Auteurs : Titouan Verdier, Maxime Chaffraix, Rémi Duran

Comparaison entre ELM et JavasScript pour TcTurtle

Dans le cadre de ce projet, le cœur de l'application est entièrement conçu en Elm, puis compilé vers JavaScript pour un déploiement facile dans un navigateur. Nous présentons ici un bilan succinct, accompagné d'éléments de comparaison entre Elm et JavaScript.

I. Structure et Sécurité :

Elm, avec son architecture MVU (Model-View-Update), nous a forcés à structurer clairement le code (état, logique, rendu). La gestion immuable de l'état a évité des bugs liés aux modifications accidentelles. En JavaScript, une approche similaire aurait nécessité des librairies comme Redux, avec un risque accru d'erreurs manuelles.

II. Gestion des Erreurs :

Le système de types fort d'Elm (ex: Result String (List Instruction)) a rendu la gestion des erreurs de parsing explicite et sécurisée. En JavaScript, les erreurs de syntaxe dans les commandes TcTurtle auraient nécessité des try/catch et des vérifications manuelles, moins fiables.

III. Rendus SVG Dynamiques :

Elm génère le SVG de façon déclarative (display construit une liste de Svg msg). En JavaScript, nous aurions manipulé le DOM directement (createElement, setAttribute), avec un risque de performances ou d'oublis de mise à jour.

IV. Parsing :

Le module Parser d'Elm (inspiré des parseurs combinatoires) a permis de décrire la grammaire TcTurtle de façon lisible et modulaire. En JS, nous aurions probablement utilisé des expressions régulières ou un parseur manuel, moins maintenables.

V. Mesure des Performances :

La gestion des temps de traitement avec Time.now et Cmd a été naturelle en Elm grâce au système de gestion des effets. En JS, les appels asynchrones (setTimeout, Promise) auraient introduit plus de complexité et de risques de "race conditions".

VI. Débogage :

Les erreurs de compilation d'Elm (ex: filtrage incomplet des case) nous ont guidé dès l'écriture du code. En JavaScript, ces erreurs seraient passées inaperçues jusqu'au runtime, rallongeant les phases de test.

VII. Écosystème :

Elm offre des outils intégrés (compilateur, gestion de packages) mais un écosystème plus restreint. Pour des fonctionnalités avancées (ex: animations fluides), JavaScript aurait offert plus de librairies (ex: D3.js, Three.js), mais avec une intégration moins homogène.

Conclusion :

Elm s'est révélé idéal pour ce projet grâce à sa rigueur et sa clarté, minimisant les bugs et améliorant la collaboration. JavaScript aurait permis un prototypage initial plus rapide (sans compilation) mais au prix d'une maintenance accrue. Le choix dépend du contexte : Elm pour des projets structurés et éducatifs, JavaScript pour des besoins étendus ou interopérabilité. Notre expérience avec Elm a été exigeante mais gratifiante, transformant des concepts complexes en code prévisible