CBNA SmartLight

Documentation - Version GMS
16 Août 2013
http://www.lecbna.org

Concepts, code et documentation: Bastien Wild *alias* ***Bast***

Vous pouvez me joindre aux adresses suivantes:
bast@lecbna.org
le.bast@gmail.com

Légende couleurs:
Bleu : variable dont la valeur peut être modifiée par l'utilisateur
Vert : variable automatique, lecture seulement
Rouge : fonction
Jaune : script

1 - Caractéristiques

CBNA SmartLight est un moteur graphique de gestion de lumières et d'ombres écrit en GML pour le logiciel *GameMaker*. Conçu comme un système multi-usages disposant d'un large éventail de fonctionnalités, et axé sur la facilité d'accès et l'ergonomie, il donne au développeur sous *GameMaker* des outils de qualité pour la création d'effets visuels élaborés et variés.

> Lumières et ombres dynamiques

Créez facilement des lumières dynamiques projetant des ombres en fonction de leur environnement.

- Pas de polygones
Nul besoin de passer par une configuration préalable des objets destinés à projeter une ombre ni de définir des vertices pour chacun d'eux: contentez-vous de créer vos lumières, les ombres seront automatiquement et instantanément calculées à partir des sprites, quelles que soient leurs formes.

- Ombres douces en temps réel
Les ombres sont lissées, et également atténuées proportionnellement à leur distance par rapport à la source lumineuse, de même que dans la réalité.

- Éclairage volumétrique des objets
Le système d'atténuation des ombres permet un éclairage partiel des objets se trouvant dans le rayon de la lumière, en permettant à cette dernière de filtrer au travers des sprites.

- Vitesse de rendu indépendante du nombre d'instances
Le calcul des lumières n'est que peu affecté par le nombre d'instances pour lesquelles une source de lumière doit projeter des ombres, grâce à son système de buffers.

- Facilité de création de lumières pré-calculées
Créez un nombre illimité de lumières et ombres pré-calculées sans le moindre impact sur les performances, à l'aide d'une simple variable.

- Personnalisation
Personnalisez facilement et en temps réel des caractéristiques telles que la texture, la couleur, l'intensité lumineuse, la direction, la portée de la lumière, etc.

- Lumières basiques
En plus du système de lumières avec ombres dynamiques, *SmartLight* dispose également de fonctions conventionnelles permettant d'effectuer de simples affichages de sprites, afin de créer des effets de lumière à moindre coût.

> Cycle jour/nuit

Prise en charge de l'horloge et de l'avancement du temps, ainsi que des différentes variantes d'éclairage et d'ombrage à chaque heure du jour et de la nuit.

- Gestion complète du rapport luminosité globale / lumières dynamiques / ombres
Les différents systèmes offerts par *SmartLight* sont unifiés afin de permettre une gestion globale et visuellement crédible des lumières et ombres.

- Personnalisation
Configurez aisément les ombres, valeurs de luminosité et de couleur pour chaque heure du jour et de la nuit, à l'aide d'un éditeur spécialement conçu pour *SmartLight*.

> Ombres solaires

Le système d'ombres solaires de *SmartLight* permet de simuler des ombres projetées par le soleil. Le système est associé au cycle jour/nuit afin de permettre une variation de l'apparence des ombres en fonction de la position du soleil aux différentes heures du jour.

- Pas de configuration préalable
De même que les lumières dynamiques, les ombres solaires peuvent être calculées automatiquement à partir des sprites des objets.

- Gestion de la profondeur
Le rendu des ombres intègre un système de layers permettant aux objets de projeter leurs ombres les uns sur les autres en fonction de leur profondeur, améliorant la crédibilité de l'image.

- Prise en charge complète du cycle jour nuit
Les différentes caractéristiques des ombres solaires sont associées au système jour/nuit afin d'offrir un rendu réaliste et crédible. L'apparence des ombres varie tout au long de la journée selon la trajectoire et la puissance du soleil.

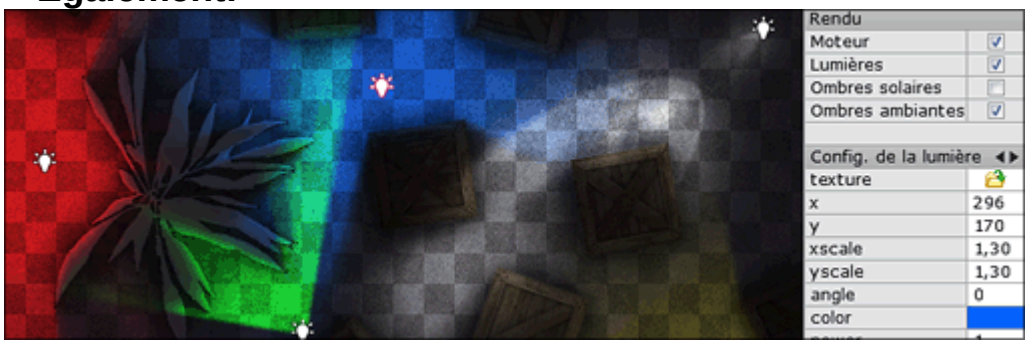
- Personnalisation
Le rendu des ombres est entièrement configurable: longueur, transparence, direction, etc.

> Ombres ambiantes

Créez facilement des effets d'ombres douces résultantes de l'illumination globale.

- Nette amélioration du visuel d'une scène pour un coût en performances négligeable
En associant à chaque texture une ambient-map pré-calculée automatiquement et ensuite intégrée dynamiquement au rendu des lumières, le visuel du jeu est grandement enrichi sans pour autant ralentir le framerate.

> Également:



- Système modulable
N'activez et n'utilisez que les éléments dont vous avez besoin. Le système ne calculera que les éléments choisis par l'utilisateur. De même, ne modifiez que les variables dont vous avez besoin: les variables non déclarées par l'utilisateur seront automatiquement gérées par *SmartLight*.

- Performances et rendu entièrement configurables
Nombre d'opérations par seconde, taille des textures, nombre de surfaces: tous les rouages de *SmartLight* sont facilement modulables afin d'offrir les meilleures possibilités d'optimisation.

- Éditeur
Créez, modifiez et pré-visualisez l'ensemble des lumières et ombres en temps réel grâce à un éditeur spécialement conçu pour ce système.

- Facilité et praticité
Le système a été pensé pour être facile, rapide et pratique à utiliser, que ce soit pour les utilisateurs expérimentés ou ceux venant tout juste de découvrir le GML.

2 - Objets

La logique de fonctionnement de *SmartLight* comprend deux types d'objets qu'il sera nécessaire d'ajouter à votre .gmk: l'objet *engine* et le ou les objet(s) *light*. Il vous est en pratique possible de nommer ces objets comme bon vous semble, les termes *obj_engine* et *obj_light* n'étant utilisés qu'à titre indicatif. Ce paragraphe présente la logique d'utilisation des fonctions et variables de base dans les différents événements de ces dits objets.

Note: Il vous faut obligatoirement faire usage d'une view afin que le système puisse fonctionner. Vous pouvez déterminer l'index de la view à utiliser avec la variable *global.sl_viewid*.

- L'objet *engine*

S'il ne vous faut créer qu'un seul objet afin de permettre le fonctionnement de *SmartLight*, c'est celui-là: il centralise la majeure partie des variables du système, et se charge de l'ensemble du rendu et de l'affichage. Il gère le cycle jour/nuit, ainsi que les systèmes d'ombres solaires et d'ombres ambiantes (les variables relatives aux lumières dynamiques, quant à elles, sont gérées localement par les objets *light* eux-mêmes). Assurez-vous que la variable ***depth*** (profondeur) de l'objet *engine* soit inférieure à celle des objets sur lesquels les effets d'ombres et lumières doivent être affichés.

Ses événements sont structurés comme suit: initialisation, puis rendu, puis affichage, et destruction si nécessaire. Les fonctions sont ici présentées dans l'ordre dans lequel elles doivent être exécutées (par exemple, initialiser l'heure avant d'avoir initialisé le ToD provoquerait une erreur). Dans tous les cas, les variables doivent être déclarées avant l'exécution des scripts, à l'exception de *SL_engine_ini_begin* qui doit être exécuté avant tous les autres scripts et variables.

Note 1: les rendus des ombres solaires et ombres ambiantes sont désactivés par défaut, il vous faudra donc les activer afin de pouvoir en faire usage. De même, la variable *sl_timespeed* est par défaut à 0: il vous faut déclarer une valeur pour que l'heure avance.

Note 2: il n'est pas nécessaire de déclarer toutes les variables disponibles pour que le système fonctionne: les variables non existantes seront automatiquement déclarées avec des valeurs par défaut lors de l'initialisation.

Create / Game Start / Room Start

```
// Initialisation de l'objet engine //

SL_engine_ini_begin(); // Commence l'initialisation du système

sl_sunshadows_active = true; // Active le rendu des ombres solaires
sl_ambientshadows_active = true; // Active le rendu des ombres ambiantes
sl_timespeed = (1/60)/room_speed; // Détermine la vitesse de l'horloge

// Génération des ambient maps
SL_sprite_set_ambient(spr_character);
SL_sprite_set_ambient(spr_tree);

// Liste des objets projetant une ombre globale
SL_global_cast_obj(0,obj_character,-1,1,1);
SL_global_cast_obj(1,obj_tree,spr_tree_shadow,2,1);

// Liste des objets lumière
SL_add_light(obj_light_01);
SL_add_light(obj_light_02);

SL_ToD_01(); // Initialisation du ToD
SL_set_time(13.5); // Initialisation de l'heure

SL_engine_ini_end(); // Termine l'initialisation du système
```

Begin Step / Step / End Step

```
// Rendu de l'ensemble des ombres et lumières //
SL_engine_render();
```

Draw

```
// Affichage de l'ensemble des ombres et lumières //
SL_engine_draw();
```

Destroy

```
// Libère la mémoire utilisée par l'ensemble des surfaces //
SL_engine_free();
```

- L'objet *light*

Le ou les objet(s) *light* (vous pouvez en créer plusieurs) permet de gérer les lumières dynamiques. Le rendu des ombres est totalement automatisé: il suffit de configurer quelques variables dans l'objet *light* lui-même, et *SmartLight* se charge du reste. En soi cet objet ne fait rien de plus que gérer des variables: le rendu de la lumière, lui, est effectué par l'objet *engine*. Il est donc techniquement possible de gérer ces variables à partir de n'importe quel objet, sans forcément user d'un objet *light* dédié.

Contrairement à l'objet *engine*, il n'est pas obligatoire de créer un objet *light* pour que le système puisse fonctionner. Par exemple, vous pouvez très bien faire usage uniquement du cycle jour/nuit et rien d'autre: aucun bug ne naîtra de l'absence d'objet *light*.

Les événements sont structurés très simplement: initialisation, et destruction si nécessaire. Rien de plus.

Note 1: Dans la version GMS du moteur, le système ne reconnaît pas automatiquement les objets *light*: ils vous est donc nécessaire de les déclarer manuellement avec la fonction *SL_add_light(object)*.

Note 2: Il est possible de créer autant d'instances que voulu à partir du même objet *light*.

Note 3: De même que pour l'objet *engine*, il n'est pas requis de déclarer toutes les variables disponibles pour que le système fonctionne. Cependant, une seule variable doit obligatoirement être déclarée: *sl_light_texture*.

Create

```
// Initialisation de l'objet light //

SL_light_ini_begin(); // Commence l'initialisation de l'objet light

sl_light_texture = spr_light; // Index de la texture de lumière (obligatoire)

// Liste des objets pour lesquels la lumière doit projeter une ombre
SL_light_cast_obj(obj_character,-1);
SL_light_cast_obj(obj_tree,-1);

SL_light_ini_end(); // Termine l'initialisation de l'objet light
```

Destroy

```
// Libère la mémoire utilisée par les surfaces de l'objet //
SL_light_free();
```

3 - Variables - Vue d'ensemble

Voici une liste de l'ensemble des variables et fonctions composant le système de *SmartLight*.

Objet *engine*

sl_tod[x1,x2]
sl_tod_active
sl_timespeed

sl_buffer_texturesize
sl_buffer_sync
sl_buffer_xmargin
sl_buffer_ymargin

sl_sunshadows_active
sl_sunshadows_texturesize
sl_sunshadows_quality
sl_sunshadows_alpha
sl_sunshadows_alphalimit
sl_sunshadows_margin
sl_sunshadows_refresh
sl_sunshadows_refreshrate
sl_sunshadows_layerscale[x]

sl_ambientshadows_active
sl_ambientshadows_alpha
sl_ambientshadows_mapscale

sl_maxexposure

sl_tod_index
sl_ambient_color

sl_layers_count
sl_layers_surface[x]
sl_global_list[x1,x2]

sl_sunshadows_list[x1,x2]
sl_sunshadows_texlist[x1,x2]
sl_sunshadows_refreshcounter
sl_sunshadows_direction
sl_sunshadows_length
sl_sunshadows_light
sl_sunshadows_surface1[x]
sl_sunshadows_surface2

sl_ambientshadows_lock
sl_ambientshadows_map[x]
sl_ambientshadows_list[x1,x2]
sl_ambientshadows_surface

sl_view_xprevious
sl_view_yprevious
sl_view_xspeed
sl_view_yspeed

sl_buffer_width
sl_buffer_height
sl_buffer_surface1
sl_buffer_surface2

Objet *light*

sl_light_active
sl_light_texture
sl_light_x
sl_light_y
sl_light_xscale
sl_light_yscale
sl_light_angle
sl_light_color
sl_light_power
sl_light_ambientpower
sl_light_shadowlength
sl_light_shadowfactor
sl_light_shadowsharpness
sl_light_castshadow
sl_light_refresh
sl_light_refreshrate

sl_light_shadowlist[x1,x2]
sl_light_refreshcounter
sl_light_visible
sl_light_cast
sl_light_surface

Globales

global.sl_viewid *(à déclarer de préférence dans l'objet engine)*
global.sl_z[obj_id]

global.sl_time
global.sl_ambient_light
global.sl_light_txsize
global.sl_light_gbuffer
global.sl_castlist[x1,x2]
global.sl_castlist_index
global.sl_texlist_light[x1,x2]
global.sl_texlist_light_index
global.sl_texlist_shad[x1,x2]
global.sl_texlist_shad_index
global.sl_lightlist[x]

Fonctions

SL_light_cast_obj(object,mask)
SL_global_cast_obj(layer,object,sprite,sun,ambient)
SL_cast_sprite(layer,castsun,castamb,castlights,sprite,subimg,x,y,z,xscale,yxscale,rot,alpha)
SL_add_light(object)
SL_draw_sprite_light(sprite,subimg,x,y,xscale,yscale,rot,color,alpha)
SL_draw_sprite_shadow(layer,sprite,subimg,x,y,xscale,yscale,rot,alpha)
SL_sprite_set_ambient(sprite)
SL_set_time(time)

3.1 - Variables - Objet *engine*

L'objet *engine*, centre névralgique du système de *SmartLight*, comprend de nombreuses variables importantes qui sont parfois amenées à être appelées à partir d'autres objets.

sl_tod[a,b]

Tableau déterminant les différentes composantes de la lumière ambiante ainsi que la forme des ombres projetées par le soleil aux différentes heures du cycle jour/nuit. L'ensemble de ce tableau est appelé *ToD* ("*Time of Day*"), et se trouve préférablement dans un script à part, exécuté lors de l'initialisation de l'objet *engine*.

Ce tableau est structuré selon la forme suivante:

sl_tod[a,0]	Heure
sl_tod[a,1]	Taux de rouge
sl_tod[a,2]	Taux de vert
sl_tod[a,3]	Taux de bleu
sl_tod[a,4]	Facteur de luminosité (0 noir complet, 1 lumière complète)
sl_tod[a,5]	Direction des ombres du soleil (en degrés)
sl_tod[a,6]	Longueur des ombres du soleil (en pixels)

La variable ***a*** correspond à l'index du segment contenant les valeurs (***a*** > 0). Un minimum de deux segments, chacun disposé à une extrémité de l'horloge (heure 0 et heure 24) est nécessaire pour que le système puisse fonctionner.

Note 1: Même si cette liste est présentée comme étant modifiable manuellement, son édition ne se fait pas directement, mais de manière automatique à l'aide de l'éditeur fourni avec *SmartLight* (*Cf* paragraphe **6 - Éditeur**).

Note 2: Il vous est possible de changer de *ToD* en cours de jeu: mais il sera alors nécessaire de réactualiser l'heure avec la fonction *SL_set_time*, comme ceci:

```
SL_ToD_02(); // Exécute le script contenant le nouveau ToD
SL_set_time(global.sl_time); // Réactualise l'heure
```

sl_tod_active

Active/désactive la gestion de la luminosité et des ombres solaires par le système de cycle jour/nuit (activé par défaut). Si désactivé, les variables *global.sl_ambient_light*, *sl_ambient_color*, *sl_sunshadows_direction* et *sl_sunshadows_length* sont alors déverrouillées et peuvent être modifiées sans passer par le ToD. Valeur booléenne (true / false).

sl_timespeed

Vitesse d'avancement de l'heure. Elle est incrémentée à chaque step à la variable *global.sl_time*. Elle ne doit pas être négative.

Exemples:

```
sl_timespeed=(1/60)/room_speed; // L'heure avance d'une minute à chaque seconde.

sl_timespeed=(1/60)/(room_speed*2); // L'heure avance d'une minute toutes les deux secondes.

sl_timespeed=0: // L'heure n'avance pas.
```

sl_buffer_texturesize

Facteur de taille du buffer de lumière, basé sur la taille de la view dans laquelle les effets de lumière sont affichés (cette view étant définie par *global.sl_viewid*).

Par exemple: avec un facteur de 1, la taille du buffer sera égale à celle de la view; avec un facteur de 0.5, la taille du buffer sera la moitié de celle de la view.

La valeur doit être définie lors de l'initialisation de l'objet *engine*, et ne peut plus être modifiée par la suite.

sl_buffer_sync

Active/désactive la synchronisation entre l'affichage du rendu et la view, permettant de compenser le mouvement de la view et d'éviter un désagréable effet de 'retard' dans l'affichage. Valeur booléenne (true / false).

sl_buffer_xmargin
sl_buffer_ymargin

Variables relatives à la synchronisation affichage/view, permettant de définir une marge autour du buffer afin d'amortir les mouvements de la view.

Exemple:

```
sl_buffer_xmargin=view_hspeed[0]+5;
sl_buffer_ymargin=view_vspeed[0]+5;
```

sl_sunshadows_active

Détermine si le rendu et l'affichage des ombres solaires sont effectués ou non. Valeur booléenne (true / false).

sl_sunshadows_texturesize

Facteur de taille de la surface contenant les ombres, basé sur la taille de la view dans laquelle les ombres sont affichées (cette view étant définie par *sl_sunshadows_viewid*).

Par exemple: avec un facteur de 1, la taille de la surface sera égale à celle de la view; avec un facteur de 0.5, la taille de la surface sera la moitié de celle de la view.

La valeur doit être définie lors de l'initialisation de l'objet *engine*, et ne peut plus être modifiée par la suite.

sl_sunshadows_quality

Facteur déterminant le nombre de samples composant les ombres projetées dynamiques. Il est basé sur la valeur de longueur des ombres lors du rendu par l'opération suivante:

```
sl_sunshadows_quality*global.sl_sunshadows_length
```

Ainsi par exemple:
Une ombre longue de 400 pixels avec un facteur de 0.5 sera composée de 200 samples.
Une ombre longue de 300 pixels avec un facteur de 1 sera composée de 300 samples.
Une ombre longue de 600 pixels avec un facteur de 0.3 sera composée de 180 samples.

Rappel: la longueur des ombres est définie dans l'objet *engine* à partir du tableau *sl_tod[a,6]*.

Exemple:

```
sl_sunshadows_quality=0.25;
```

sl_sunshadows_alpha

Facteur de transparence des ombres solaires (0 invisible, 1 parfaitement visible). Il multiplie la variable *global.sl_ambient_light* lors du rendu.

Ainsi par exemple:
Pour global.sl_ambient_light=1 et sl_sunshadows_alpha=0.5, l'alpha final sera de 0.5.
Pour global.sl_ambient_light=0.5 et sl_sunshadows_alpha=0.5, l'alpha final sera de 0.25.
Pour global.sl_ambient_light=0 et sl_sunshadows_alpha=0.5, l'alpha final sera de 0.

De ce fait, la visibilité des ombres est proportionnelle à la luminosité du soleil.

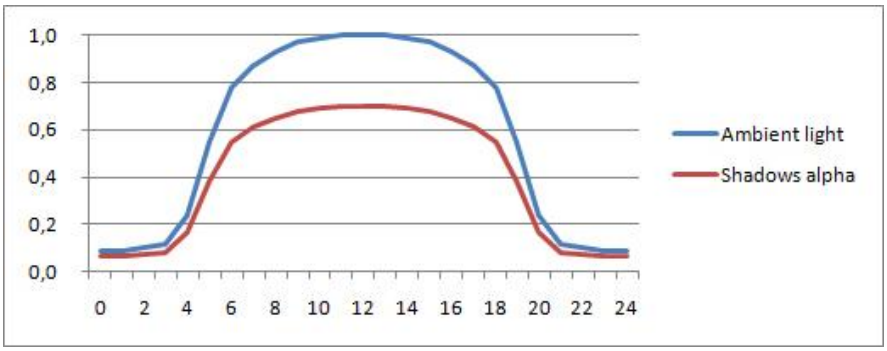
sl_sunshadows_alphalimit

Permet de définir une valeur de luminosité minimum en dessous de laquelle les ombres solaires ne seront pas visibles. Elle complète la variable *sl_sunshadows_alpha* en offrant plus de précision dans la relation entre l'alpha des ombres et la luminosité du soleil.

En pratique, cette variable est née de la nécessité de rendre les ombres solaires invisibles sans pour autant que la luminosité ambiante soit à 0. Dans la réalité, lors par exemple d'un coucher de soleil, les ombres projetées directement par le soleil ne sont plus visibles, alors que la luminosité ambiante est elle encore importante.

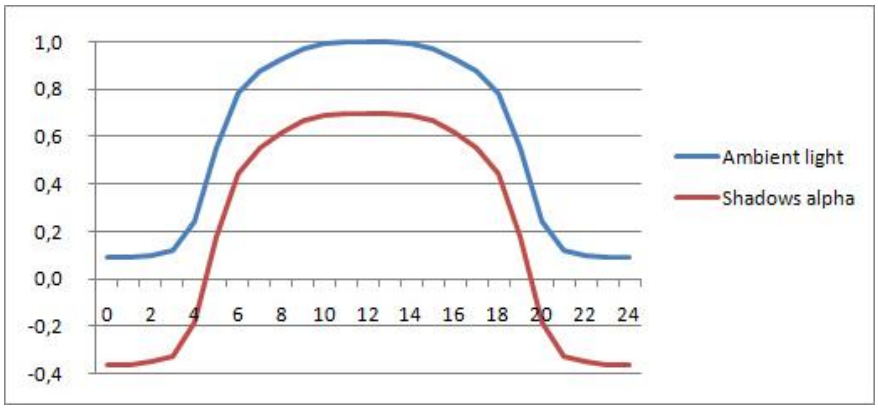
Exemple:

Voici la situation de départ, en considérant la variable de base *sl_sunshadows_alpha* = 0.7, associée à un *ToD* quelconque. L'axe des abscisses présente l'heure, celui des ordonnées les valeurs de lumière et d'alpha:



On constate que même pendant la nuit, l'alpha des ombres solaires est toujours nettement supérieur à 0, ce qui n'est évidemment pas cohérent avec un rendu réaliste de la lumière.

Voici maintenant les mêmes courbes, mais avec *sl_sunshadows_alphalimit* = 0.4 :



On remarque que l'alpha des ombres est inférieur à 0 dans tout l'intervalle où la luminosité ambiante est inférieure à 0.4, soit le moment où la valeur de *global.sl_ambient_light* est égale ou inférieure à celle de *sl_sunshadows_alphalimit*. Les ombres ne sont plus visibles pendant la fin de journée et la nuit: le rendu est alors bien plus crédible.

sl_sunshadows_margin

Détermine une marge autour de la view (en pixels) permettant aux objets situés hors la view d'être tout de même pris en compte dans le rendu de l'ombre. Cela peut être utile pour éviter un effet de clipping survenant avec des ombres longues.

La valeur doit être définie lors de l'initialisation de l'objet *engine*, et ne peut plus être modifiée par la suite.

sl_sunshadows_refresh

Détermine si les ombres sont rafraîchies ou non. Valeur booléenne (true / false).

sl_sunshadows_refreshrate

Fréquence de rafraîchissement des ombres, en nombre de steps. Indiquer 0 pour un rafraîchissement continu.

sl_sunshadows_layerscale[x]

Facteur appliqué à la variable *global.sl_sunshadows_length* pour le layer d'index ***x***.

En pratique, tous les objets d'un même layer projettent une ombre de la même longueur. Dans un souci d'esthétisme, cette variable permet de choisir une longueur différente pour chaque layer.

Exemple:

```
sl_sunshadows_layerscale[0]=0.4;
sl_sunshadows_layerscale[1]=1;
sl_sunshadows_layerscale[2]=1.6;
```

sl_ambientshadows_active

Détermine si l'affichage des ombres ambiantes est effectué ou non. Valeur booléenne (true / false).

sl_ambientshadows_alpha

Facteur de transparence des ombres ambiantes (0 invisible, 1 parfaitement visible).

sl_ambientshadows_mapscale

Facteur de taille des ambient maps générées à partir des sprites des objets avec la fonction *SL_sprite_set_ambient(sprite)*. Déclarez la variable avant d'exécuter la fonction sus-citée; une fois cela fait, il est préférable de ne plus modifier sa valeur.

sl_maxexposure

Variable relative à l'affichage du buffer de lumière. Elle détermine le facteur maximal d'exposition des lumières, afin de limiter leur saturation. Cela permet par exemple d'éviter que des lumières soient totalement blanches en plein jour, indépendamment de la puissance de celles-ci.

sl_tod_index

Index du segment courant du ToD. Elle correspond à la variable ***a*** de *sl_tod[a,b]*.

sl_ambient_color

Couleur de la lumière ambiante.

sl_layers_count

Nombre de layers composant le rendu des ombres globales.

sl_layers_surface[x]

id des surfaces des layers.

sl_global_list[x1,x2]

Liste contenant les id de l'ensemble des objets projetant une ombre globale (qu'elle soit solaire ou ambiante).

sl_sunshadows_list[x1,x2]

Liste contenant les id des objets projetant une ombre solaire dynamique.

sl_sunshadows_texlist[x1,x2]

Liste contenant les id des objets projetant une ombre solaire à partir d'une texture.

sl_sunshadows_refreshcounter

Compteur permettant la gestion de la fréquence de rafraîchissement.

sl_sunshadows_direction
sl_sunshadows_length
sl_sunshadows_light

Angle, longueur et luminosité des ombres solaires au step courant.

sl_sunshadows_surface1[x]
sl_sunshadows_surface2

id des surfaces utilisées dans le rendu des ombres.

sl_ambientshadows_lock

Variable relative à l'initialisation du système d'ombres ambiantes.

sl_ambientshadows_map[x]

Liste des ambient maps pour les sprites d'index ***x***.

sl_ambientshadows_list[x1,x2]

Liste contenant les id des objets projetant une ombre ambiante.

sl_ambientshadows_surface

id de la surface utilisée dans le rendu des ombres ambiantes.

sl_buffer_width
sl_buffer_height

Dimensions du buffer.

sl_buffer_surface1
sl_buffer_surface2

id de la surface constituant le buffer de lumière.

3.2 - Variables - Objet *light*

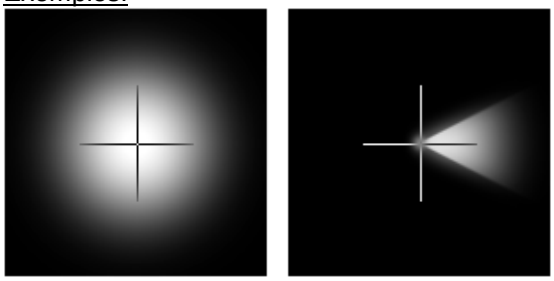
sl_light_active

Détermine si le rendu et l'affichage de la lumière sont effectués ou non. Valeur booléenne (true / false).

sl_light_texture

Index de la sprite constituant la texture de la lumière, dont l'origine doit être obligatoirement centrée. La texture doit être blanche sur fond noir: la couleur de la lumière étant déterminée lors du rendu par la variable *sl_light_color*.

Exemples:



Les dimensions des surfaces utilisées pour le rendu sont basées sur les dimensions de la sprite: ainsi, il n'est possible de modifier cette sprite en cours de jeu qu'à la condition que ses nouvelles dimensions soient inférieures ou égales aux dimensions de la sprite de départ. Egalement, et ceci est très important: en raison de la méthode utilisée pour calculer les ombres et afin de gagner en performances, toutes les textures de lumière doivent avoir les mêmes dimensions.

sl_light_x
sl_light_y

Coordonnées *x* (axe horizontal) et *y* (axe vertical) de la lumière dans la room. Elles déterminent le centre de la source lumineuse.

sl_light_xscale
sl_light_yscale

Facteurs d'étirements horizontal et vertical de la lumière, relatifs aux dimensions de la sprite déclarée dans *sl_light_texture*. Assurez-vous de ne pas déclarer une valeur de 0.

Par exemple: 1 taille normale, 2 taille double, 0.5 taille réduite de moitié, etc.

sl_light_angle

Angle de rotation de la texture de lumière.

sl_light_color

Couleur de la lumière. Elle peut être déclarée avec l'ensemble des syntaxes de couleur de *Game Maker*.

Exemples:

```
sl_light_color=c_yellow;
sl_light_color=make_color_rgb(255,255,0);
sl_light_color=$00FFFF;
```

sl_light_power

Facteur de puissance de la lumière, compris entre 0 (invisible) et 1 (maximum).

sl_light_ambientpower

Facteur de puissance de la luminosité résiduelle constituant les zones ombrées.

Par exemple:
Pour un facteur de 0, les zones d'ombre sont dépourvues de toute luminosité.
Pour un facteur de 0.5, les zones d'ombre sont moitié moins lumineuses que la lumière principale.

Plus simplement, cette variable détermine la 'transparence' des zones d'ombre.

sl_light_shadowlength

Longueur des ombres projetées par la lumière: plus la valeur est grande, plus l'ombre porte loin, et inversement. Elle détermine en fait le nombre d'itérations du rendu de l'ombre.

sl_light_shadowfactor

Facteur de grossissement appliqué successivement à chaque sample composant le rendu de l'ombre. En d'autres termes, elle détermine la qualité du rendu de l'ombre en agissant sur le nombre de samples la composant. Sa valeur doit être obligatoirement supérieure à 1.

Il n'est pas indispensable de pleinement comprendre son fonctionnement pour pouvoir l'utiliser: retenez simplement que plus la valeur est petite, plus l'effet sera esthétique, et plus la valeur est grande, plus le rendu sera rapide. (*Note:* la valeur par défaut est de 1.03).

sl_light_shadowsharpness

Facteur de dureté et de netteté des ombres. Il détermine la manière dont les objets ombrés sont éclairés par la source lumineuse.

Par exemple:
Pour un facteur de 1, les ombres sont nettes et les objets ombrés ne sont pas éclairés.
Pour un facteur de 0.3, les ombres sont adoucies et les objets ombrés sont partiellement éclairés.

sl_light_castshadow

Détermine si la lumière projette des ombres ou non. Valeur booléenne (true / false).

sl_light_refresh

Détermine si la lumière est rafraîchie ou non. Valeur booléenne (true / false).

Le rendu des ombres dynamiques est pris en compte, mais aussi celui de la texture de lumière elle-même: la variable *sl_light_angle* dépend par exemple du rafraîchissement.

sl_light_refreshrate

Fréquence de rafraîchissement de la lumière, en nombre de steps. Indiquer 0 pour un rafraîchissement continu.

sl_light_shadowlist[x]

Liste des id d'objets pour lesquels la lumière doit projeter une ombre. Son utilisation se fait de manière automatisée avec la fonction *SL_light_cast_obj(obj,mask)*.

sl_light_refreshcounter

Compteur permettant la gestion de la fréquence de rafraîchissement.

sl_light_visible

Détermine si la lumière est visible ou non (donc par extension s'il est nécessaire de la calculer ou non).

sl_light_cast

Détermine si l'id de l'objet light a été déclarée dans la liste *global.sl_lightlist[x]*.

sl_light_surface

id de la surface utilisée dans le rendu de la lumière.

3.3 - Variables - Globales

Le système de *SmartLight* comprend quelques variables globales (dont la grande majorité est faite de variables automatiques), permettant aux différents objets de communiquer entre eux indépendamment des noms leur étant attribués par l'utilisateur.

global.sl_viewid

Index de la view dans laquelle les effets de lumière doivent être affichés. Dans un souci d'ergonomie, il est préférable de la déclarer dans l'objet *engine* lors de l'initialisation de celui-ci.

Exemple:

```
global.sl_viewid=view_current;
```

global.sl_z[obj_id]

Variable z de l'objet d'id ***obj_id***, déterminant le facteur d'altitude par rapport au sol de l'ensemble des instances issues de cet objet. Elle agit sur le rendu des ombres solaires, en donnant l'impression que les objets sont élevés au dessus du sol. Elle est déclarée automatiquement avec une valeur par défaut de 0.

global.sl_time

Heure courante. Elle est supérieure ou égale à 0 et inférieure à 24. Sa vitesse de variation est déterminée par la variable *sl_timespeed* de l'objet *engine*.

Exemples:
Pour sl_time=0, il est minuit.
Pour sl_time=4, il est 4 heures du matin.
Pour sl_time=15, il est 15 heures (ou 3 heures de l'après-midi).
Pour sl_time=19.5, il est 19h 30 (ou 7h 30 de l'après-midi).

Le format décimal de la variable *global.sl_time* comprend donc les minutes, mais pas sous leur forme directe. Voici un petit code permettant d'obtenir l'heure et les minutes à partir de la variable *global.sl_time* brute:

```
heure=global.sl_time-frac(global.sl_time);
minutes=floor(60*frac(global.sl_time));
```

Cette variable est en lecture seule et ne peut être modifiée directement. Cependant, la fonction *SL_set_time(time)* permet de modifier l'heure à votre guise.

global.sl_ambient_light

Facteur courant de luminosité ambiante (puissance de la lumière du soleil).

global.sl_light_txsize

Dimension/taille unique des textures de lumière.

global.sl_light_gbuffer

id d’une surface servant de buffer global pour le rendu des lumières.

global.sl_texlist_light[x1,x2]

Liste des textures affichées dans le buffer de lumière à partir de la fonction *SL_draw_sprite_light(sprite,subimg,x,y,xscale,yscale,rot,color,alpha)*.

global.sl_texlist_light_index

Gestion de l'index d'enregistrement de la variable *global.sl_texlist_light[x1,x2]*.

global.sl_texlist_shad[x1,x2]

Liste des textures affichées dans le buffer d'ombre à partir de la fonction *SL_draw_sprite_shadow(layer,sprite,subimg,x,y,xscale,yscale,rot,alpha)*.

global.sl_texlist_shad_index

Gestion de l'index d'enregistrement de la variable *global.sl_texlist_shad[x1,x2]*.

global.sl_lightlist[x]

Liste des objets *light*.

4 - Fonctions

Certains scripts composant *SmartLight* comportent des arguments et sont conçus pour être utilisés comme des fonctions.

SL_light_cast_obj(obj_index,mask)

Fonction devant être appelée localement à partir d'un objet *light*, et permettant d'attribuer à ce dernier des objets à ombrer. En d'autres termes, elle permet de déterminer la liste des objets pour lesquels la lumière doit projeter une ombre.

Arguments:

obj_index : Index de l'objet devant occluder la lumière, formant ainsi une ombre.

mask : Index de la sprite devant être utilisée pour projeter l'ombre, à la place de la sprite de l'objet. Pour ne pas utiliser de mask et donc faire en sorte que la sprite projetée soit celle de l'objet elle-même, indiquez simplement -1.

Exemple:

```
SL_light_cast_obj(obj_character,-1);
```

SL_global_cast_obj(layer,obj_index,spr_index,sun,amb)

Fonction devant être appelée localement à partir de l'objet *engine*, et permettant de définir les objets devant bénéficier d'une ombre globale. Le terme "ombres globales" désigne les ombres solaires et les ombres ambiantes, soit des ombres affectant l'ensemble des objets à l'écran. Les arguments ***sun*** et ***amb*** permettent de sélectionner les types de rendus d'ombres globales à effectuer parmi les trois disponibles:

- Ombre solaire projetée (***sun*** = 1): l'objet projette une ombre provenant du soleil, calculée automatiquement à partir de sa sprite. Si vous désirez utiliser une autre sprite que celle de l'objet, vous pouvez définir un mask avec l'argument ***spr_index***. Cette option est préférable pour des objets tels que des murs, caisses, barils, véhicules etc.

- Ombre solaire projetée à partir d'une texture (***sun*** = 2): l'objet projette une ombre provenant du soleil, calculée à partir d'une texture prédéfinie et affichée à partir de l'origine de l'objet. La texture en question doit être définie avec l'argument ***spr_index***. Cette option est préférable pour des objets tels que des arbres ou des buissons.

- Ombre ambiante (***amb*** = 1): l'objet projette une ombre douce résultante de l'illumination globale.

Arguments:

layer : Index du layer dans lequel l'ombre de l'objet doit être affichée. Le système de layers permet de donner un effet de volume au rendu des ombres en faisant en sorte que les objets d'un layer donné (par exemple le layer 2) projettent leurs ombres sur les objets des layers inférieurs (dans notre exemple, les layers 1 et 0). Plus nombreux sont les layers, plus nombreuses sont les surfaces utilisées. Seules les ombres globales usent du système de layers (pas les lumières).

obj_index : Index de l'objet devant projeter une ombre globale.

spr_index : Cette variable joue un rôle différent selon le type de rendu d'ombre solaire utilisé. Pour ***sun*** = 1, elle désigne le mask à utiliser pour la projection dynamique de l'ombre (indiquez -1 pour ne pas utiliser de mask). Pour ***sun*** = 2, elle désigne l'index de la texture devant être utilisée pour la projection de l'ombre texturée.

sun : Détermine le type de rendu d'ombre solaire à effectuer. Valeurs: 1 ombre projetée, 2 ombre texturée, 0 désactivé.

amb : Active/désactive le rendu de l'ombre ambiante. Valeur booléenne (true / false). Assurez-vous d'avoir généré une ambient map pour la ou les sprite(s) de votre objet avec la fonction *SL_sprite_set_ambient(spr_index)*.

Exemples:

```
SL_global_cast_obj(2,obj_tree,spr_tree_shadow,2,1); // Désigne l'objet obj_tree comme devant projeter une ombre solaire texturée et une ombre ambiante dans le layer 2
```

```
SL_global_cast_obj(1,obj_character,-1,1,1); // Désigne l'objet obj_character comme devant projeter une ombre solaire et une ombre ambiante dans le layer 1
```

SL_cast_sprite(layer,castsun,castamb,castlights,sprite,subimg,x,y,z,xscale,yxscale,rot,alpha)

Affiche une ombre à partir d'une simple sprite, contrairement aux fonctions précédentes qui projettent des ombres à partir d'objets. Elle peut être appelée à partir de n'importe quel objet, et doit être exécutée à chaque step à la manière d'une fonction draw native; elle n'affiche cependant pas la sprite en elle-même, mais uniquement les ombres.

Le moment d'exécution de cette fonction doit être configuré avec précaution. Si son exécution se produit après celle du script *SL_engine_render* de l'objet *engine*, la sprite censée être affichée au step courant ne sera pas prise en compte dans le rendu, ce qui créera un effet de décalage. Assurez-vous donc que cette fonction soit exécutée avant que le rendu soit effectué (dans un événement ***Step*** ou ***Begin Step*** par exemple).

Arguments:

layer : Index du layer dans lequel les ombres doivent être affichées. Ne concerne que l'affichage des ombres solaires et ambiantes; vous pouvez y inscrire -1 si vous ne comptez afficher des ombres qu'avec les lumières dynamiques.

castsun : Détermine si une ombre solaire doit être projetée. Valeur booléenne (true / false).

castamb : Détermine si une ombre ambiante doit être affichée. Valeur booléenne (true / false).

castlights : Détermine si des ombres provenant des lumières dynamiques doivent être projetées. Chaque source lumineuse projettera alors une ombre pour cette sprite. Valeur booléenne (true / false).

z : altitude de la sprite par rapport au sol. Cette valeur n'affecte que le rendu des ombres solaires, et agit par multiplication de la valeur de longueur des ombres.

Les arguments restants sont identiques à ceux de la fonction native **draw_sprite_ext**.

Exemple:

```
SL_cast_sprite(0,1,1,1,spr_object,0,42,42,0,1,1,0,1);
```

SL_add_light(object)

Définit un objet comme étant une lumière. Elle peut être appelée à partir de n'importe quel objet et événement, mais seulement après que *SL_engine_ini_begin* ait été exécuté: il est donc généralement préférable de l'exécuter lors de l'initialisation de l'objet *engine*.

Argument:

object : Index de l'objet à définir comme étant une lumière.

Exemple:

```
SL_add_light(obj_light01); // L'objet obj_light01 peut maintenant être utilisé comme lumière dynamique
```

SL_draw_sprite_light(sprite,subimg,x,y,xscale,yscale,rot,color,alpha)

Affiche une sprite dans le buffer de lumière. A la différence d'une fonction draw classique, elle peut être appelée dans n'importe quel événement (***Step*** par exemple), et pas nécessairement ***Draw***. En pratique, cette fonction permet de créer des effets de lumière basiques ne nécessitant pas de rendu avancé.

Ses arguments sont identiques à ceux de la fonction native ***draw_sprite_ext***.

Exemple:

```
SL_draw_sprite_light(spr_light,0,300,20,1,1,0,c_orange,0.8);
```

SL_draw_sprite_shadow(layer,sprite,subimg,x,y,xscale,yscale,rot,alpha)

Affiche une sprite dans le buffer d'ombre (cette fonction est similaire à *SL_draw_sprite_light).* Elle s'utilise quasiment de la même manière que la fonction native ***draw_sprite_ext***, à quelques différences près: elle peut être appelée dans n'importe quel événement (***Step*** par exemple), et elle ne comporte pas d'argument ***color***. En pratique, cette fonction peut être utile pour ombrer certaines zones de votre map sans pour autant utiliser d'objets (dans le cas par exemple de l'intérieur d'un bâtiment).

Argument:

layer : Index du layer dans lequel la sprite doit être affichée.

Exemple:

```
SL_draw_sprite_shadow(1,spr_object,0,50,30,1,1,0,0.8);
```

SL_sprite_set_ambient(spr_index)

Génère une ambient map pour la sprite d'index ***spr_index***, indispensable au rendu des ombres ambiantes. Les sous-images (composant les sprites animées) sont automatiquement prises en compte. La fonction doit être appelée localement à partir de l'objet *engine* (vous pouvez user de l'instruction **with** si vous désirez l'appeler à partir d'un autre objet).

Argument:

spr_index : Index de la sprite pour laquelle une ambient map doit être générée.

Exemple:

```
SL_sprite_set_ambient(spr_character); // Génère une ambient map pour la sprite spr_character.
```

SL_set_time(time)

Initialise / modifie l'heure courante. La modification de l'heure ne peut se faire qu'avec cette fonction, la variable *global.sl_time* étant en lecture seule. La fonction doit être appelée localement à partir de l'objet *engine* (vous pouvez user de l'instruction **with** si vous désirez l'appeler à partir d'un autre objet).

Argument:

time : L'heure à laquelle vous désirez régler l'horloge. La valeur entrée doit être supérieure ou égale à 0 et inférieure à 24 (pour plus de précisions sur le fonctionnement de la variable d'heure *global.sl_time*, reportez-vous au paragraphe **3.3 - Variables - Globales**).

Exemples:

```
SL_set_time(8); // Règle l'horloge à 8 heures du matin
SL_set_time(15.7); // Règle l'horloge à 15 heures et 42 minutes
with obj_engine SL_set_time(15.7); // Règle l'horloge à partir d'un objet n'étant pas l'objet
engine
```

5 - Scripts

CBNA SmartLight GMS est composé d'un ensemble de 16 scripts GML, dont voici la liste:

SL_engine_ini_begin
SL_engine_ini_end
SL_engine_render
SL_engine_draw
SL_engine_free
SL_light_ini_begin
SL_light_ini_end
SL_light_free
SL_light_cast_obj - (fonction)
SL_global_cast_obj - (fonction)
SL_cast_sprite - (fonction)
SL_add_light - (fonction)
SL_draw_sprite_light - (fonction)
SL_draw_sprite_shadow - (fonction)
SL_sprite_set_ambient - (fonction)
SL_set_time - (fonction)

Les scripts surlignés en rouge ci-dessus comportent des arguments, et sont donc destinés à être utilisés comme des fonctions. Pour plus de précisions sur ces scripts, reportez-vous au paragraphe **4 - Fonctions**.

SL_engine_ini_begin

Commence l'initialisation du système. Doit être exécuté une seule fois au début de l'initialisation de l'objet *engine* (lors par exemple d'un événement ***Create***, ***Room Start*** ou ***Game Start***), et avant tous les autres scripts et variables.

SL_engine_ini_end

Termine l'initialisation du système. Doit être exécuté une seule fois à la fin de l'initialisation de l'objet *engine*, après et dans le même événement que *SL_engine_ini_begin*.

SL_engine_render

Effectue le rendu de la totalité des ombres et lumières. Doit être placé dans l'objet *engine* dans des événements de type ***Step*** (de préférence ***End Step***), et doit être exécuté avant le script *SL_engine_draw*.

SL_engine_draw

Affiche l'ensemble des ombres et lumières. Doit être placé dans l'événement ***Draw*** de l'objet *engine*, et exécuté après le script *SL_engine_render*.

SL_engine_free

Libère la mémoire occupée par la totalité des surfaces du système (y compris celles des objets *light*). Doit être exécuté lors de la destruction de l'objet *engine* (dans un événement ***Destroy***, par exemple).

SL_light_ini_begin

Commence l'initialisation d'un objet *light*. Doit être exécuté une seule fois au début de l'initialisation de cet objet *light* (lors par exemple d'un événement ***Create***, ***Room Start*** ou ***Game Start***).

SL_light_ini_end

Termine l'initialisation d'un objet *light*. Doit être exécuté une seule fois à la fin de l'initialisation de cet objet *light*, après et dans le même événement que *SL_light_ini_begin*.

SL_light_free

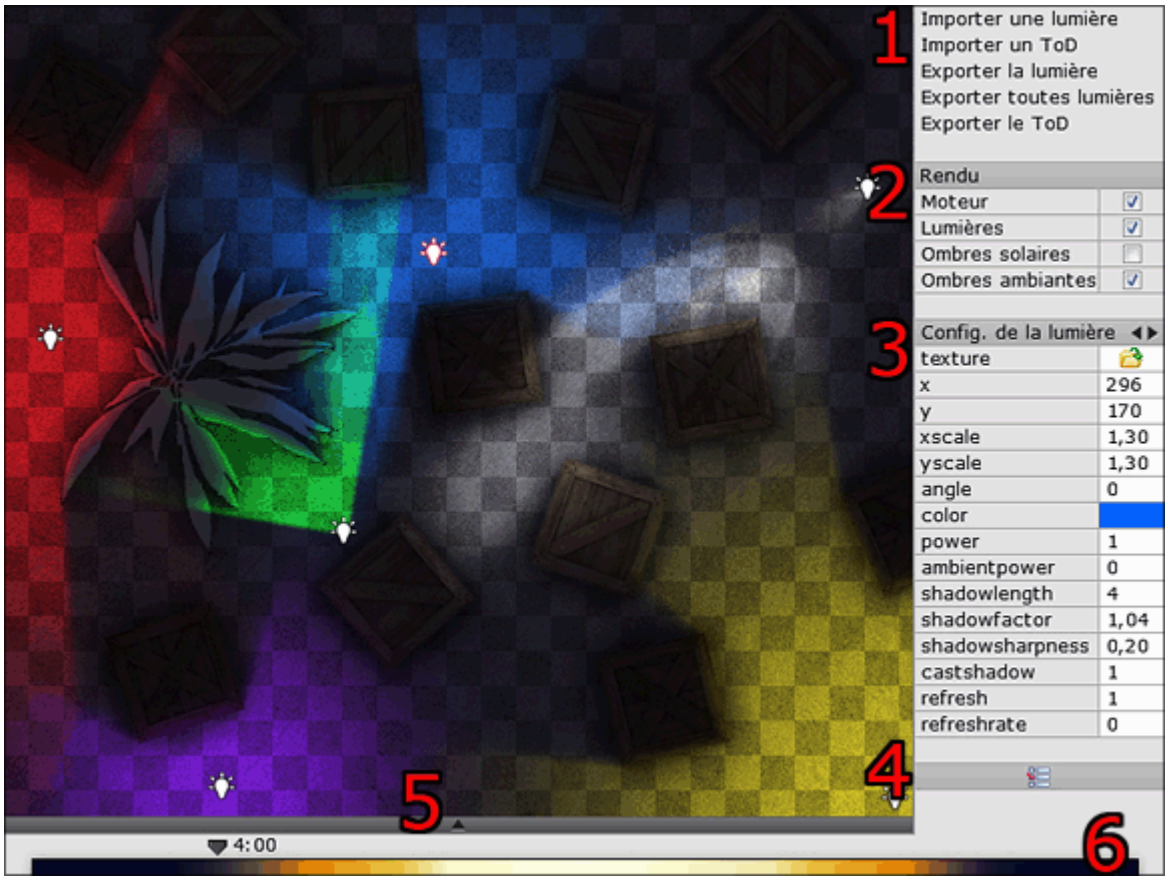
Libère la mémoire occupée par les surfaces d'un objet *light*. Doit être exécuté lors de la destruction de ce dernier (dans un événement ***Destroy***, par exemple).

6 - Éditeur

Voici un petit utilitaire permettant de créer et configurer des lumières et ToDs en temps réel, puis d'enregistrer leurs variables dans des fichiers texte (***.txt***) afin de pouvoir directement les importer dans GameMaker.

- Interface générale

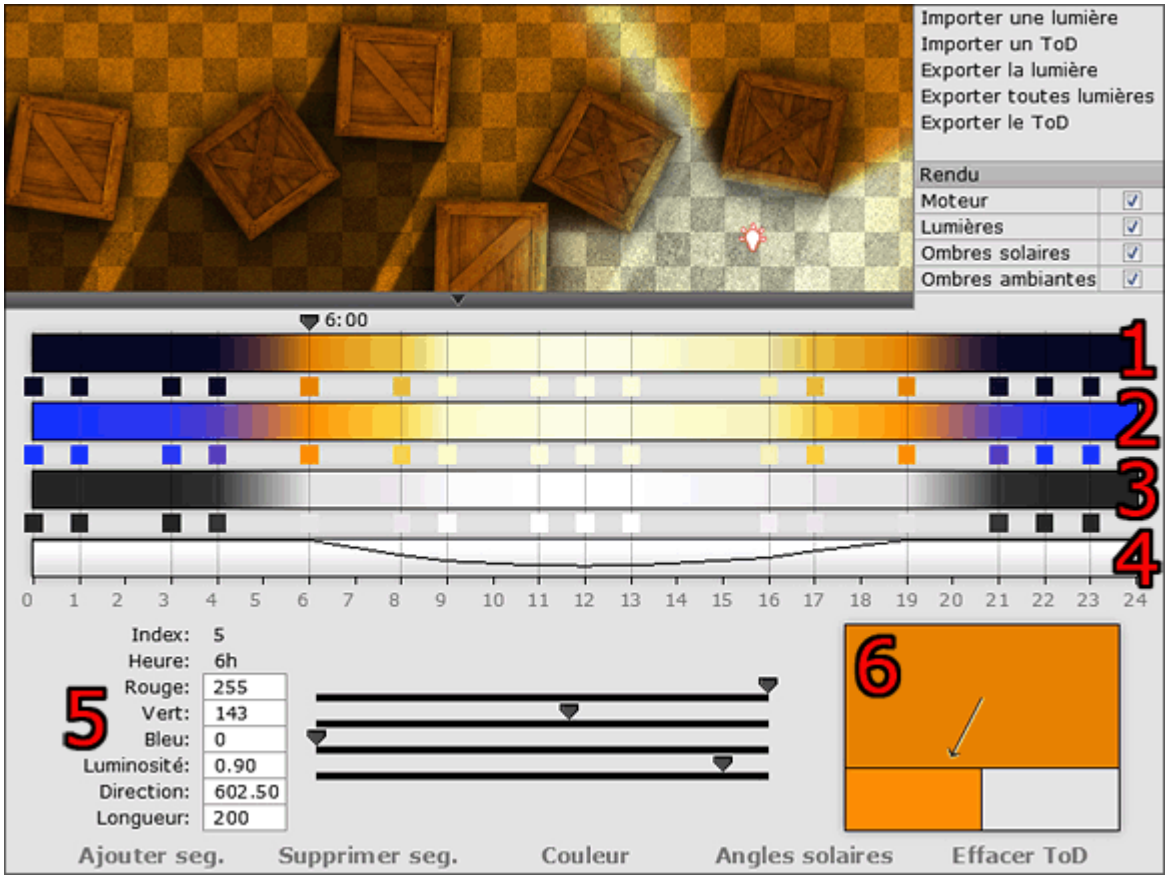
L'interface de base se compose d'une fenêtre d'aperçu des ombres et lumières au centre, d'une barre de configuration des lumières sur la droite, et de l'éditeur de ToD au bas de l'écran. Les objets peuvent être déplacés et pivotés par clic de la souris, et des options supplémentaires sont disponibles par clic droit sur un objet précis.



[1] - Options de chargement et sauvegarde des lumières et ToDs (en fichiers ***.txt***).
[2] - Liste des modules à calculer et afficher.
[3] - Valeurs de l'objet lumière sélectionné / Valeurs globales (cliquez sur les flèches pour passer d'un mode à l'autre)
[4] - Options / préférences.
[5] - Bouton permettant d'afficher ou masquer l'interface d'édition du ToD.
[6] - Aperçu global du ToD.

- Interface ToD

Le système du cycle jour/nuit de *SmartLight* tire ses valeurs d'un tableau de variables appelé *ToD* ("*Time of Day*"), usant de la variable *sl_tod[a,b]*. Ce tableau permet de définir la couleur de la lumière du soleil, la puissance de la lumière du soleil, ainsi que la direction et la longueur des ombres projetées par le soleil, pour chaque heure du jour et de la nuit. L'interface suivante permet de créer et d'éditer ces ToD. (pour plus de précisions sur la variable *sl_tod* en elle-même, reportez-vous au paragraphe **3.1 - Variables - Objet engine**)



[1] - Aperçu global des valeurs de couleur et luminosité.
[2] - Aperçu global des valeurs de couleur uniquement.
[3] - Aperçu global des valeurs de luminosité uniquement.
[4] - Aperçu global des valeurs de longueur des ombres solaires.
[5] - Valeurs du segment sélectionné.
[6] - Couleur, luminosité et direction des ombres du segment sélectionné.

Les commandes au bas de l'écran permettent respectivement d'ajouter un segment à l'heure précisée, de supprimer le segment sélectionné, d'éditer la couleur du segment sélectionné, de calculer automatiquement la rotation du soleil pour tous les segments, et d'effacer l'ensemble du ToD. Il vous est également possible de cliquer dans la zone d'aperçu (*1-4*) afin de pré-visualiser le rendu des lumières et ombres à une heure spécifique.

- Autres commandes clavier/souris

Afficher/masquer l'interface du ToD	**flèches haut / bas**
Parcourir les segments du ToD	**flèches gauche / droite**
Déplacer un objet	**clic gauche**
Pivoter un objet	**clic gauche + clic droit**
Options des objets/background	**clic droit**