Auteurs : Alice INVERNIZZI, Marion GUILLARD

Comparaison entre ELM et JavaScript

I. Fiabilité lors de l'exécution

ELM est un langage compilé, contrairement à JavaScript qui est un langage interprété. ELM est aussi un langage fonctionnel avec un système de typage fort. Cela permet d'avoir une gestion claire des données et de lever toutes les erreurs lors de la compilation. Ainsi, cela garantit qu'il n'y a pas de bugs lors de l'exécution (runtime). À l'inverse, JavaScript est interprété ligne par ligne, et, au bout d'un certain temps d'exécution, des comportements inattendus peuvent survenir (par exemple, des dessins incorrects). Ainsi, ELM assure que si le programme fonctionne correctement une fois, il fonctionnera toujours de la même manière.

II. Gestion des évènements

En ELM, le code se repose sur The ELM Architecture avec un modèle central (Model), des messages (Msg) pour les interactions utilisateur, et une fonction update pour gérer les changements d'état. Alors qu'en JavaScript, la gestion d'évènement se fait avec des écouteurs d'évènements. Dans notre projet, cette gestion d'interactions aurait été sujet à des erreurs de synchronisation.

III. Génération HTML

En ELM, la page HTML est générée à partir du code source. Le processus de génération se fait automatiquement lors de la compilation. La structure HTML est définie directement dans le code, et c'est ELM qui s'occupe de créer la page finale. À l'inverse, avec JavaScript, il faut commencer par une page HTML existante et y ajouter un attribut pour la lier au code JavaScript, souvent à l'aide de la balise `<script>`.

Ainsi, ELM se suffit à lui-même pour créer la page HTML et gérer la logique de l'application. Cependant, pour des pages au design complexe, cette approche peut devenir lourde. Cela peut rendre le code plus difficile à maintenir, d'autant plus que le fichier HTML généré peut devenir volumineux.

Néanmoins, une fois que la page est générée, ELM ne va mettre à jour que le canvas, ce qui évite les rendus redondants et permet d'avoir des performances optimales.

IV. Bibliothèques

JavaScript dispose d'un nombre beaucoup plus important de bibliothèques et est globalement plus performant lorsqu'il s'agit de manipuler le DOM. Par exemple, en JavaScript, il aurait été plus facile de récupérer et de transformer le champ Entry dans la structure désirée. Cependant, pour notre projet, nous n'avions besoin que de la bibliothèque Parser, qui est bien disponible en ELM. Sur ce point, ELM répondait donc parfaitement à nos besoins.

V. Conclusion

Dans ce projet ELM permet une gestion claire avec une exécution sûre et des performances optimisées. Néanmoins, Javascript aurait pu tout aussi bien être utilisé et nous aurait permis d'avoir un développement simplifié et plus flexible. Il aurait été aussi plus facile d'utiliser des bibliothèques externes, notamment pour la traduction du langage TcTurtle.