Editeur de texte OCaml

Ewen Expuesto

Table des matières

1 Introduction

Le but est de construire un éditeur de texte. La partie graphique est gérée par le module `Graphics` d'Ocaml. Grâce à la proposition de structure de données utilisant les buffers et lignes, on peut implémenter les fonctions gérant les touches pressées sur le clavier.

J'ai commencé à implémenter les fonctions de base qui permettent le déplacement du curseur, l'insertion et la suppression d'un caractère et le saut de ligne. Dans le fichier `projet.ml`, il y a deux versions de chaque fonction. La première est l'implémentation primitive qui respecte les premières limitations fixées dans le sujet. La deuxième est celle qui améliore l'ergonomie des déplacements et des suppressions.

2 Choix de conception

2.1 Typage

Au début, je n'avais pas compris pourquoi il y avait aussi des types `buf` pour `move_left` et `move_right`. Je m'apprêtais à modifier les parties qu'il ne fallait pas modifier, pour changer en `line`, mais je me suis arrêté. C'est en comprenant pleinement la remarque qui proposait d'utiliser une combinaison de fonctions génériques et `update_with` que j'ai commencé à l'utiliser abondamment.

2.2 Suppression d'un caractère `entrée`

J'ai implémenté la possibilité de faire `backspace` et `suppr` même quand le caractère précédent ou suivant est `entrée` : cela déplace toute une ligne. Pour cela, la fonction `do_suppr` regarde s'il n'y a pas de caractères à la suite du caractère sur lequel le curseur est présent. Dans ce cas, elle concatène intelligemment la ligne courante et la ligne suivante. Elle rassemble d'abord les `before` et `after` de la ligne suivante après les avoir mis dans l'ordre de la phrase avec `List.rev`. Ensuite, la fonction `do_suppr` rassemble le tout dans une même ligne.

```
match buf.current.after with
  | [] -> (
      match buf.after with
      | [] -> buf
      | next_line :: next_rest ->
          let merged_after = (List.rev next_line.before) @ next_line.
             after in
          let merged_line = {
            before = buf.current.before;
            current = Cursor;
            after = merged_after;
            pos = buf.current.pos
          } in
          {
            before = buf.before;
            current = merged_line;
            after = next_rest;
            pos = buf.pos
          }
    )
```

Listing 1 – Extrait de l'implémentation ergonomique de la suppression avec `do_suppr`

2.3 Saut de ligne avec `move_right` et `move_left`

J'ai implémenté la possibilité de descendre à la ligne de dessous quand on va à droite à la fin d'une ligne. Puis symétriquement pour la gauche. Pour cela, on reprend l'idée précédente de merge du `before` et `after`, mais ici de la ligne suivante.

```
match buf.current.before with
  | [] -> (
      match buf.before with
      | [] -> buf
      | prev_line :: rest ->
          let merged_before = (List.rev prev_line.before) @ prev_line.
             after in
          let new_pos = List.length merged_before in
          let current_line = get_current buf in
          {
            before = rest;
            current = {
              before = List.rev merged_before;
              current = Cursor;
              after = [];
              pos = new_pos
            };
            after = {before = []; current = Cursor; after = (List.rev
               current_line.before) @ current_line.after; pos = 0} ::
               buf.after;
            pos = buf.pos - 1
          }
    )
```

Listing 2 – Extrait de code de la fonction `move_left`

2.4 Déplacement avec `move_down` et `move_up`

J'ai implémenté, de manière similaire à précédemment, la possibilité d'aller en fin de ligne quand on va en bas en fin de buffer, et symétriquement en haut dans les fonctions `move_up` et `move_down`.

2.5 Consistence du déplacement du curseur en haut et en bas

Jusque-là, lorsque l'on descend ou monte une ligne en déplaçant le curseur d'une ligne en haut ou en bas, le position du curseur pour une ligne est sauvegardée dans cette ligne. Ainsi, si on se place au début d'une ligne, que l'on descend, puis que l'on bouge à droite et enfin que l'on remonte, on sera ramené au début de la première ligne. Pourtant il est courant dans les éditeurs de texte d'avoir une position du curseur consistante au fil du déplacement le long des lignes.

```
match buf.before with
  | prev_line :: rest_before ->
      let new_after = buf.current :: buf.after in
      let new_pos = buf.pos - 1 in
      let new_current_before_rev, new_current_after = split_at (get_pos
          buf.current) ((List.rev prev_line.before) @ prev_line.after)
        in
```

```
      let new_current_before = List.rev new_current_before_rev in
      {
        before = rest_before;
        current = {before = new_current_before; current = Cursor; after
            = new_current_after; pos = get_pos buf.current};
        after = new_after;
        pos = new_pos
      }
;;
```

Listing 3 – Implémentation du déplacement ligne en ligne ergonomique extrait de `move_up`

2.6 Retour en début de ligne

Enfin j'ai implémenté le déplacement au début de la première ligne si l'on déplace le curseur vers le haut en haut en étant sur la première ligne du buffer. Et similairement en bas du buffer.

```
match buf.before with
  | [] ->
      update_with (fun line -> {
        before = [];
        current = Cursor;
        after = (List.rev line.before) @ line.after;
        pos = List.length (line.before @ line.after);
      }) buf
```

Listing 4 – Implémentation du retour en début de ligne en début de buffer dans `move_up`

3 Limites

Le programme pourrait bénéficier :

— D'une **interface graphique** plus jolies et adaptée à la taille de l'écran

— De l'implémentation de toutes les touches

— De la possibilité de l'**utilisation du curseur** pour le déplacement et la mise en surbrillance

4 Tests

Pour exécuter le fichier `test.ml`, il est possible d'utiliser l'interpréteur directement en exécutant `ocaml test.ml` dans le terminal. Il est aussi possible de le compiler avec `ocamlc -o test_file test.ml` et de l'exécuter avec `./test_file`. Ces deux méthodes permettent d'afficher dans le terminal les résultats des tests.

Les tests m'ont permis de détecter une erreur dans la fonction `move_up` qui n'influençait pas la vue graphique mais la structure de données. En effet, je mettais mal à jour la variable `pos` du curseur.

5 Manuel d'utilisation

— Version d'Ocaml conseillée : 4.08.0 car elle contient le module `Graphics` par défaut à télécharger avec `apt install ocaml`

— Ou bien toute autre version récente en installant le module `Graphics` avec `opam install graphics`

— Ouvrir un terminal et se placer dans le répertoire du projet

— Construire l'exécutable dans le terminal avec `make`, si cela ne fonctionne pas, commenter l'une des ligne 6/7 et décommenter l'autre dans le Makefile

— Lancer le programme dans le terminal : `./test`

Touches (simultanées)	Action
Control q	déplacement du curseur d'un caractère vers la gauche
Control d	déplacement du curseur d'un caractère vers la droite
Control z	déplacement du curseur d'un caractère vers le haut
Control s	déplacement du curseur d'un caractère vers le bas
Backspace	effacement du caractère précédent
Delete	effacement du caractère courant
Escape	arrêt du programme