Comparaison entre Elm et JavaScript dans le contexte du projet

1. **Typage strict**
   Elm est un langage fonctionnel typé. Contrairement à JavaScript, où des erreurs inattendues peuvent survenir (ex. `undefined, 1=='1'`)
   Par exemple, dans Elm, le type Bounds est clairement défini dans 'DrawTcTurtle.elm':

   type alias Bounds = { xmin : Float , xmax : Float , ymin : Float , ymax : Float }

   Tandis qu'en JS, le type est définis ligne 6471 de main.js comme

   {xmax: xmax , xmin: xmin , ymax: ymax, ymin: ymin}

En JS, les types ne sont pas imposés, Bounds . `xmin="Toto"` est valide

1. **Gestion des Effets de Bord**
   Elm impose une approche purement fonctionnelle sans effets de bord. JavaScript, en revanche, permet des effets de bord n'importe où, ce qui peut rendre la recherche de bug plus complexe. Pendant tout notre projet si le programme ELM se compilait alors il fonctionnait.

2. **Parsing de commandes**
   En ELM, la syntaxe est claire. Une fois que nous avons compris comment fonctionne le module Parser alors la syntaxe est plutôt simple. Par exemple parseProgram en elm

   ```
   parseProgram : Parser (List Instruction)
   parseProgram =
       succeed identity
           |. symbol "["
           |. spaces
           |= parseInstructions
           |. spaces
           |. symbol "]"
   ```

   Tandis qu'en javascript, le programme est moins lisible, et nécessite des appelles de fonctions imbriqués

   ```
   var $author$project$ParseTcTurtle$parseProgram = A2(
       $elm$parser$Parser$keeper,
       A2(
           $elm$parser$Parser$ignorer,
           A2(
               $elm$parser$Parser$ignorer,
               $elm$parser$Parser$succeed($elm$core$Basics$identity),
               $elm$parser$Parser$symbol('[')),
           $elm$parser$Parser$spaces),
       A2(
           $elm$parser$Parser$ignorer,
           A2(
               $elm$parser$Parser$ignorer,
               A2($elm$parser$Parser$ignorer
               ,$author$project$ParseTcTurtle$parseInstructions,$elm$parser$Parser$spaces),
               $elm$parser$Parser$symbol(']')),
           $elm$parser$Parser$end));
   ```

3. **Gestion des erreurs** En ELM, il n'y a pas à se soucier de la gestion des erreurs, il n'y a pas d'effet de bord. En JS, on retrouve la structure `try catch` (ligne 6258 de main.js)

4. **Accès au DOM** En JavaScript, on manipule directement le DOM via getElementByID, En Elm, on n'accède pas directement au DOM. Tout passe par un modèle de "Virtual DOM". Par exemple, au lieu de modifier directement un élément, on met à jour le modèle, et la vue est reconstruite automatiquement

5. **Gestion des événements** Dans le JS compilé par elm , les événements sont gérés via: on`Click`, on`Input` avec des callbacks. En ELM, les événements sont gérés via le système de messages et de mise à jour. Par exemple, pour un champ texte interactif :

   ```
   input [ onInput UpdateText ] []
   ```

Conclusion Dans le cadre de ce projet, Elm offre une approche plus robuste et structurée pour gérer l'interface et l'interprétation des commandes (parser). JavaScript emmène une complexité accrue dans la gestion des erreurs et du code asynchrone (beaucoup de callback dans le code compilé pour géré les events). Le choix de ELM fait sens pour ce projet en particulier pour le parser et la gestion du DOM