.

### 2.3 Assegnazione punti

Ciascuna lettera del gioco ha un punteggio associato ad essa.  
Sul tabellone di gioco sono presenti delle caselle di colore diverso con delle scritte particolari, queste caselle indicano dei moltiplicatori.

I moltiplicatori sono di 2 tipi diversi

- Moltiplicatori di lettera
- Moltiplicatori di parola

Ciascun moltiplicatore può avere un valore di 2 o di 3.

I moltiplicatori di lettera aumentano il punteggio della singola lettera posizionata sulla casella del moltiplicatore di lettera, il punteggio finale generato da quella lettera è dato da  $puntiLettera \times valoreMoltiplicatoreLettera$   
I moltiplicatori di parola aumentano il punteggio della parola formata passante per la casella del moltiplicatore di parola, il punteggio finale generato dalla parola è  $sommaPuntiLettere \times valoreMoltiplicatoreParola$

A questi punti vanno aggiunti eventuali punti *speciali*.

- 10 punti bonus se per formare la parola sono state utilizzate 6 lettere
- 30 punti bonus se per formare la parola sono state utilizzate 7 lettere
- 50 punti bonus se per formare la parola sono state utilizzate tutte 8 le lettere
- 100 punti bonus se si riesce a comporre la parola *scarabeo* o *scarabei*

A fine partita, l'ultimo giocatore ad aver composto una parola, riceverà come bonus la somma dei punti delle lettere presenti nelle mani degli avversari.

## 3 Strutture Dati

Dovendo rappresentare elementi di gioco complicati ho deciso di utilizzare la programmazione ad oggetti per l'implementazione del progetto.

Ho cominciato stilando una diagramma delle classi, ragionando sugli elementi necessari per giocare a scarabeo, subito pensai a dover creare le seguenti classi:

- Bag
- Board
- Dictionary
- Player
- Tile

### 3.1 Tile

La maggior parte delle classi elencate sono vincolate dall'utilizzo dei tasselli di gioco, per questo motivo la classe **Tile** risulta essenziale.

Ogni Tile è definito da una lettera, un punteggio, uno stato, una sprite per essere rappresentato graficamente e un booleano per identificare i jolly

Le Tile singolarmente sono utili, ma per le prossime classi sarà necessario poter immagazzinare le Tile in una struttura dati, in particolare una lista.

Nonostante la classe list sia estremamente rinomata, efficiente e stabile ho deciso di scrivere una mia classe *tileList*, principalmente per piacere e sfida personale e per il fatto che si tratta di un progetto universitario, che quindi non verrà distribuito su larga scala.

### 3.2 tileList

Lista di Tile, definita da due variabili, una contenente il puntatore al Tile corrente e l'altra contenente il puntatore al prossimo nodo della lista.

I metodi di questa classe sono i classici metodi necessari per l'estrazione di un nodo dalla lista, di *push\_back()*, ma anche altri metodi specifici come *listPoints()* che restituisce il totale dei punti delle tile presenti nella lista

### 3.3 Bag

Bag è la classe che si occupa di creare le tile necessarie per lo svolgimento della partita, le immagazzina fornendo metodi per l'estrazione dei tasselli. Bag è definita da due variabili: un puntatore a tileList e size, in cui è salvato il numero di tile presenti all'interno del sacchetto. Size non è indispensabile, ma permette di evitare di chiamare il metodo *size()* della classe tileList velocizzando un minimo il processo.

I Tile generati durante la costruzione del sacchetto sono gli stessi che verranno utilizzati per tutta la durata della partita e che verranno, durante il gioco, spostati dal sacchetto alla mano del giocatore ed infine sul tabellone di gioco (nel caso dei jolly potrebbero tornare nuovamente in mano al giocatore).

### 3.4 Player

La classe *Player* ha il compito di immagazzinare informazioni relative al giocatore come nome e punteggio attuale e di gestire la mano, permettere di pescare tasselli dal sacchetto o scambiare completamente i propri con altri presenti nel sacchetto.

Player contiene anche metodi che permettono di controllare la validità della posizione in cui una lettera può essere giocata, in particolare controlla che il giocatore nello stesso turno non sovrapponga due lettere presenti nella propria mano sul tabellone.

### 3.5 Dictionary e dNode

Il dizionario ha come funzione quella di contenere le parole accettabili dal programma e di verificare se quelle formate dall'utente sono valide.

Per salvare le parole valide ho deciso di utilizzare un albero in cui ogni nodo contiene le possibili lettere successive alla lettera scelta nel nodo precedente. Per fare ciò ho avuto bisogno di creare una classe dNode, nodo del dizionario. Per definire dNode è stato necessario scrivere una struct, *Dnode\_cell*, la quale è composta da un bool *exists* e da un puntatore a dNode.

dNode è formata da un array di *Dnode\_cell* di lunghezza pari alla lunghezza dell'alfabeto italiano +1 (carattere terminatore), ogni cella dell'array corrisponde ad una lettera dell'alfabeto.

La costruzione del dizionario avviene nel modo seguente:

Utilizzando la libreria *fstream* viene letto carattere per carattere un file di

testo in cui sono salvate le parole accettabili. Per ogni lettera viene modificato il dNode corrente rendendo *true* la variabile *exists* nella posizione dell'array di *Dnode\_cell* relativa alla lettera letta, fatto ciò si modifica la variabile *nextptr* in modo che punti ad un nuovo *dNode* appena creato.

Giunti al termine della parola viene resa *true* la variabile *exists* apposita per il carattere terminatore, questo permette di segnare quando una parola ha senso senza precludere la possibilità di accettare parole con la stessa radice che continuano in modo differente, come per esempio *cassa* e *cassaforte*; fino alla quinta lettera i dNode percorsi sono gli stessi, il dNode successivo sarà comunque lo stesso, ma conterrà sia il carattere terminatore che la lettera f necessaria per completare la parola cassaforte.

### 3.6 Board

Board ha lo scopo di immagazzinare i tasselli già giocati, controllare che il tabellone di gioco sia in regola, assegnare i punteggi delle parole ai giocatori, caricare esempi e generare suggerimenti quando richiesti.

Board è definita da due matrici bidimensionali: tBoard, contenente puntatori a Tile ed sBoard contenente le Sprite del tabellone per l'interfaccia grafica.

tBoard è una matrice di dimensione *board\_size* × *board\_size* che contiene tutti i tasselli giocati la cui validità è stata controllata.

sBoard è una matrice di dimensione 2 × *board\_size*; la prima riga viene utilizzata per salvare le sprite di sfondo del tabellone, mentre la seconda per immagazzinare quelle dei Tile giocati sul tabellone.

## 4 Interfaccia grafica e SFML

Per creare l’interfaccia grafica ho fatto uso della libreria SFML, la quale mette a disposizione classi per la gestione di sprites, finestre, testi, eventi, ecc..

Per cominciare è necessario creare un oggetto *RenderWindow*, cioè una finestra, con le proprie dimensioni e altre opzioni che ne definiscono lo stile. Successivamente bisogna creare i vari oggetti grafici che vogliamo rappresentare nella finestra.

Le *Sprite* sono elementi grafici definiti da una *Texture*, la quale permette di caricare immagini da file che poi andranno disegnate sulla finestra tramite la *Sprite*.

Definiti gli elementi necessari è possibile disegnarli tramite il metodo *draw* di *RenderWindow*, chiamato una volta per frame all’interno di un ciclo che dura fintanto che la finestra rimane aperta.

Ogni volta che viene compiuta un’azione dall’utente, questa viene salvata all’interno di una variabile permettendo di scrivere codice in relazione al comportamento dell’utente.

Gli eventi di cui ho fatto uso sono *MouseButtonDown*, *MouseButtonReleased*, *Closed* (click sul pulsante per la chiusura della finestra).

Utilizzando in modo combinato gli eventi, la posizione del puntatore del mouse e delle varie sprites ho potuto scrivere il codice relativo al click sui vari pulsanti e del drag & drop, codice abbastanza semplice che non risiede spiegazioni particolari.

## 5 Main

Tutto il codice legato all’interfaccia grafica si trova nel main (per necessità legate alla durata di vita della finestra), assieme alla gestione dei turni.

Il turno viene passato quando un giocatore inserisce una parola o chiede un aiuto e la parola giocata è valida e quindi genera punti. Semplicemente dopo aver premuto il pulsante di conferma/aiuto e dopo aver risolto tutte le chiamate di funzioni per il controllo validità parola, ecc.. viene controllato il punteggio del giocatore corrente, se è diverso dal punteggio precedente alla conferma/aiuto allora viene passato il turno al giocatore successivo.

Per la gestione del turno ho utilizzato semplicemente una variabile di tipo *size\_t*, *turn*, che incremento ad ogni passaggio di turno fintanto che *turn < numeroGiocatori*.

All’avvio del programma, dopo aver scelto la modalità di gioco, vengono

inizializzati un oggetto Bag, un oggetto Board e tot oggetti Player (in base alla scelta). I player sono salvati all'interno di una lista, che da flessibilità sulla scelta del numero di giocatori e che permette di gestire facilmente in cambio turno.

## 6 Algoritmi principali

### 6.1 Ricerca nel dizionario

Il compito del dizionario è quello di controllare la validità di una parola.

La ricerca di una parola avviene in modo ricorsivo, chiedendo in input la parola da ricercare, la posizione *pos* della prossima lettera da controllare all'interno della parola, la posizione finale della parola (per evitare di chiamare costantemente *word.size()*) ed infine il dNode corrente.

Ad ogni chiamata ricorsiva viene controllata la presenza nel dNode corrente della lettera in posizione *pos* della parola cercata, se questa esiste viene effettuata la chiamata ricorsiva passando come parametri la stessa parola, *pos+1*, la stessa posizione finale e il dNode successivo alla lettera appena controllata.

Se la lettera controllata non è presente nel dNode corrente allora la funzione restituirà *false*, se invece il carattere in posizione *pos* della parola è un carattere terminatore, e questo è "presente" nel dNode, allora la funzione restituirà *true*.

### 6.2 Controllo del tabellone di gioco

Utilizzando l'interfaccia grafica, il giocatore può trascinare le proprie lettere sul tabellone e confermare la parola appena inserita, a questo punto comincia il controllo del tabellone di gioco.

Inizialmente vengono controllate tutte le lettere presenti nella mano del giocatore (le lettere appena trascinate con l'interfaccia grafica sono ancora parte della mano) cercando tutti i tile la cui posizione a livello grafico è contenuta all'interno del tabellone di gioco.

Ogni lettera all'interno dell'area del tabellone viene estatta dalla mano del giocatore, il suo status viene modificato a *checking* e viene infine "inserita" in tBoard nella posizione in cui è stata giocata (la posizione all'interno della matrice viene ricavata tramite il calcolo *int(posizionGrafica/dimensioneGraficaImmagine)*).

A questo punto avvengono vari controlli:

**isWordSeg :** Consiste nel controllare che le varie lettere siano state giocate in posizioni consentite, cioè che la parola non sia "segmentata".

**wordCrossed :** Consiste nel controllare che venga intersecata almeno una delle parole presenti sul tabellone se il tabellone non è vuoto, altrimenti controlla che la parola passi per il centro del tabellone.

**rowToString :** Permette di convertire in stringa le righe della matrice in modo da poter controllare le varie parole successivamente. Le righe da convertire sono solo quelle in cui è stata appena inserita una nuova lettera, le altre non necessitano di controlli ulteriori.

**columnToString :** Funzione analoga a *rowToString* ma che permette di convertire le colonne in stringa al posto delle righe.

**checkString :** Esamina singolarmente le parole contenute nella stringa passata come parametro e ne controlla la validità usando i metodi della classe *Dictionary*. Viene chiamata passando come parametro le stringhe generate da *rowToString* e da *columnToString*.

Se tutti i controlli vanno a buon fine il giocatore pesca dal sacchetto le lettere mancanti per arrivare ad avere 8 tasselli in mano (se vi sono abbastanza lettere disponibili nel sacchetto), gli vengono assegnati i punti ottenuti dalla formazione della parola e tutte le lettere sul tabellone vengono impostate a *status = checked*.

Se invece uno dei controlli restituisce un risultato negativo le lettere appena giocate sul tabellone vengono riposte nella mano del giocatore.

### 6.3 Calcolo del punteggio

Il punteggio ottenuto dall'inserimento di una parola è calcolato come la somma dei punti della parola inserita + il punteggio delle eventuali altri parole formate da incroci delle lettere appena inserite con lettere già presenti sul tabellone.

Sul tabellone sono inoltre presenti dei moltiplicatori di parola e dei moltiplicatori di lettera, gli effetti di questi vengono considerati solo se la lettera presente sopra di essi è stata giocata in questo turno.

Il punteggio delle parole formate va calcolato individualmente, ognuna con i propri moltiplicatori, infine per ottenere il punteggio totale va effettuata la

somma tra questi.

Il calcolo del punteggio viene effettuato tramite il metodo *wordPoints()* della classe Board.

*wordPoints()* si occupa anche di aggiungere eventuali punteggi speciali chiamando la funzione *specialPoints()*.

Illustrerò il caso dell'inserimento di una parola in verticale, il caso orizzontale è analogo, cambia solamente la direzione.

Inizialmente viene calcolato il punteggio legato alla parola "principale" cioè quella nel verso di inserimento (definito dal metodo *WordDirection()* basato sulla posizione dei Tile checking presenti sul tabellone di gioco).

Per fare ciò comincio scorrendo il tabellone di gioco alla ricerca del primo Tile checking, che sarà per forza il primo della parola formata (scorrendo la matrice da sinistra a destra e dall'alto verso il basso). A questo punto controllo le caselle precedenti nella direzione di inserimento, cioè nel caso verticale controllo la presenza di eventuali tile presenti sopra la Tile appena trovata. Continuo a controllare le caselle superiori fintanto che non trovo uno spazio vuoto o raggiungo il limite del tabellone, a questo punto so che la prima lettera a comporre la parola si troverà esattamente sotto lo spazio vuoto, oppure nella posizione corrente nel caso in cui avessi raggiunto il bordo della griglia di gioco.

Una volta ottenuta la posizione della prima lettera, scorro tutte le lettere della parola, ne sommo i punti e le salvo in una stringa, cioè scorro la colonna della matrice in cui si trova la parola fino a quando trovo la prima casella contenente un puntatore a nullptr, sommo i vari punti e accedo ad una stringa ognuna delle lettere così incontrate.

Salvata la stringa controllo se contiene "scarabeo" o "scarabei" ed è quindi necessario aggiungere 100 punti bonus a quelli ottenuti dalle singole lettere. Ottenuto il punteggio della singola parola vado a controllare eventuali altre intersezioni con parole già formate, ne calcolo i punteggi e li aggiungo al punteggio totale. Ovviamente se l'inserimento della parola è avvenuto in verticale allora le altre parole da controllare saranno orizzontali e viceversa. Il procedimento è quindi analogo a quello per trovare la parola principale, ma effettuato per ogni tile checking.

Infine conto quante lettere sono state giocate in questo turno e grazie al metodo *specialPoints* vado ad aggiungere i punteggi legati alle lettere usate. Per quanto riguarda l'inserimento di lettere singole, per calcolare il punteggio si utilizza il controllo punti per le "ulteriori parole formate", sia quello scritto per le parole con direzione verticale sia quello scritto per le parole inserite in orizzontale.

## 6.4 Suggerimento

L'algoritmo di suggerimento ha come scopo quello di testare tutte le parole possibili formabili sul tabellone, salvarne la migliore e giocarla.

Il primo problema consiste nel capire quali caselle del tabellone possono essere utilizzate per formare una parola, è infatti inutile, per esempio, andare a provare a formare parole a partire da ogni casella del tabellone se poi gli unici incroci sono possibili solamente su 2 colonne.

Con posizioni accettabili si intendono celle successive a celle già occupate (che verrebbero utilizzati per cominciare la parola) oppure celle precedenti una cella occupata ad un massimo di *hand.size()* caselle di distanza (si tratta solo di una scrematura iniziale che permette però di ridurre di molto i tempi del processo). Le caselle di partenza per tentare di formare un suggerimento vengono salvate all'interno di 2 matrici di bool, una per le parole verticali e una per le parole orizzontali.

A questo punto partendo dalle caselle appena scelte vado a controllare se la casella precedente è vuota per evitare di formare una parola accodandola ad una parola già esistente e di formare perciò con tutta probabilità una parola insensata. Se la casella precedente è vuota procedo a chiamare la funzione ricorsiva *addtile()* una volta per ogni lettera presente in mano che non sia ancora stata utilizzata(cioè tutte nel caso della prima chiamata), in caso contrario invece controllo le caselle precedenti fino a trovare uno spazio vuoto oppure fino al raggiungimento del bordo del tabellone e chiamerò la funzione *addtile()* riferita alla lettera presente sul tabellone nell'ultima casella piena trovata.

le chiamate verticali di *addtile()* sono separate dalle chiamate orizzontali in modo da poter scegliere le posizioni corrette delle lettere per formare la parola.

Prima di scegliere la lettera di partenza vengono utilizzati i dNode (nodi del dizionario) per controllare che la lettera scelta, in quella posizione all'interno di una parola abbia senso.

Il metodo *addtile()* funziona in modo abbastanza intuitivo, prende in considerazione una lettera, passa al nodo successivo dell'albero del dizionario in base alla lettera scelta, scorre la mano del giocatore alla ricerca di una lettera inutilizzata, controlla la sua esistenza all'interno del nodo corrente del dizionario e, se valida richiama se stessa andando a compiere gli stessi passaggi per le tot lettere successive.

Nel caso in cui, la lettera scelta o la posizione non fossero accettabili o se, semplicemente, non fossero più disponibili lettere nella mano del giocatore, allora la funzione provvederebbe al controllo del tabellone con la

parola formata fino a quel momento e restituirebbe il punteggio legato ad essa (negativo nel caso in cui la parola non fosse accettabile) e salverebbe la parola migliore in una stringa.

Il controllo del tabellone avviene tramite la chiamata del metodo *hintPoints* che controlla in modo analogo a *updateBoard* la validità della parola utilizzando i metodi *wordCrossed* e *checkString*, infine chiama *wordPoints* per ottenere il punteggio associato ad essa.

Risulta una chiamata ricorsiva e ritornati al punto in cui la chiamata era avvenuta, viene controllato il punteggio formato dalla parola creata in questo modo, se è maggiore delle altre parole formate fino a quel momento, allora viene temporaneamente salvata in una stringa, ne viene salvato il punteggio e vengono effettuati i confronti con tutte le parole componibili a partire dalle successive lettere della mano.

Una volta ritornati "in superficie", cioè alla prima chiamata di *addtile()*, viene effettuato nuovamente il controllo sul punteggio massimo e salvata eventualmente la parola migliore con il relativo punteggio.

Essenzialmente ad ogni chiamata ricorsiva corrisponde un avanzamento nella parola, in ognuna delle chiamate, se la casella relativa alla chiamata è vuota, vengono testate tutte le lettere disponibili in quel momento nella mano del giocatore (tutte le tile non checking), altrimenti viene utilizzata la lettera presente nella casella.

L'effettivo posizionamento delle lettere e la gestione della parte grafica avvengono in secondo luogo, essendo in possesso della parola da formare e dalla posizione di partenza, scorro la parola e vado ad estrarre dalla mano del giocatore una lettera dello stesso tipo (se la lettera non viene trovata allora cerco un jolly), la gioco sul tabellone nella posizione apposita e imposto la posizione a livello grafico.

## 7 Appendice

Elenco varie risorse utilizzate durante la stesura dell'elaborato

- <https://stackoverflow.com/> immancabile fonte di ispirazione per ogni programmatore
- <http://www.cplusplus.com/>
- <https://www.sfml-dev.org/learn.php> guide ufficiali per imparare ad utilizzare SFML
- <https://www.youtube.com/> per vari tutorial riguardanti SFML
- Bjarne Stroustrup programming principles and practice using c++