Rapport de la semaine de 18/6/2018 :

Application nombre voyelles/nombre consonnes :

Voici la liste détaillée des fonctions utilisées :

JavaScript :

```
nbr_voys

nbr_carac

nbr_cons1

clicked
```

Js_of_ocaml :

```
nbr_voys : string -> int -> int -> int

nbr_space : string -> int -> int -> int

nbr_cons : string -> int

nbr_cons2 : string -> Element -> unit Lwt.t

nbr_voys2 : string -> Element -> unit Lwt.t

clicked : string -> Element -> Element -> unit Lwt.t
```

Commande de compilation :

```
$ocamlfind ocamlc voy_cons.ml -o voy_cons.byte -linkpkg -package lwt_ppx -pa

ckage js_of_ocaml -package js_of_ocaml-ppx -package lwt.unix -linkpkg

$js_of_ocaml --disable genprim  voy_cons.byte
```

Obrowser :

Avec Lwt :

```
string_input : string -> string -> stirng → (Node.t * Fragment.t)

button : string -> string -> ('a t -> 'a t) -> Fragment.t

p : string -> (Node.t * Fragment.t)

nbr_cons2 : string -> Element -> Lwt_Obrowser t

nbr_voys2 : string -> Element -> Lwt_Obrowser t
```

la commande de compilation ne change pas.

Avec Threads :

Pas de nouvelles fonctions, c'est uniquement le contenu de la fonction qui se déclenche lors du click sur le bouton qui a été modifié.

Commande de compilation :

```
ocamlc -thread unix.cma threads.cma concurrence_simple2.ml -o concurrence_s

imple
```

BuckleScripts :

```
puis : float -> float

perim : float -> float

clicked : unit -> Js.Promise.t('a)
```

la commande de compilation ne change pas.

**Il y a également deux programmes Caml l'un avec le module Lwt et l'autre avec le module Thread commentés en pièce-Jointe.

La concurrence ,la suite :

Js_of_ocaml :

La concurrence en js_of_ocaml se fait par le billet du module Lwt (voir chapitre d'avant) qui s'apprend plutôt facilement par le programmeur grâce à sa syntaxe assez compréhensible ,en revanche le passage par variables globales etc … (des effets de bord) pourrait être considéré comme un point négatif , d'autant plus qu'il n'y a pas la possibilité d'utiliser la bibliothèque Threads ou autres en js_of_ocaml.

Obrowser :

Il existe un module intitulé lwt_Obrowser en Obrowser : Il s'agit d'une adaptation de la bibliothèque Lwt dans l'environnement Obrowser.Il y a aussi la possibilité d'utiliser le module Lwt directement.Le manque de tutorat/exemples pour cette partie (concurrence) de la bibliothèque de Obrowser peut malheureusement complexifier l'apprentissage de lwt_obrowser mais le point fort de Obrowser serait l'accès à d'autres bibliothèques de threading tel que la bibliothèque Threads.

BuckleScripts :

Contrairement à ses rivaux js_of_ocaml et Obrowser, à ce jour Lwt n'est pas disponible en BuckleScript , par conséquent on n'a guère d'autre choix que d'utiliser la bibliothèque Promise de JavaScript qui peut être dérangeant pour le programmeur Caml qui n'a pas forcément le temps (ou l'envie) d'apprendre la syntaxe de cette bibliothèque JavaScript.

Les Promises en JavaScript :

Avant de continuer notre aventure dans le monde Ocaml il serait intéressent d'étudier brièvement le fonctionnement des promises en JavaScript.En effet une promise n'est rien d'autre qu'une monade qui calcul un résultat et qui en cas de succès applique une fonction sur ce résultat.

Une promise a trois états : terminée avec succès , terminée avec échec et endormie , On peut alors dire qu'une promise est l'équivalent d'une thread en Lwt.

La création d’une thread se fait en créant un objet Promise et la propriété .next est équivalente à la fonction `bind` , la propriété `resolve` correspond au calcul que la thread doit effectuer.
En ayant en tête ces propriétés basiques , on peut programmer les threads en BuckleScript (voir le fichier voycons.re)

	Js_of_ocaml	O’Browser	BuckleScript
Lwt	Oui	Oui	Non
Threads	Non	Oui	Non
Promises	Oui	Non	Oui