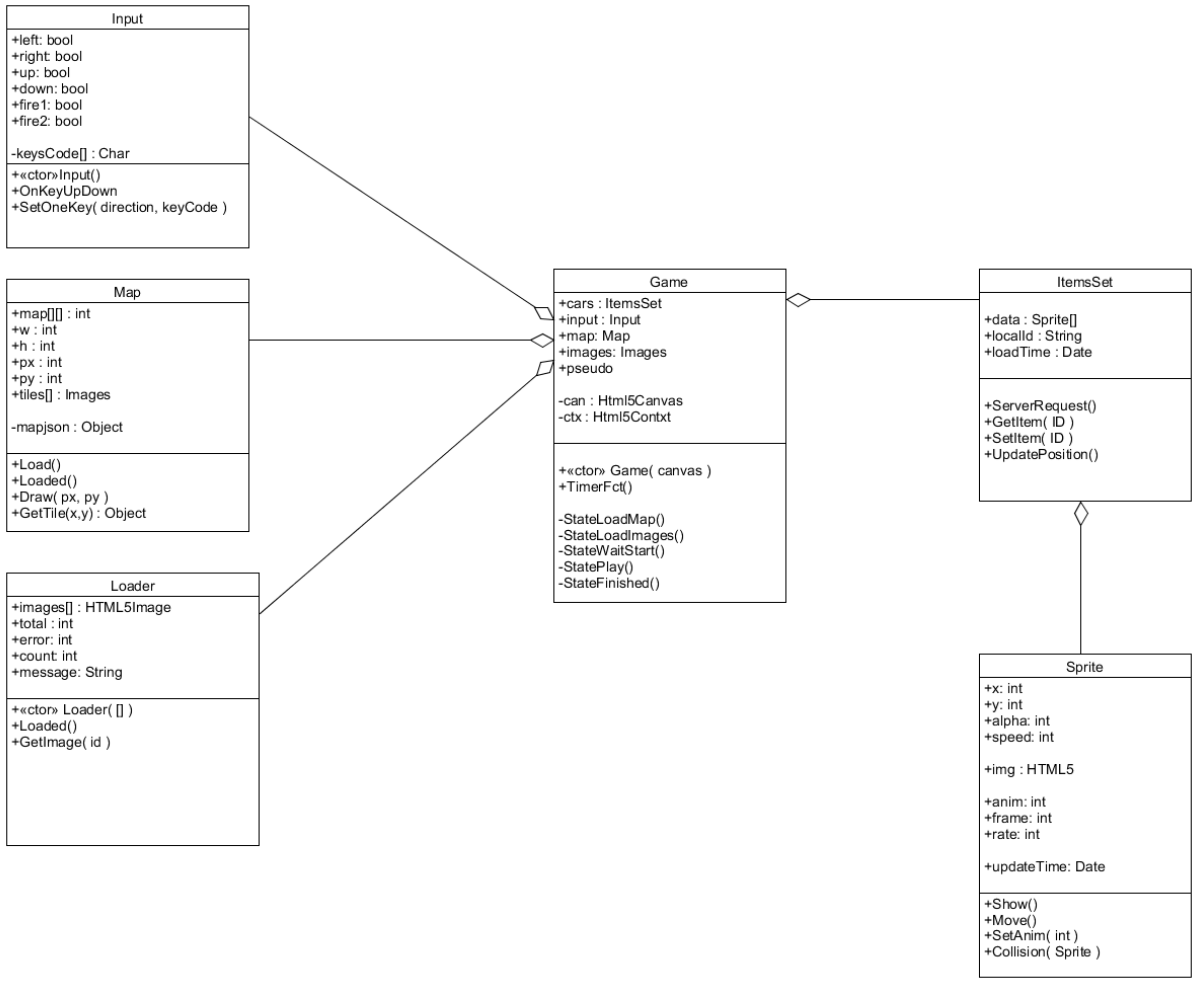
PROJET : VOITURE DE COURSE

Le but de ce projet est de créer un jeu multi-utiliasteurs en Javascript / PHP qui offre une course de voitures entre 4 participants qui se trouvent sur des machines différentes.

1.1 Diagramme de classe

Le diagramme de classe complet est le suivant :



Le détail des classes peut être obtenu dans les pages suivantes :

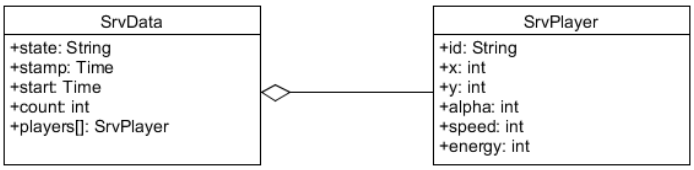
- Classe Game
- Classe Input
- Classe Map
- Classe Loader
- Classe Items
- Classe Sprite

1.2 Échange de données

quand le client envoie des données au serveur, il le fait en utilisant la méthode GET du protocole HTTP. Les paramètres à envoyer sont :

champ	Description
id	Nom du joueur (son pseudo)
x	Position x du joueur dans la map (nb décimal)
y	Position y du joueur dans la map (nb décimal)
angle	Orientation du joueur en radian (nb décimal)
speed	Vitesse du joueur (nb décimal)
energy	L'énergie du joueur (0 .. 100)

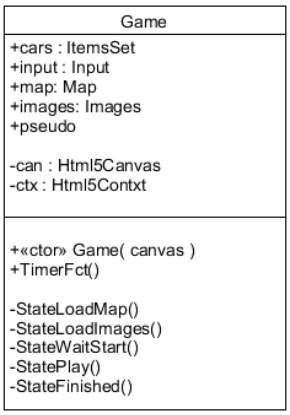
A chaque fois que le joueur aura envoyé sa position, il recevra du serveur un string JSON qui encode l'objet décrit dans le diagramme de classe suivant :



Pour éviter de saturer le serveur, les clients envoient leur poisition au plus 5 fois par seconde (200ms) mais jamais avant d'avoir reçu une réponse du serveur.

Classe Game

La classe Game est responsable du déroulement du jeu dans son entier. Elle est composée d'objets provenant de toutes les autres classes.



2.1 Les attributs

2.1.1 cars

L'attribut **cars** est un objet de la classe Items. Il permet de gérer l'ensemble des concurents de la course.

2.1.2 input

L'attribut **input** est un objet de la classe Input. Il permet de gérer les entrées de l'utilisateur pour diriger la voiture du joueur.

2.1.3 map

L'attribut **map** est un objet de la classe Map. Il permet de charger et d'afficher la carte.

2.1.4 images

L'attribut **images** est un objet de la classe Loader. Il permet de précharger les images utilisées dans le jeu et de les mettres à disposition du reste de l'application.

2.1.5 pseudo

L'attribut **pseudo** contient le pseudo du joueur local. Cette information est utilisée comme identifiant du sprite du joueur.

2.1.6 can

C'est le canvas de sortie

2.1.7 ctx

C'est le contexte de dessin en 2D.

2.2 Les méthodes

2.2.1 Game(canvas)

```
1 /** Constructeur de la classe.
2  *  @param canvas = L'ID du canvas utiliser pour afficher le jeu.
3  */
```

Le constructeur instancie tous les obets qui le compose. Il démarre aussi un timer qui exécutera toutes les 50ms (20 fois par secondes) la méthode *TimerFct()* décrite plus loins dans le document.

2.2.2 TimerFct()

Cette méthode est appellée péridoiquement par le timer de l'application. Elle s'ocuupe de :

1. Effacer le canvas
2. Appeller la méthode selon l'état (State)
3. Afficher le temps de course et l'énergie restante.

2.2.3 StateLoadMap()

Dans cet état, il faut attendre que le chargement de la carte soit terminé. Une fois terminé, il faut commencer le chargement des images.

2.2.4 StateLoadImg()

Dans cet état il faut attendre que les images soient toutes chargée. Une fois que le chargement est terminé, il passe dans l'état **StateWaitStart()**.

2.2.5 StateWaitStart()

Dans cet état, le jeu est en attente pour le début de la partie. La carte est déjà affichée ainsi que les voitures déjà inscrites.

Un message indique s'il faut encore attendre un joueur. Si tous les joueurs sont présent un compte à rebours est affiché.

Quand le décompte arrive à 0 la partie commence.

2.2.6 StatePlayGame()

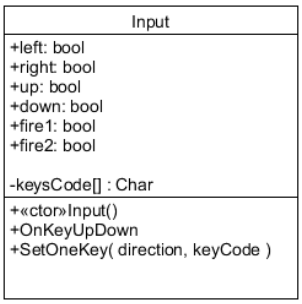
C'est le jeu. Celui-ci se termine quand le joueur local à passé la ligne d'arrivée.

2.2.7 StateFinished()

Dans cet état, le classement est affiché. Il est possible de quitter cet état sur la pression d'une touche.

Classe Input

La classe Input est responsable de gérer les entrées utilisateurs (clavier et éventuellement souris) et de mettre à disposition du reste du jeu les indications de mouvement.



3.1 Les attributs

3.1.1 left

L'attribut **left** est de type booléen (true/false). S'il vaut true alors la touche pour aller à gauche est actuellement pressée.

3.1.2 right

L'attribut **right** est de type booléen (true/false). S'il vaut true alors la touche pour aller à droite est actuellement pressée.

3.1.3 up

L'attribut **up** est de type booléen (true/false). S'il vaut true alors la touche pour aller en haut est actuellement pressée.

3.1.4 down

L'attribut **down** est de type booléen (true/false). S'il vaut true alors la touche pour aller en bas est actuellement pressée.

3.1.5 fire1

L'attribut **fire1** est de type booléen (true/false). S'il vaut true alors la touche pour accélérer est actuellement pressée.

3.1.6 fire2

L'attribut **fire2** est de type booléen (true/false). S'il vaut true alors la touche pour freiner est actuellement pressée.

3.1.7 keysCode()

C'est un tableau associatif qui relie une direction à un code de touche (keyCode dans event).

3.2 Les méthodes

3.2.1 Input()

Cette méthode est le constructeur de la classe. Elle ne prend pas d'argument. Son rôle est d'attacher les événements du clavier (et de la souris) à des méthodes de la classe pour les traiter.

3.2.2 OnKeyUpDown(event)

Cette méthode est appelée quand une touche est pressée ou relâchée et enregistre l'état de la touche.

Attention cette méthode doit gérer correctement ces événements car les navigateurs répètent les événements KeyDown quand la touche reste enfoncée.

3.2.3 SetOneKey(direction, keyCode)

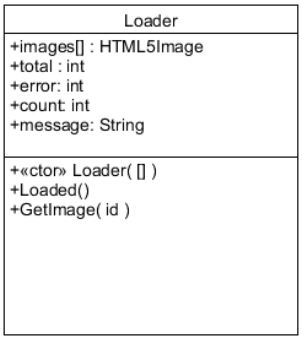
Set méthode permet de définir la touche qui est ratachée à une direction. Par exemple l'appel suivant :

```
1 this.SetOneKey( "left", "A".charCodeAt(0) );
```

Relie la touche "A" à la direction gauche. Il n'est pas possible de définir plusieurs touches pour une même direction.

Classe Loader

La classe Loader est responsable de charger toutes les images utilisées dans le jeu et de les mettre à disposition en utilisant un identifiant.



4.1 Les attributs

4.1.1 images(id)

C'est un tableau associatif qui met en relation un identifiant et une image.

4.1.2 total

Cet attribut compte le nombre d'images gérée par l'objet.

4.1.3 error

Cet attribut compte le nombre d'erreur de chargement.

4.1.4 count

Cet attribut compte le nombre d'images qui ont été chargées avec succès. Quand ce nombre arrive à la valeur de l'attribut **total** alors toutes les images ont été chargées.

4.1.5 message

Cet attribut contient une indication sur la dernière erreur de chargement survenue.

4.2 Les méthodes

4.2.1 Loader(list)

```
1 /** Constructeur de la classe. Crée les images et démarre le
     chargement.
2  *  @param list = Tableau d'objets contenant un id et une url des
       images.
3  */
```

Voici un extrait de code qui montre ce qu'il faut faire pour chaque image dans la liste reçue :

```
// this    : l'objet Loader
// listOne : Définition d'une image

var img     = new Image();
img.onload  = FctLoadFinished;  // Fonction appelée quand l'image
   est chargée
img.onerror = FctLoadError;     // Fonction appelée si erreur
img.src     = listOne.url;

this.image[ listOne.id ] = img;
```

4.2.2 Loaded()

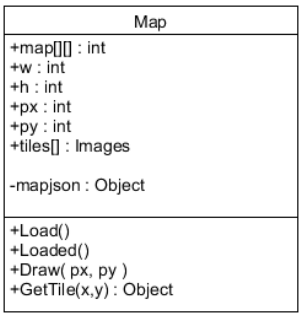
```
/** Indique si le chargement de toutes les images est terminé.
 *  @return true quand le chargement est terminé, autrement false.
 */
```

4.2.3 GetImage(id)

```
/** Tente de retrouver l'image qui correspond à l'id fournie
 *  @return L'image qui correspond à l'ID. null si l'id n'existe
    pas.
 */
```

Classe Map

La classe Map est responsable de charger la carte de façon asynchrone depuis le serveur et de l'afficher en essayant de placer le joueur local au centre de l'image.



5.1 Les attributs

5.1.1 map(y)(x)

La map est un tableau à 2 dimensions. Chaque case contient un nombre qui renseigne sur la nature de la case. Ce nombre est utiliser pour retourver les informations dans le tableau **tiles** de la classe.

5.1.2 w et h

Ces deux nombres entiers indiquent le nombre de cases que contient la carte. **w** donne la largeur et **h** la hauteur.

5.1.3 px et py

Ces deux nombres entiers indique la position de la case qui se trouve dans le coin supérieur - gauche de l'écran.

5.1.4 tiles

C'est un tableau indexé qui donne les informations suivantes :

1. L'image à afficher.
2. Est-ce que la progression est freinée.
3. Est-ce que le joueur perd de l'énergie.

C'est à dire que chaque cases de ce tableau contient un objet

5.1.5 mapjson

Cet attribut contient un objet des données reçues du serveur.

1. La carte.

2. Le nom des images des tuiles.
3. Le nom des images pour les items.

Le format de l'objet contenu dans cet attribut est détaillé dans une autre section du document.

5.2 Les méthodes

5.2.1 Load

```
/** Télécharger la carte de jeu
 *  @param url = nom du fichier contenant la carte de jeu
 */
```

Attention cette méthode démarre le téléchargement de la carte de jeu. Une fois que la carte est chargée, les données sont sauvées dans l'attributs mapjson de l'objet. Un attribut "privé" indique quand le chargement est terminé.

5.2.2 Loaded()

```
/** Indique si le téléchargement de la carte est termnié.
 *  @return true quand fini, false si le téléchargement est en
    cours.
 */
```

5.2.3 Draw(x, y)

```
/** Affiche la carte en tenant compte de la position du joueur
   local
 *  @param x, y = position du joueur local dans la grille.
 */
```

En plus d'afficher la map, cette méthode va enregistrer dans les attributs **px** et **py** de l'objet la position de la cellule affichée en haut à gauche du canvas. Ces informations sont utiles pour afficher les items à leurs bonne positions.

5.2.4 GetTile(x,y)

```
/** Retourne un objet contenant les information sur la case
   actuelle.
 *  @param x, y = position de la case à lire
 *  @return { slow : Number, energy : Number }
 */
```

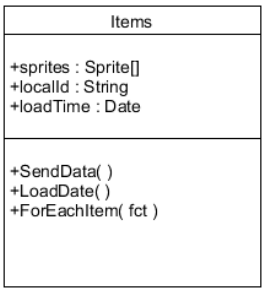
Dans l'objet retourné, **slow** indique le ralentissement du sprite (0 stopé, 1 pas de ralentissement). alors que **energy** indique la quantité d'énergie perdue par le joueur.

C'est un nombre entier compris entre 0 et 100. si la valeur est de 100, cela indique une mort immédiate du joueur.

Si la case demandée n'existe pas, la valeur retournée est null.

Classe Items

La classe Items est responsable de mémoriser l'ensemble des items (voitures) dans le jeu. Elle est également responsable de synchroniser ses données avec le serveur de jeu.



6.1 Les attributs

6.1.1 sprites()

Tableau associatif de sprite (id → sprite)

6.1.2 localId

Identifiant (pseudo) du joueur local

6.1.3 loadTime

Ticket horaire du dernier chargement des items du jeu

6.2 Les méthodes

6.2.1 SendData(local)

Envoie les informations concernant l'utilisateur local au serveur et attend en retour un tableau des nouvelles position de tous les joueurs. Met a jour les sprites avec le informations reçues

6.2.2 LoadData(json)

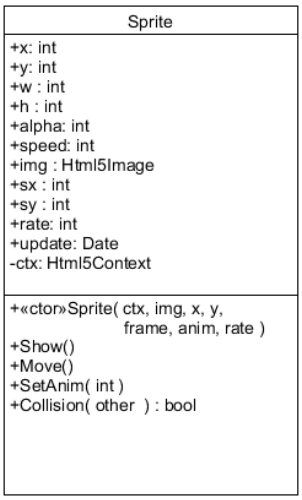
Est appelé quand les données sur l'ensemble des joueurs sont chargées.

6.2.3 ForEachItem(fct)

Appelle la fonction fct pour chaque sprite dans le tableau (Move, Show, ...). La fonction fct reçoit quand à elle deux paramètres le sprite ainsi que l'objet Items (pour les collisions).

Classe Sprite

La classe sprite est responsable d'afficher un sprite animé à la position courante en tenant compte du décalage causé par le scrolling de la Map.



7.1 Les attributs

7.1.1 x, y

C'est la position du **centre** du sprite dans la carte. Cette position est exprimée avec des nombres à virgule. La partie entière indique la cellule qui se trouve sous le centre du sprite. La partie décimale indique un décalage dans la cellule.

7.1.2 w, h

Dimension d'une image du sprite. Nous considérons que toutes les images contenues dans la table de sprites sont de mêmes dimensions.

7.1.3 alpha

C'est un angle qui défini la direction de déplacement du sprite. L'angle est exprimé en radian $[0, 2 \cdot \pi]$. Un angle de 0 correspond à un sprite dirigé vers la droite.

7.1.4 speed

Indique la vitesse de déplacement du sprite (nombre à virgule). Grâce à cette vitesse, il est possible de calculer la nouvelle position du sprite.

7.1.5 img

C'est l'image qui contient la table de sprite.

7.1.6 sx, sy

Ce sont les coordonées de la sous-image qui doit actuellement être affichée. **sx** donne l'image de l'animation actuelle. **sy** donne l'animation actuellement choisie.

7.1.7 rate

Cet attribut indique le nombre de fois qu'il faut attendre avant de passer à l'image suivante de l'animation.

7.1.8 update

L'attribut **update** contient le ticket horaire fourni par le serveur. Il indique la dernière vrai position du sprite. Elle est utilisée pour savoir s'il faut recopier les données reçues du serveur.

7.2 Les méthodes

7.2.1 Sprite(ctx, img, x, y, frame, anim, rate)

```
1 /** Constructeur de la classe.
2  *  @param ctx    = Le contexte de dessin.
3  *  @param img    = Image utilisée par le sprite.
4  *  @param x, y   = Position de départ du sprite.
5  *  @param frame  = Nombre d'images dans une animation.
6  *  @param anim   = Nombre d'animations dans le sprite.
7  *  @param rate   = Vitesse de l'animation.
8  */
```

Le paramètre `frame` est utilisé pour calculer la largeur d'une image (frame). De la même façon, `anim` est utilisé pour calculer la hauteur d'une image.

7.2.2 Show(ox, oy)

```
1 /** Afficher le sprite en tenant compte de la Map.
2  *  @param ox, oy = décalage de la position du sprite
3  *                  pour tenir compte de la position de
4  *                  la map.
5  */
```

7.2.3 Move()

```
1 /** Déplacer le sprite.
2  */
```

7.2.4 SetAnim(num)

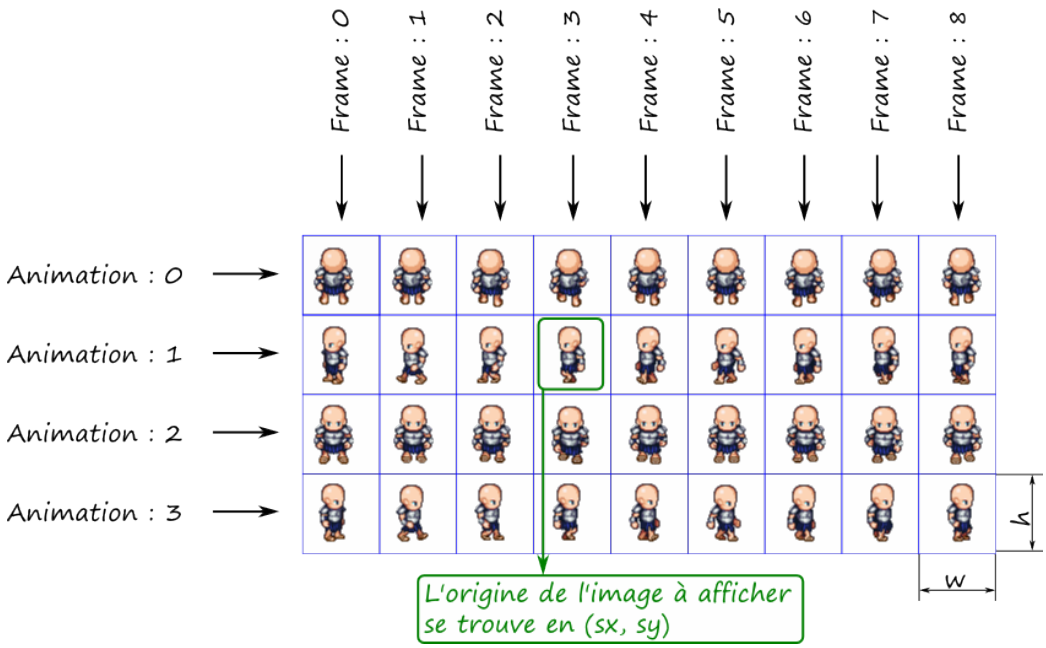
```
/** Changer l'animation courante du sprite (choix de la ligne)
 *  @param  num = numéro de l'animation
 */
```

7.2.5 Collision(other)

```
/** Détecter la collision avec un autre sprite
 *  @param  other = l'autre sprite
 *  @return true s'il y a collision. false autrement.
 */
```

7.3 Principe de fonctionnement

Une des méthodes utilisée pour donner l'impression d'avoir un sprite animé est d'utiliser une image qui contient toutes les parties de l'animation. C'est un planche de sprite (ci-dessous).



Pour simplifier la programmation, la planche de sprite est organisée de la façon suivante :

Une ligne contient toutes les images d'une animation. Par exemple pour le déplacement en haut, à gauche, en bas ou à droite. Dans une ligne, les cases représentent les frames (étapes d'une animation).

De plus, toutes les sous-images ont la même taille (w et h) de cette façon, pour passer d'une frame à la suivante, il suffit de calculer la position X suivante avec la formule :

```
this.sx = (this.sx + this.w) % this.img.width;
```

De même quand l'animation est changée, la position sy est recalculée selon :

```
this.sy = (animation * this.h) % this.img.height;
```