**Technique Avancée**

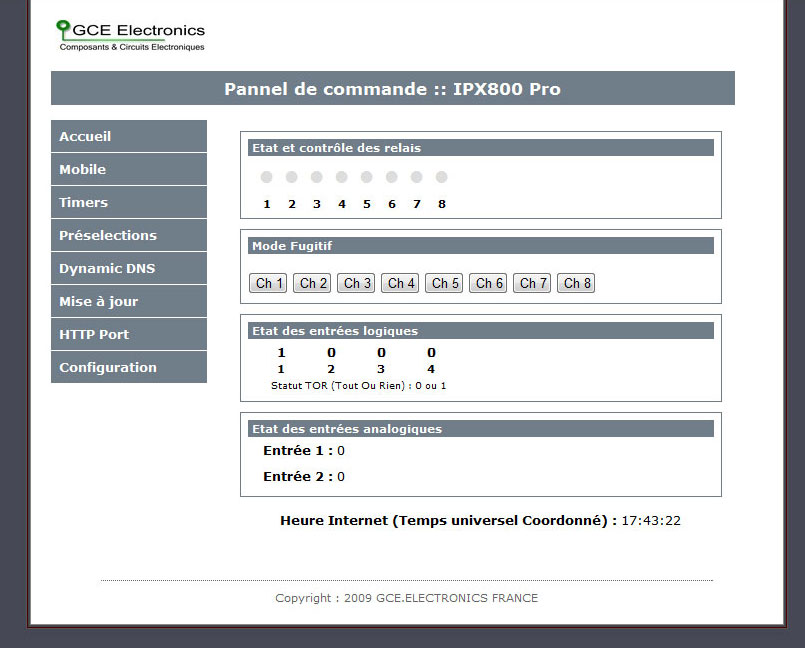
A - Exploration interne de la carte

Il a fallut dans un premier temps examiner le code existant fourni par le constructeur. Nous avons alors découvert que toutes les interactions entre l'utilisateur et le système se faisaient grâce à un navigateur, donc à des fichiers de type HTML pour la structure, CSS pour le design, JavaScript pour la dynamique et AJAX permettant d'envoyer des ordres à la carte et de rafraîchir le contenu de la page sans pour autant rafraichir la page totalement.

Certaines contraintes techniques ne sont pas énoncées par le constructeur telle que la capacité maximale de la mémoire interne à la carte qui héberge les fichier de l'interface de contrôle. C'est en cherchant sur les forum que nous avons trouvé notre réponse : 63,5 ko autant dire, pas grand chose.

Cette contrainte de taille impose donc la non-utilisation de fichiers images et de réduire au maximum le poids de tous les fichiers.

Voici donc un bref aperçu de l'interface originale fournie par le constructeur (après l'avoir traduite et avoir mis le code au propre car le code fourni ne respecte pas les standards imposés par le W3C).



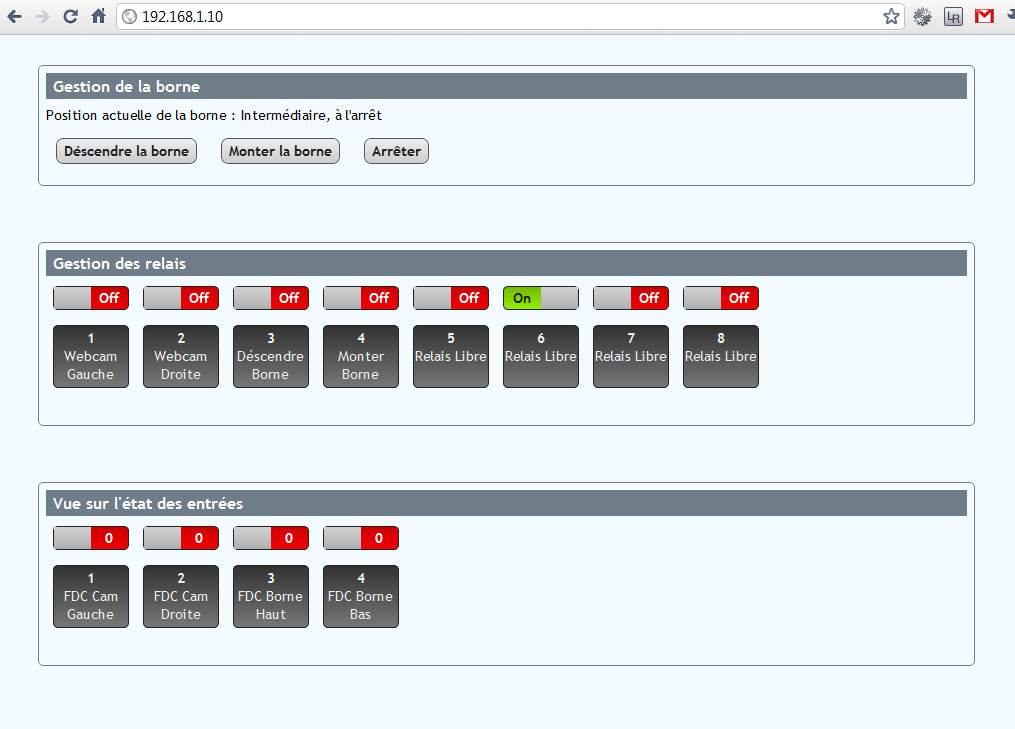
*Interface originale après restructuration et traduction*

L'interface est sobre et emplie de multiples fonctionnalités dont nous n'avons pas besoin dans ce projet. Ils sont donc retirés de l'affichage mais restent disponible dans le système de la carte.

B - Réalisation de l'interface utilisateur

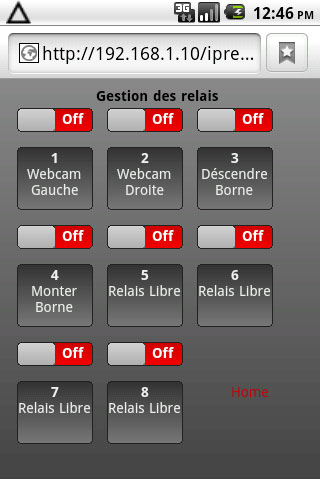
Suite à divers croquis et réflexions voici donc l'interface finale.





*Aperçu final de l'interface utilisateur depuis un ordinateur*

Une page spéciale pour les petits écrans tels que les téléphones mobiles type smartphone est accessible depuis le lien "Mobile" dans le menu du haut. Cette page permet uniquement d'agir sur l'état des différents relais.



*Version mobile de l'interface utilisateur*

C - Comment cela a-t-il été réalisé

Il est préférable pour un tel projet d'avoir de bonnes connaissances en développement web de manière à l'appliquer à ce système. Il a fallu dans un premier temps comprendre comment gérer les relais, ce qui était déjà fait initialement par le constructeur. Pour une utilisation simple et intuitive l'utilisation est la suivante : on enfonce le bouton gauche de la souris tant qu'on veux tourner, on le relâche quand on veux s'arrêter ; la flèche lorsque nous atteignons la fin de course. La gestion des évènements a été faite directement dans le code HTML grâce aux attributs *onmousedown* et *onmouseup,* ce qui correspond à : "quand le bouton gauche souris est enfoncé", "quand le bouton gauche souris est relâché". On peux comparer ces comportement à un front montant puis descendant dans un Grafcet.

Certaines actions étant à effectuer régulièrement et éviter que le code ne devienne trop imposant à force de répéter les mêmes commandes, l'utilisation de fonctions dans le code JS était idéal. Les fonctions sont comparables à des macros ; c'est une suite d'action à effectuer lors de son appel. Voici donc les fonctions utilisées dans le programme.

METTRE TABLEAU DES FONCTIONS

L'utilisation des fonctions permet notamment de reproduire le comportement d'interruption mais en JavaScript. Ainsi, la fonction *updateStatus* est exécutée toutes les 500ms grâce à l'utilisation de la fonction native *setTimeout*.

Afin d'éviter de solliciter la carte trop souvent, certaines valeurs sont enregistrées dans des variables au début de la fonction *updateStatus*. Pourquoi au début ? Tout simplement pour "l'initialisation" de celle-ci, c'est à dire qu'à la première exécution de la fonction, les valeurs des variables utilisées sont justes. Si nous avions enregistré les valeurs à la fin de la fonction, nous aurions eu 500ms de décalage possible par rapport à la réalité. Une demi-seconde suffit à abimer les composants, notamment les capteurs fin de courses qui pourraient se voir écrasé une demi-seconde de plus que nécessaire.

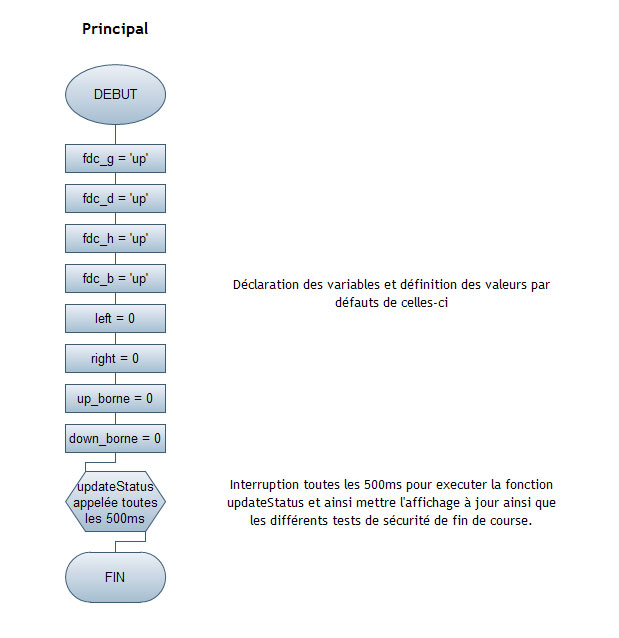
METTRE TABLEAU DES VARIABLES

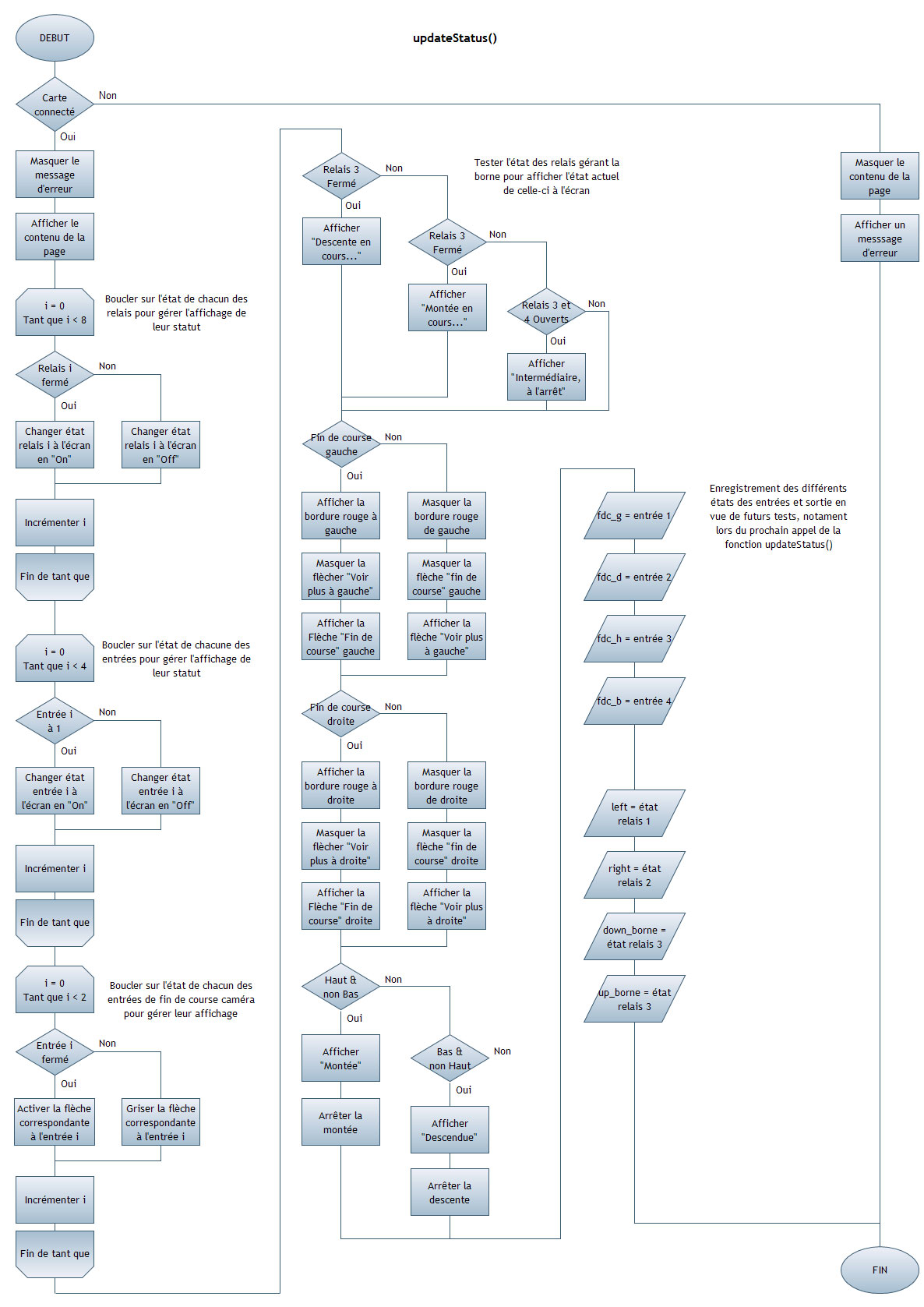
Tout a été codé "à la main". Le logiciel utilisé est Intype, car il est très léger et gratuit. Pour le design, un programme en Ruby nommé Compass a été utilisé pour le compresser et autres fonctionnalités.

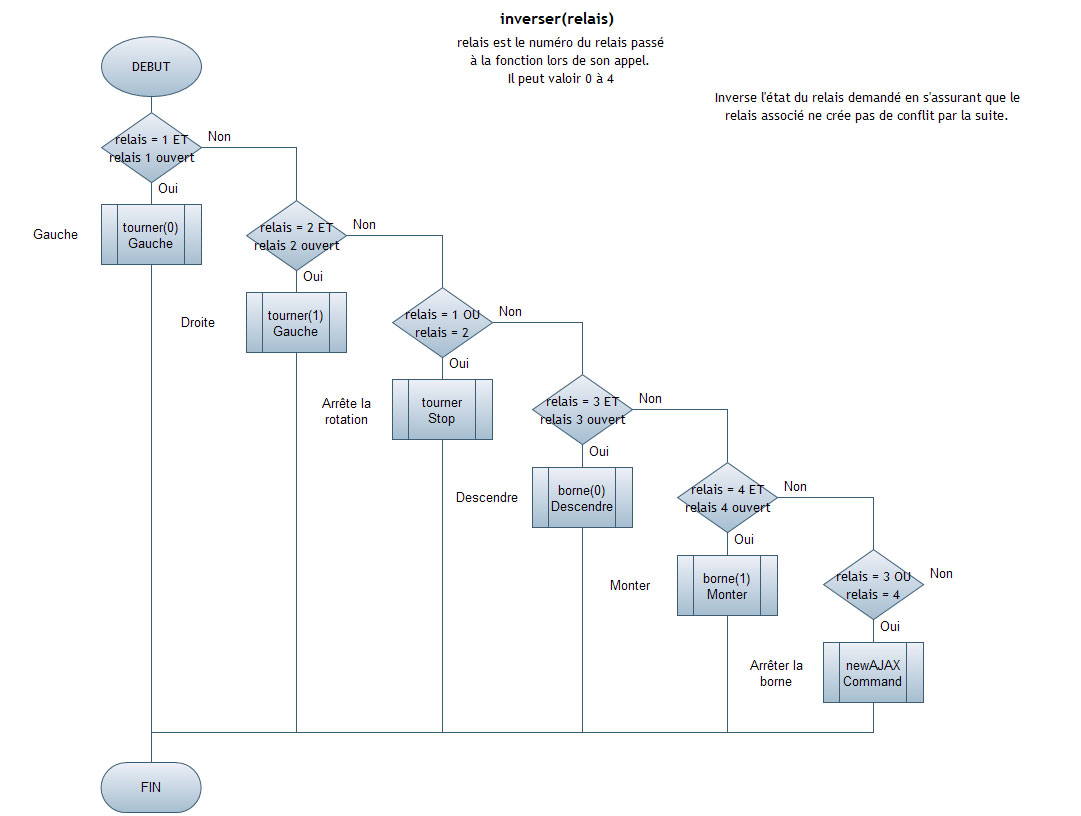
D - Structure et détails du programme

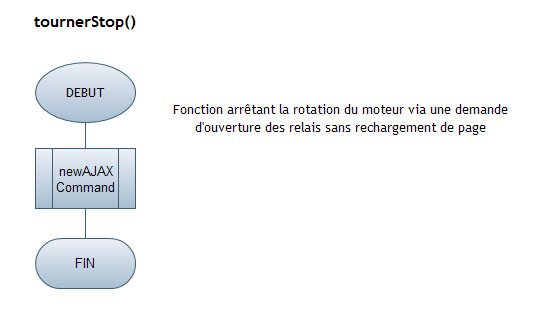
Pour avoir une lecture plus simple du programme nous avons opté pour la création d'organigrammes ayant une nomenclature semblable à celle de Flowcode. Voici donc le détail des différentes fonctions composant la dynamique de l'interface utilisateur.

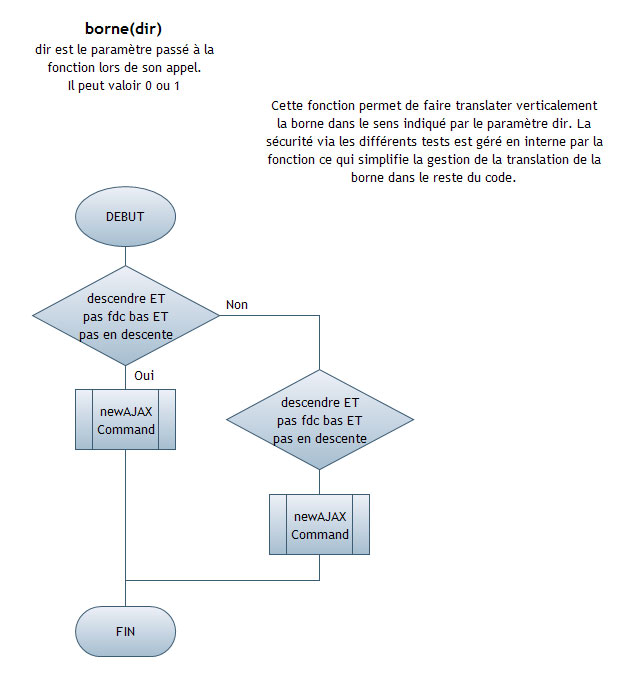
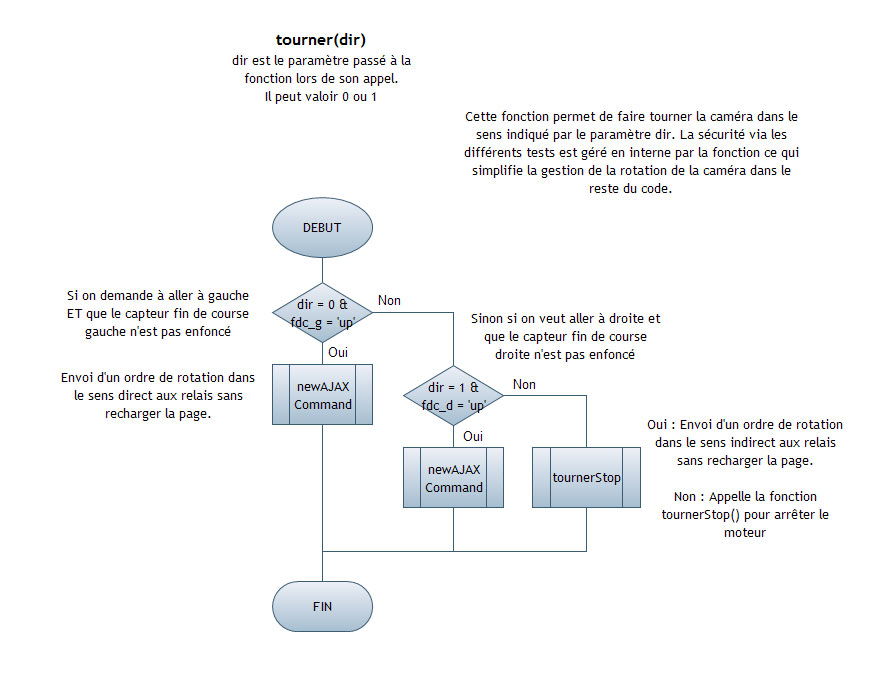
Il est divisé en 6 parties que sont les fonctions ci-dessus citées ainsi que la fonction principale qui est en réalité l'initialisation du programme, car par la suite c'est la fonction *updateStatus* qui gère tout.











E - Sécurité

Des tests de sécurité sont effectués tout au long de l'exécution du programme. Le premier test de la fonction *updateStatus* permet de savoir si la carte est toujours accessible dans le cas où celle-ci aurait perdu la connexion, un message d'erreur s'affiche et arrête l'exécution de la fonction. Un test est effectué régulièrement sur les capteurs fin de course pour pouvoir arrêter le moteur de la caméra lorsqu'il est en fin de course.

En cas de perte de connexion, le navigateur ne gère donc plus les capteurs ni les relais. On peux imaginer la situation suivante : on demande la rotation vers la droite, le relais NO droite est donc fermé. La connexion est perdue. La caméra va tourner jusqu'à écraser le capteur fin de course, ceci n'aura plus aucun effet car le navigateur ne contrôle plus le système. Il faudrait donc des capteurs "coupe-circuit" en plus des capteurs fin de course de manière à arrêter le moteur tandis que les relais NO sont fermés. Ces capteurs devraient donc pouvoir voir passer en eux une tension de la chaîne d'énergie, ou encore des capteurs avec relais associés. Ce problème majeur n'est donc pas corrigeable via la programmation car ça n'est pas la carte qui gère la sécurité mais bien le navigateur.

F - Lexique

**HTML** – HyperText Markup Language – C'est le langage qui permet de créer l'ossature de la page web et les divers éléments comme les boutons, champs texte, liens, titres, paragraphes, ...

**Balise** – C'est un élément composant la structure HTML. Par exemple pour faire un lien, nous mettrons quelque chose comme ceci :

*<a href="http://google.fr/" title="Aller sur Google">Faire une recherche...</a>*

**Attribut** – Un attribut est une sorte de paramètre pour les balises HTML. Les écouteurs d'évènements sont sous forme d'attribut : *onclick*, *onmouseup*, *onmousedown*, *onload*, ... tout comme *href* et *title* vus ci-dessus.

**CSS** – Cascading Styling Sheet – Ce langage permet la mise en forme de l'ossature crée via le HTML. Le CSS seul est complètement inutile, il doit être appelé par un fichier HTML pour remplir sa fonction. C'est grâce à ce langage que l'on peux mettre des couleurs, changer la taille du texte, la police, créer des dégradés (depuis la version 3 de CSS : CSS3), placer les éléments dans la page, ...

**JS** – JavaScript – Permet de rendre les pages dynamiques du côté du client, c'est à dire que c'est le navigateur web qui interprète ce code, tout comme le HTML et le CSS. C'est notamment en JS que tous les tests de sécurité sont effectués.

NB : il ne faut pas confondre le JS avec le JAVA qui lui est un langage dit de haut niveau.

**AJAX** – ........ – C'est en réalité la partie du JS qui permet d'effectuer des requêtes côté serveur (ici, des requêtes vers la carte) sans recharger la page, car ordinairement pour mettre à jour l'affichage, on rafraîchit celle-ci de manière à ce que le serveur régénère le code.

**Evènement** – Un clic, une pression sur une touche, un déplacement de souris, le chargement d'une image, ... toutes ces choses sont appelés des évènements. Il est possible d'écouter ces évènements. C'est à dire que l'on peux faire en sorte que "Quand on enfonce le clic gauche de la souris, on exécute la fonction tourner(1)" ce qui se code de la manière suivant, directement dans le code HTML sur le lien visé. Ceci se codera de la manière suivante :

*<a href="javascript:void(0);" onmousedown="tourner(1);">Voir plus à droite</a>*

L'attribut href étant l'adresse url vers laquelle le lien redirige, on indique que l'on va exécuter du JS et non rediriger vers une page grâce à la fonction JS : *javascript:void(0);*

**Fonction** – C'est un morceau de code que l'on peux appeler à tout moment. Celle-ci peux prendre en compte des paramètres. Par exemple, nous pouvons créer une fonction qui ajoute 1 à la valeur envoyée en paramètre, si on met 5 la fonction nous renverra 6. Ce sont les actions que l'on exécute souvent qui sont mises sous forme de fonctions, de cette manière le code est plus souple et plus simple. Si il y a un problème, il suffira de le corriger dans cette fonction et non à tous les endroits où celle-ci aurait été copié-collé. Une fonction en JavaScript se déclare comme suit :

*nomDeLaFonction(parametre1, parametre2, ...){*

*// Actions de la fonction*

*}*

**Variable** – Telle le x des maths, on lui assigne une valeur de manière à pouvoir la réutiliser dans la suite du programme. Il est également possible de la modifier au cours du programme. Pour pouvoir utiliser une variable en JS il faut la déclarer, ce qui se fait de cette manière :

*var nomDeLaVariable;*

ou avec une valeur par défaut :

*var nomDeLaVariable = 'up';*