Rapport du Projet ExceML
Partie 2.

Pour cette deuxième partie du projet nous implémentons une interface graphique dans un navigateur web au tableur de la partie 1 à travers les langages Ocaml et JS en interopérabilité.

J'ai trouvé que le chargement des cellules et la gestion des événements ne posait pas le problème d'un choix technique particulier, l'énoncé détaillant suffisamment les démarches à suivre. Elles nécessitaient néanmoins une bonne conpréhension de la structure qu'on souhaite mettre en œuvre, notamment la structure cellule/(input,text) dérite dans l'énoncé.

Stockage des données.

- Pour la fonction grid_to_string j'ai choisi d'appliquer un pattern matching afin d'exclure les cellules vides (que nous ne souhaitons pas transcrire dans la chaine de stockage). Dans le cas d'une case pleine, j'ai utilisé la fonction sprintf du module Printf afin de créer une chaine formatée pour représenter la cellule. (format : i|j| chaine). Cette transcription se propage sur toute la grille à travers deux Array.mapi imbriqués. Nous créons ensuite une liste de chaine de caractère avec cette ligne :

```
let s = Array.fold_left List.append [] (Array.map (fun a -> Array.fold_left acc [] a) grid_string),
```

acc étant une fonction qui ajoute un nouvel élément à une liste. Pour terminer, la fonction concat du module String m'a permis de réunir ces chaines en un seul string, (j'ai choisi dans ce cas '~' comme séparateur entre chaque représentation de case, n'ayant pas réussi à utiliser '\n'.)
- Pour la fonction cells_of string, j'utilise String.split_on_char pour découper la chaine produite par la méthode précédente de façon judicieuse. L'utilisation d'un pattern matching par la suite permet de récupérer les données facilement.
-Pour load_storage je définis une fonction interne qui charge une cellule à partir d'un triplet (i,j,s) puis je l'applique sur la liste sortante de cells_of_string. L'update des données et des liens de dépendance cellule par cellule lors du chargement d'une grille se fait dans cette fonction interne.

Evaluation d'un case.

La récupération d'une chaine pour la transformer en expression est assez explicite. Ast.make est appelée dans update_deps qui sera appelée dans update.
L'appel de Tableur.eval est assez simple. Dans mon cas, utilisant mon programme de la partie 1, j'appelle Tableur.eval_expr.

-Pour l'implémentation de result_to_string j'ai utilisé Printf.sprintf pour formater les entiers et flottants en string. Dans le cas des chaines de caractères je choisis de tronquer les chaines qui font plus de 10 caractères.

Propagation.

La mise à jour des dépendances est prise en charge par la fonction update_deps qui génère les dépendances parent en utilisant la fonction direct_deps (prédéfinie). L'utilisation de Set avant le passage en liste permet d'ajouter des éléments à un ensemble de dépendances sans avoir à tester l'appartenance. Une fois cette liste créée, on visite chacun des parents pour mettre à jour sa liste fils.
Pour la fonction propagate l'énoncé est assez explicite. Cependant lors de son implémentation j'ai choisi de couper la propagation en cas d'erreur. Dans le cas d'un cycle cela revient à ne pas mettre à jour l'affichage de toutes les cellules du cycle. (Ne pas traiter ce cas revient à créer une boucle récursive infinie tandis que mes idées d'implémentation qui prennent en compte ce problème tout en parvenant à afficher toutes les cases erreurs me semblait trop lourdes).

Conclusion.

Je pense que ce projet, en plus des cours en lignes, m'a permis de bien rester dans l'ambiance de cours, en m'incitant non seulement à pratiquer la programmation en Ocaml mais aussi à voir un cas concret d'interopérabilité entre langages. (ici ocaml et js).
Dans mon cas je constate une erreur que je n'ai pas pu corriger ainsi qu'un point que j'aurais souhaité améliorer :
- Les erreurs Mauvais_indice pour l'évaluation des cases n'est pas soulevée lorsque j'appelle des cases avec des mauvais indices.(j'obtiens à la place un « uncaught exception: 0,248,Invalid_argument,-4,index out of bounds » dans la console de mon navigateur.) Ce problème vient probablement d'une mauvaise implémentation d'eval_expr dans la partie 1 de mon projet que je n'ai pas réussi à corriger.
-Ecrire directement une chaine de caratère dans une case soulève une erreur de syntaxe. L'utilisation obligatoire des « » est dommage, surtout considérant que dans le cas B + 1, on obtient une erreur de syntaxe plutôt qu'une erreur Argument_invalide ; pour soulever cette dernière erreur il faut écrire « B » + 1.
J'aurais souhaité que toute valeur non reconnue par l'ast soit considérée comme un String, tout en détectant les erreurs de syntaxe flagrants. Ma première implémentation de Ast.make transformait d'ailleurs les erreurs de syntaxe de l'ast en Chaine(input). Cette forme d'implémentation transformait alors toutes les erreurs possibles de syntaxe en chaine de caractères, ce qui m'a forcé à abandonner cette idée

d’implémentation. Dans ce sens, un projet commun à MPIL et COMPIL pour gérer tous ces aspects dans un même projet aurait été intéressant.