RELAZIONE LEDA

Il progetto è composto da 7 file:

- “classi.h”: contiene le dichiarazioni delle classi “tabellone”, “dizionario” e “game”;
- “interfaccia.h”: contiene la dichiarazione della classe “interfaccia”;
- “tabellone.cpp”: contiene le implementazioni delle funzioni della classe “tabellone”;
- “dizionario.cpp”: contiene le implementazioni delle funzioni della classe “dizionario”;
- “game.cpp”: contiene le implementazioni delle funzioni della classe “game”;
- “interfaccia.cpp”: contiene le implementazioni delle funzioni della classe “interfaccia”;
- “main.cpp”: contiene la funzione principale “main”;

Come si può notare dall'elenco dei file, per svolgere il progetto ci siamo serviti di 4 classi che ora descriveremo:

- Classe “tabellone”: questa classe si occupa principalmente della gestione del tabellone, quindi del caricamento dei file dei dadi, dell'estrazione casuale delle lettere e dell'inserimento di queste ultime nel tabellone, viene definita la struct “casella” che è composta da un char contenente la lettera estratta, e da un bool che ci dirà se in quella casella siamo già passati. I membri della classe sono un puntatore a casella “mat”, e un intero “lato” contenente la dimensione del lato della tabella
  - Attributi:
    - struct casella
      ```
      {
          char l;
          bool full;
      };
      ```
    - casella* mat;
    - int lato;

    La struttura “casella” è composta da un carattere “l” contenente la lettera di una casella del tabellone, e da un booleano “full” che serve a sapere quando il programma è già passato in una determinata casella. Il puntatore a tipi “casella” “mat” serve per poterlo poi puntare al tabellone (un array di “casella”) che verrà allocato dinamicamente. L'intero “lato” contiene la dimensione del lato del tabellone.

  - Metodi:

    - tabellone(int): riceve in ingresso la dimensione del lato del tabellone e la assegna a “lato”, crea il tabellone assegnando a “mat” un array di dimensione “lato” x “lato”, richiama le funzioni “creadadi” e “creatabellone”;

    - void creadadi(): si occupa della creazione dei dadi, vengono creati 2 array, uno per le consonanti e uno per le vocali, grazie alla funzione “checkdadi” determiniamo il numero di dadi da creare nel file “dadi.in”, vengono estratte per ogni dado 3 vocali e 3 consonanti;

- void creatabellone(): legge il file dei dadi, estrae le lettere e le inserisce nel tabellone;

- int checkdadi(): controlla se il file dei dadi è presente, se è presente controlla se sono presenti tutti i dadi e restituisce il numero di dadi mancanti;

- char* car(int): data una posizione nel tabellone restituisce la lettera corrispondente;

- ~tabellone(): dealloca la memoria assegnata dal costruttore a "mat";

- Classe "game": questa classe gestisce l'intera fase di gioco che comprende la lettura delle parole inserite dall'utente, ricerca di tutte le parole esistenti nel tabellone, conta del punteggio del giocatore e del computer e salvataggio delle parole richieste dall'utente nel file del dizionario.
  - Attributi
    - map<string, bool*> hwbool;
    - map<string, bool*> cwbool;
    - map<string, bool*> exception;

    Questi attributi sono di tipo map (un tipo contenuto nelle librerie stl) composto da una chiave di tipo stringa e da un valore mappato di tipo puntatore a booleano, e contengono rispettivamente (hwbool) le parole inserite dall'utente e in una seconda fase solo le parole corrette, (cwbool) tutte le parole trovate dal computer, (exception) tutte le parole inserite dall'utente presenti nel tabellone ma non nel dizionario. Il tipo map ci è stato molto utile in quanto consente l'inserimento ordinato, impedisce la presenza di doppioni e infine consente di associare ad ogni parola un array di booleani che consente in seguito di evidenziare le parole trovate nel tabellone.

  - Metodi:

    - void hgame(string): questa funzione data una stringa (inserita dall'utente) controlla che sia più grande di 3 caratteri, se rispetta la condizione la parola viene inserita nella lista dell'utente "hwbool";

    - void cgame(int n,tabellone &tab,dizionario* d): gestisce per così dire il turno del computer, ovvero trova tutte le parole esistenti nel tabellone e nel dizionario. Per fare ciò si scorre tutto il tabellone. Per ogni casella viene richiamata la funzione "ricorsiva" e infine vengono inizializzati i booleani del tabellone a false in modo da poter effettuare nuovamente la ricerca partendo dalla nuova casella;

    - void ricorsiva(int i,int a,char word[],int n,tabellone &tab,dizionario* d): questa funzione controlla ricorsivamente le caselle adiacenti, le caselle non utili vengono escluse tramite dei controlli; una volta che una casella supera il controllo viene fatta una ricerca nella struttura dati del dizionario a partire dalla seconda lettera in poi, questo consente di ridurre di molto il tempo per trovare tutte le parole possibili visto che vengono esclusi da subito i casi non utili. Se la

stringa non è presente nel dizionario si controlla un'altra casella adiacente, se la stringa è presente viene chiamata la “ricorsiva”, se la stringa è presente e corrisponde a una parola viene salvata nella lista del computer, al puntatore booleano viene assegnato un array di booleani contenente i valori dei booleani del tabellone e viene chiamata la “ricorsiva”;

- int punteggio(map<string, bool*> lista): questa funzione si occupa del calcolo del punteggio. Scorre la lista in ingresso, e per ogni parola ne viene misurata la lunghezza e a quest'ultima viene sottratto 3 in modo da ottenere il punteggio. Il punteggio di ogni singola parola viene sommato via via alla variabile “punti” che verrà infine restituita;

- void compare(): questa funzione controlla che le parole presenti nella lista del giocatore “cwbool” siano presenti nella lista del computer “cwbool”. Se la parola è presente viene assegnato l'array di booleani presente nella lista del computer per la parola in considerazione. Se la parola non è presente viene spostata nella lista “exception”;

- int fword(int i,int a,char word[],tabellone &tab,int n): è simile alla funzione “ricorsiva” ma con la differenza che controlla le caselle adiacenti a quella di partenza avendo già una parola in ingresso. Questa funzione è utilizzata per controllare se una parola data in ingresso è presente nel tabellone. Se la parola viene trovata restituisce 0 altrimenti restituisce 1;

- void save(const char*): salva nel file dizionario.txt le parole che l'utente desidera aggiungere al dizionario;

- void wordtab(int n,tabellone &tab): controlla che le parole presenti nella lista “exception” siano presenti nel tabellone, per fare questo si “appoggia” alla funzione “fword” alla quale passa ciascuna parola della lista. Se “fword” restituisce 0 la parola viene mantenuta in lista altrimenti viene cancellata;

- ~game(): cancella le liste “hwbool”, “cwbool”, “exception”;

- classe “interfaccia”: questa classe si occupa principalmente dell'interfaccia grafica, ovvero della gestione degli eventi del mouse, della tastiera e dello svolgimento delle varie sezioni del gioco. Abbiamo deciso di sviluppare l'interfaccia grafica in una classe apparte per avere il codice relativo all'interfaccia il più indipendente possibile dal resto del gioco, in modo da rendere facile un'eventuale sostituzione o aggiornamento di essa.

  - Attributi:

    - SDL_Event event : in questo membro della classe, ogni qual volta si verifica un evento sia del mouse o della tastiera, vengono salvate le informazioni relative ad ogni evento. Gli eventi vengono rilevati attraverso una funzione di libreria delle SDL, SDL_PollEvent(&event), che ritorna 1 se si è verificato un evento altrimenti ritorna il valore 0;

    - SDL_Surface *number : in questo membro viene memorizzata un bmp dove sono memorizzati tutti i numeri da 0 a 9;

- SDL_Surface *sfondo, *sfondo_game : in questi due attributi vengono memorizzati lo sfondo del menu principale e lo sfondo che c'è durante il gioco;

- SDL_Cursor *mouse : in questo membro vengono memorizzate tutte le informazioni inerenti al cursore, come posizione e forma del cursore;

- tabellone *t : membro di tipo puntatore a tabellone in modo tale da gestire il tabellone di gioco tramite la classe interfaccia;

- dizionario *d : membro di tipo puntatore a dizionario in modo tale da gestire il dizionario delle parole tramite la classe interfaccia.

- game *g : membro di tipo puntatore a game in modo tale da gestire le varie fasi del gioco tramite la classe interfaccia;

- char *parola : puntatore a char dove vengono memorizzate le lettere inserite da tastiera durante la partita;

- int tab_x,tab_y,lato,dim_casella : questi attributi servono per le informazioni relative alla stampa del tabellone, ovvero l'ascissa, l'ordinata, il lato del tabellone e la dimensione in pixel della casella;

- Metodi:

  - interfaccia() : costruttore che inizializza le parti fondamentali del gioco, come il dizionario, lo sfondo del caricamento e il menu principale;

  - ~interfaccia() : distruttore che dealloca la memoria allocata da alcune immagini e inoltre si occupa della chiusura dell'ambiente SDL tramite la funzione SDL_Quit();

  - void menu(int x,int y) : funzione che si occupa di stampare tutti gli elementi che compongono il menu principale alle coordinate x e y del video e di gestire gli eventi del mouse in modo tale da permettere all'utente di cliccare sulle varie scelte proposte dal menu;
  - void scorrimento(int x, int y, map<strng,bool*> lista, int delay=0) : questo metodo permette di stampare una lista di parole a video minore o uguale a 22 parole. Se le parole sono di numero superiore viene abilitato lo scroll del mouse per poter visualizzare le parole dapprima non visibili. Il parametro delay si riferisce al ritardo con cui deve essere stampata ogni parola, con un delay >0 si avrà un effetto grafico a tendina;
  - void opzioni(int x,int y) : funzione che si occupa di stampare a video il menu opzioni alle coordinate x e y, esso permette di modificare la dimensione del tabellone a tempo di esecuzione;
  - void stampatabellone(int x, int y, int delay) : metodo che permette la stampa a video del tabellone alle coordinate x e y. Delay permette di ritardare la stampa di ciascuna casella del tabellone in modo tale da avere un effetto riempimento graduale. Questo metodo si serve del metodo stampacasella;

- SDL_Surface *caricabmp(char*,int,int,int,bool bg=true,int delay=0) : questo metodo permette di caricare a video un file bmp alle coordinate x e y. Il parametro di tipo bool permette di impostare il background : se viene passato true la funzione non stamperà nessuno dei pixel di colore bianco presenti nell'immagine. Il parametro delay permette un effetto dissolvenza al momento del caricamento;
- void tastiera(int&) : durante il gioco questa funzione si occupa di salvare tutti i caratteri digitati dall'utente. Quando il carattere è "invio" la parola viene salvata nella lista delle parole inserite dall'utente che verranno poi utilizzate per confrontarle con le parole del computer e controllare i punti totalizzati dall'utente;
- void gioco() : funzione che si occupa di stampare a video tutti gli elementi del tavolo di gioco e della gestione degli eventi durante le fasi di gioco;
- void scrivinum(int c,int x,int y) : funzione che permette la stampa a video di un numero <10 alle coordinate x e y. Esso si serve dell'immagine salvata in *number;
- int punteggio (int c,int x ,int y) : funzione che, dato in ingresso il punteggio dell'utente o del computer, separa le singole cifre del numero e le passa alla funzione scrivinum in modo tale da poter scrivere il numero a video;
- void stampacasella(char* c,int cw, int ch, int x, int y, int lato, bool segna) : stampa una casella con la lettera contenuta in "c". cw è la larghezza della lettera, ch l'altezza. La casella viene stampata alle coordinate x e y. "lato" è la larghezza in pixel della casella. "segna" è un booleano che se contiene il valore false la casella viene stampata normalmente, se contiene il valore true viene contrassegnata con un altro colore. Questa scelta è stata fatta per poter visualizzare sul tabellone le parole trovate sia dal computer che dall'utente;
- void casella_testo(int x,int y, int w, int h) : consente di stampare un riquadro dove poter inserire le lettere digitate dall'utente al momento del gioco;
- void dissolvi(SDL_Surface *sur, int x, int y,int start, int end, int time) : permette di ottenere un effetto dissolvenza per l'immagine a cui fa riferimento "sur". L'immagine viene stampata alle coordinate x e y. Il parametro "start" contiene il valore della proprietà ALPHA da cui partire e "end" il valore a cui deve arrivare la consistenza dell'immagine. "time" consente di regolare la velocità della dissolvenza;
- void coloracasella(int pos,map<string,bool*> lista) : questo metodo permette di colorare le caselle del tabellone inerenti alla parola presente alla posizione "pos" in "lista";
- void riquadro(int x,int y, int color) : permette di stampare un riquadro alla posizione (x,y) di colore "color" dove saranno poi contenute le liste delle parole dell'utente e del computer;
- SDL_Cursor* init_system_cursor(const char* image[]) : questo metodo permette di inizializzare un cursore disegnando pixel per pixel la forma della freccetta utilizzando una stringa di caratteri;

- Classe "dizionario" : questa classe si occupa della gestione del dizionario, che

comprende il caricamento in ram del dizionario mediante una struttura ad albero particolare che spiegherò in dettaglio più avanti, metodi per la ricerca di stringhe per vedere se son presenti nel dizionario.

- Attributi
  - struct nodo

    ```
    {
        char l;
        bool fine;
        nodo* head;
        nodo* next;
    }*testa[26];
    ```

  - *testa[26] è un array di 26 puntatori che rappresentano la testa di 26 alberi, ognuno per ogni lettera dell'alfabeto.
  - Il campo puntatore "*head" punta ad una lista di elementi di tipo nodo che non sono altro che tutti i figli della struct a cui fa riferimento. Questo fa si che per ogni lettera presente nell'albero abbia tanti figli quante sono le lettere successive possibili. Ad esempio: ammettiamo che nel tabellone sia memorizzata solo la parola “ANDIAMO”; un nodo figlio di A avrà come lettera corrispondente la N e non avrà nessun altro figlio, ciò comporta che i figli che sono uguali a 0 non vengano nemmeno allocati.
  - Il campo *next permette di collegare assieme in una lista dinamica tutti i figli di un dato nodo. Ad esempio, se oltre alla parola ANDIAMO memorizziamo anche ABBIAMO, avremo che il puntatore *head del nodo A punterà come prima al nodo corrispondente alla lettera N, e il puntatore *next del nodo N punterà al nodo che ha la lettera B. Possiamo sintetizzare che *head serve per poter scendere di livello nell'albero e *next per poter scorrere i nodi appartenenti ad uno stesso livello.
  - "l" contiene la lettera che il nodo rappresenta.
  - Infine il campo "fine" ci dice se una sequenza di stringhe corrisponde ad una parole presente nel dizionario; ad esempio se chiediamo al nostro dizionario se esiste la parola ANDIAMO, lui andrà a a vedere se nell'ultima lettera della parola, la "O", il campo fine è true, se è true allora la parola è presente.
- Metodi
  - void crea_ramo(nodo*,const char*) : è una funzione ricorsiva che permette di inserire all'interno dell'albero dentro cui viene memorizzato il dizionario una stringa passata per parametro;
  - void crea_albero() : si occupa di caricare da file le parole del dizionario e

passarle a crea_ramo che le inserisce una per una;

- int trovastringa(char* s) : si occupa di convertire la stringa tutta con caratteri minuscoli e di passarla a tr_ric per controllare se la stringa è presente nel dizionario. La funzione ritorna 1 se la stringa è presente, 2 se la stringa è presente e forma una parola presente nel dizionario, 0 se nessuna delle precedenti si verifica;
- int tr_ric(nodo*t,char*s) : è una funzione ricorsiva che permette di visitare l'albero in modo tale da verificare se la stringa contenuta in *s è presente o no nel dizionario. La funzione restituisce 1 se la stringa è presente, 2 se la stringa è presente e forma una parola presente nel dizionario, 0 se nessuna delle precedenti si verifica;
- nodo* inschar(nodo* t, char c) : permette di inserire un nuovo nodo nell'albero. Il nuovo nodo sarà figlio del nodo puntato da "t" e verrà inserito in testa alla lista dei figli di "t";
- nodo* checkchar(nodo* t, char c) : controlla se esiste un figlio del nodo "t" che ha come carattere "c". In caso esista la funzione restituisce l'indirizzo del nodo, in caso contrario restituisce NULL;
- void inserisciparola(const char*) : permette di inserire una parola all'interno dell'albero senza interpellare il dizionario dal disco fisso;
- dizionario() : il costruttore della classe dizionario che si occupa di richiamare crea_alber() in modo tale da creare il dizionario;
- ~dizionario() : il distruttore della classe dizionario. Da solo non può distruggere tutti gli alberi interamente, si serve della funzione cancellazione(nodo*temp) richiamandola per ognuno dei 26 alberi dichiarati in *testa[26];
- void cancellazione(nodo*temp) : è una funzione ricorsiva che visita l'albero passato per parametro sino al livello più basso, una volta arrivato in fondo inizia a cancellare sino a tornare alla radice.