Esercitazione guidata 11/03/2024

Editor di testo

Partiamo dal codice dal programma `main_text_editor`.

1. Cambiamo font nello stato per usarne uno a larghezza costante `DejavuSansMono-5m7L.ttf` e cambiamone la dimensione; aggiungiamo anche una costante per la larghezza del font:

   ```
   const int FONT_SIZE = 24; const int FONT_WIDTH = 14;
   ```

2. Progettiamo la nostra struttura dati. Il testo viene memorizzato nella stessa struttura utilizzata precedentemente `std::vector<std::string> log;` all'interno dello stato. Assegniamo all'area occupata dal testo una tabella virtuale di caratteri, indicizzata con (0,0) a partire dall'angolo in alto a sinistra (prima riga, primo carattere); la prima coordinata varia con i caratteri lungo una riga e la seconda varia con le righe.

3. La finestra inquadra una porzione di tale tabella. In questa soluzione, a ogni frame disegniamo solo la porzione di testo inquadrata dalla finestra. [Sono possibli soluzioni ialternative in cui si disegna sempre tutto il testo ma se ne visualizza solo la porzione inquadrata, agendo sulla vista associata alla finestra - lo vedremo poi]. Per ricordarci la porzione di tabella visualizzata nella finestra aggiungiamo un campo allo stato: `sf::IntRect text_view;` che è un rettangolo a coordinate intere, caratterizzato da due coppie di numeri:

   - coordinate della sua origine in alto a sinistra (posizione nella tabella del primo carattere di testo visualizzato nella finestra)
   - estensione della finestra in colonne e righe della tabella.

   Questo campo andrà aggiornato ogni volta che:

   - ridimensioniamo la finestra
   - dobbiamo spostare la finestra rispetto al testo, per mantenere il cursore di scrittura visibile.

4. E il cursore? Si può trovare in qualunque parte del testo, inclusi i fine linea. Per ricordarci dove si trova, aggiungiamo un ulteriore campo allo stato: `sf::Vector2i cursor_pos;` Questo campo andrà aggiornato ogni volta che il cursore si muove, o perché l'utente sta scrivendo (o cancellando), o perché l'utente lo sposta con le freccette.

5. Modifichiamo il costruttore dello stato in modo da inizializzare `text_view` e `cursor_pos`. Per la prima, dobbiamo consultare le dimensioni iniziali della finestra e rapportarle al numero di caratteri che può contenere in larghezza e altezza; per convenzione, scegliamo di mostrare 2 caratteri in

meno in entrambe le direzioni, per poter lasciare margini attorno al testo. Quindi:

- la sua origine è in `{0, 0}`
- la sua larghezza è `window.getSize().x / FONT_WIDTH - 2`
- la sua altezza è `window.getSize().y / FONT_SIZE - 2`

Il cursore viene semplicemente posizionato a `{0, 0}`, visto che il testo è vuoto.

6. Semplifichiamo la struttura del main, spostando in una funzione apposita `doGraphics(State &gs)` tutto il codice che serve a generare la grafica, ossia tutto quello compreso tra `gs.window.clear();` e `gs.window.display();`.

7. Dentro la `doGraphics`: eliminiamo le due linee di codice che cancellano le prime righe del testo, perché ora vogliamo sempre conservarle tutte; in `logText.setPosition` sostituiamo la specifica dei pixel da lasciare come margine a sinistra e in alto con `FONT_WIDTH` e `FONT_SIZE`.

8. Aggiungiamo il cursore: bisogna disegnarlo e tenere traccia di dove si sposta (per ora solo quando l'utente scrive). Nella `doGraphics` aggiungiamo le righe necessarie per disegnare il carattere '_' in verde nella posizione in cui si trova il cursore. Il campo `cursor_pos` dello stato ci dice dove si trova nella tabella, ma noi dobbiamo convertire queste "coordinate tabella" in "coordinate finestra", combinandole con quelle della `text_view` e con le dimensioni dei caratteri:

   - la sua coordinata x è: `(gs.cursor_pos.x - gs.text_view.position.x + 1) * FONT_WIDTH`
   - la sua coordinata y è: `(gs.cursor_pos.y - gs.text_view.position.y + 1) * FONT_SIZE + 3` dove i $+1$ aservono a tener conto dei margini e il $+3$ serve ad abbassare leggermente il carattere rispetto al margine inferiore, in modo che non si sovrapponga ai caratteri stampati.

9. Nella `handle` che gestisce gli eventi `TextEntered`: in corrispondenza di un enter, variamo la posizione del cursore facendolo avanzare di una riga e portandolo alla colonna 0; in corrispondenza di un carattere stampabile, avanziamo di una posizione solo la colonna del cursore.

10. Per ora l'editor permette solo di scrivere testo di seguito, andando a capo quando si vuole. Tuttavia, se una riga di testo supera il limite destro della finestra, oppure il numero di righe di testo supera il limite inferiore, il cursore e il testo che sta scrivendo spariscono. Per mantenere il cursore dentro la vista, possiamo intervenire sul valore di `text_view`. Aggiungiamo allo stato un metodo `adjustView()` che verrà chiamato ogni volta che avvengono modifiche che potrebbero portare il cursore fuori dalla finestra. Questa funzione fa due cose:

    - dimensiona `text_view.size` in modo che corrisponda al numero di

righe e colonne che stanno nella finestra (ad esempio in corrispondenza di un resize della finestra stessa)
- dimensiona `text_view.position` in modo che gli estremi della finestra (in "coordinate tabella") contengano le coordinate del cursore.

Chiamiamo questo metodo alla fine di tutte le `handle` che fanno modifiche alla finestra o al cursore.

11. Spostamento del cursore. Riempiamo la `handle` che gestisce l'evento `KeyPrewssed` per catturare gli eventi associati alle frecce e spostare il cursore di conseguenza. Il discrimine ra gli eventi si gestisce con una cascata di `if`. Lo spostamento dipende dalla posizione del cursore:

    - spostamento in su o in giù: si sposta il cursore di una riga; la colonna resta la stessa se contiene testo nella riga di destinazione, altrimenti si arretra il cursore alla prima colonna utile
    - spostamento a destra: se il cursore si trova a fine linea, salta alla colonna iniziale della riga successiva; altrimenti si sposta in avanti di una colonna restando sulla stessa riga
    - lo spostamento a sinistra si gestisce in modo simile.

    Bisogna fare attenzione a non permettere spostamenti che porterebbero il cursore fuori dal testo attualmente presente.

12. Lo spostamento del cursore è inutile (anzi dannoso) se non lo si associa alla possibilità di intervenire sul testo nella posizione in cui si trova. Modifichiamo la `handle` che gestisce `TextEntered` per fare in modo che l'utente possa inserire testo nella posizione del cursore:

    - battendo Enter, la riga corrente viene terminata alla posizione del cursore e la sottostringa che lo segue deve essere inserita come nuova riga successiva alla corrente. Usiamo il metodo `substr` delle `string` per spezzare la stringa in due sottostringhe e il metodo `insert` dei `vector` per inserire la nuova riga
    - battendo un carattere stampabile, se il cursore si trova a fine linea semplicemente la linea si allunga (come funzionava prima, ma ora sulla linea che contiene il cursore, non sull'ultima); se invece si trova in altra posizione, si inserisce un nuovo carattere nella stringa di quella riga: a questo scopo, usiamo il metodo `insert` delle `string`.

13. Aggiumgiamo la gestione del backspace con un ulteriore `if` nella `handle` che gestisce `TextEntered`. Il meccanismo di gestione delle cancellazioni è simile a quello dell'inserimento ma funziona al contrario: si usa il metodo `erase` delle `string` per eliminare un carattere da una stringa; si usa il metodo `append` delle string per mettere sulla stessa riga il contenuto di due righe consecutive e il metodo `erase` dei `vector` per cancellare una riga.