Langage AL

AL est un nouveau langage (signifiant Algoid Language). Il a été inspiré par java, python, smalltalk et lua, mais il n'est pas ces langages.
L'intérêt d'Algoid Language est d'être simple, mais complet. Son but est d'aider à apprendre comment programmer et par la suite comment utiliser les différents paradigmes :
Impératif, procédural, fonctionnel et récursif, objet orienté prototype (héritage multiple) et programmation orientée aspect.
Il intègre des idiomes puissants comme le protocole meta-objet (inspiré de python) et cascade (inspiré de smalltalk).
Et parce qu'un jour vous allez, je l'espère, travailler avec un langage standard de l'industrie (très certainement proche du c et du java) AL a été conçu aussi proche de leurs syntaxes que possible.
Cette syntaxe avec ses accolades et ses symboles peut sembler très compliquée, mais on s'y fait vite et elle a l'avantage d'être concise.

En outre, les idiomes fondamentaux d'AL sont :

- Toutes les fonctions sont des expressions.
- Tous les objets sont des expressions.
- Toutes les expressions sont des objets.
- Donc toutes les fonctions sont des meta-fonctions et tous les objets sont des meta-objets.

Le reste est inspiré des différents langages énoncés plus haut.

Index

Language structure

instruction
block
expression
statement

Language primitives

while
do
loop
for
if
function
lambda
object
array

[dynamic] type

.getType
.is
.isNull
.ifNull
.ifNotBreak
.ifNullBreak
.equals
.toString
.add
.addAll
.onChanged

[dynamic] void

[dynamic] boolean

.not
.and
.or
.xor
.ifTrue
.ifFalse
.whileDo
.untilDo

[dynamic] number

.isInteger

.isReal
.isNan
.isInfinite
.toInteger
.minus
.increment
.decrement
.addition
.substract
.multiply
.divide
.modulo
.greaterThan
.smallerThan
.greaterOrEquals
.smallerOrEquals
.between
.decodePoint
.loopFor

[dynamic] string

.isEmpty
.length
.getChar
.contains
.concat
.indexOf
.count
.upper
.lower
.append
.appendSep

.subString
.subStringOf
.replace
.replaceAt
.remove
.split
.splitAt
.trim
.create
.encodePoint
.each

[dynamic] array

.isEmpty
.length
.getItem
.setItem
.getFirst
.getLast
.clone
.clear
.contains
.remove
.pop
.indexOf
.count
.swap
.decodePoint
.find
.create
.each

.eachOnRow
.eachOnCol
.eachItem
.filter
.sort
.min
.max
.join
.merge

[dynamic] function

.parameterExists
.getParametersNames
.setParameter
.setParameters
.removeParameter
.concat
.decorate

[dynamic] object

.clone
.attributeExists
.getAttributesNames
.isA
.setAttribute
object.toString
.merge
.removeAttribute

al

al.allObjects
al.allLocalObjects
al.clock

al.order

al.order.ascending
al.order.descending
al.order.random
al.order.reverse

al.combine

al.combine.sum
al.combine.product
al.combine.concat

al.types

al.types.VOID
al.types.BOOL
al.types.NUMBER
al.types.STRING
al.types.ARRAY
al.types.FUNCTION
al.types.OBJECT

math

math.E
math.PI
math.abs
math.acos
math.aim
math.asin
math.atan
math.ceil
math.cos
math.dbl
math.diagonal
math.exp
math.floor
math.log
math.max
math.min
math.pow
math.random
math.round
math.sin
math.sqrt
math.tan

Racine de l'API

eval
exists
include
load
print

game

game.createFont
game.createSound
game.createStage
game.getDebug
game.getRender
game.hasNext
game.hasPrevious
game.setDebug
game.setRender
game.goBack
game.goNext

game.color

game.color.BLACK
game.color.BLUE
game.color.CLEAR
game.color.CYAN
game.color.DARK_GRAY
game.color.GRAY
game.color.GREEN
game.color.LIGHT_GRAY
game.color.MAGENTA
game.color.MAROON
game.color.NAVY
game.color.OLIVE
game.color.ORANGE
game.color.PINK
game.color.RED
game.color.TEAL

game.color.WHITE
game.color.YELLOW

game.ease

game.ease.BACK_IN
game.ease.BACK_INOUT
game.ease.BACK_OUT
game.ease.BOUNCE_IN
game.ease.BOUNCE_INOUT
game.ease.BOUNCE_OUT
game.ease.CIRC_IN
game.ease.CIRC_INOUT
game.ease.CIRC_OUT
game.ease.CUBIC_IN
game.ease.CUBIC_INOUT
game.ease.CUBIC_OUT
game.ease.ELASTIC_IN
game.ease.ELASTIC_INOUT
game.ease.ELASTIC_OUT
game.ease.LINEAR
game.ease.QUAD_IN
game.ease.QUAD_INOUT
game.ease.QUAD_OUT
game.ease.SINE_IN
game.ease.SINE_INOUT
game.ease.SINE_OUT
game.ease.STRONG_IN
game.ease.STRONG_INOUT
game.ease.STRONG_OUT

game.key

game.key.ALT
game.key.ALTGR
game.key.CTRL
game.key.DEL
game.key.DOWN
game.key.END
game.key.ENTER
game.key.HOME
game.key.LEFT
game.key.PAGE_DOWN
game.key.PAGE_UP
game.key.RIGHT
game.key.SHIFT
game.key.SPACE
game.key.TAB
game.key.UP

actor

actor.applyForce
actor.applyForceX
actor.applyForceY
actor.bullet
actor.enablePhysic
actor.fixeRotation
actor.flipX
actor.flipY
actor.getAlpha
actor.getAngularVelocity
actor.getDensity
actor.getFriction
actor.getGravityScale

actor.getGroupId
actor.getHeight
actor.getMass
actor.getMaxVelocity
actor.getName
actor.getRotation
actor.getWidth
actor.getX
actor.getXVelocity
actor.getY
actor.getYVelocity
actor.hitCircle
actor.hitPolygone
actor.hitSquare
actor.impulse
actor.impulseX
actor.impulseY
actor.isActive
actor.isBullet
actor.isContacting
actor.isContactingBottom
actor.isContactingLeft
actor.isContactingRight
actor.isContactingTop
actor.isDisable
actor.isFixeRotation
actor.lowerLevel
actor.lowerToBack
actor.move
actor.moveBy
actor.outDisable
actor.outKeep
actor.outRemove

actor.raiseLevel
actor.raiseOnTop
actor.release
actor.releaseRotation
actor.rotate
actor.say
actor.scale
actor.scaleX
actor.scaleY
actor.setActive
actor.setAlpha
actor.setAngularVelocity
actor.setDensity
actor.setFriction
actor.setGravityScale
actor.setGroupId
actor.setHeight
actor.setMass
actor.setMaxVelocity
actor.setRestitution
actor.setSensor
actor.setSleepingAllowed
actor.setVelocity
actor.setWidth
actor.setX
actor.setXVelocity
actor.setY
actor.setYVelocity
actor.typeDynamic
actor.typeKinematic
actor.typeSensor
actor.typeStatic
actor.onContact

actor.onContactWith
actor.onEndContact
actor.onEndContactWith
actor.onPulse

actorAnim

actorAnim.getCurrentFrame
actorAnim.getFrameDuration
actorAnim.gotoFrame
actorAnim.play
actorAnim.repeat
actorAnim.reverse
actorAnim.sequence
actorAnim.setFrameDuration
actorAnim.stop
actorAnim.yoyo

button

button.getText
button.move
button.setText
button.onTap
button.onTouch
button.onTouchPulse
button.onUp

buttonImage

buttonImage.move

buttonImage.onTap
buttonImage.onTouch
buttonImage.onTouchPulse
buttonImage.onUp

checkBox

checkBox.getText
checkBox.move
checkBox.setText
checkBox.onTap
checkBox.onTouch
checkBox.onTouchPulse
checkBox.onUp

font

label

label.getText
label.move
label.setText
label.onTap
label.onTouch
label.onTouchPulse
label.onUp

layer

layer.createActor
layer.createActorAnim
layer.createButton
layer.createButtonImage
layer.createCheckBox
layer.createLabel
layer.createPad
layer.createParticle
layer.createRepeatedImage
layer.createVerticalLayout
layer.createWall
layer.follow
layer.getHeight
layer.getWidth
layer.remove
layer.scroll
layer.scrollX
layer.scrollY
layer.onScroll

particle

particle.move

repeatedImage

sequence

sequence.add
sequence.pause

sequence.repeat
sequence.repeatYoyo
sequence.resume
sequence.start
sequence.onDone

sound

sound.pause
sound.play
sound.playLoop
sound.resume
sound.stop

stage

stage.createLayer
stage.createSequence
stage.createTiledLayer
stage.createTween
stage.getHeight
stage.getWidth
stage.onAccelerometer
stage.onCompass
stage.onKey
stage.onKeyPulse
stage.onKeyUp
stage.onTap
stage.onTouch
stage.onTouchPulse
stage.onUp

touchPad

touchPad.getPrecentX
touchPad.getPrecentY
touchPad.move
touchPad.onTap
touchPad.onTouch
touchPad.onTouchPulse
touchPad.onUp

tween

tween.pause
tween.repeat
tween.repeatYoyo
tween.resume
tween.setEase
tween.start
tween.onDone

verticalLayout

verticalLayout.addActor
verticalLayout.getSpace
verticalLayout.move
verticalLayout.setSpace

instruction

L'élément unitaire du langage AL est l'instruction.
En AL, le point-virgule est totalement optionnel. Il existe parce qu'AL est inspiré du C et qu'en C il est obligatoire. Donc vous pouvez écrire votre programme avec ou sans. Préférez avec parce que la majorité des langages le demande. Ca devient une habitude.

```
SYNTAX : instruction [';']
```

Les instructions en AL se divisent en deux familles : les expressions (ou valeurs du programme) et les instructions de contrôle (issu de l'impératif).

block

Un bloc est une suite d'instructions encapsulées dans une portée locale.
La portée signifie que toute variable déclarée à l'intérieur n'est accessible que depuis cette portée et qu'elle sera détruite dès la sortie de celle-ci.

```
SYNTAX : '{' {[instruction]} '}'
```

expression

Toute valeur du langage s'appelle une expression. AL comprend plusieurs types d'expressions : VOID, BOOLEAN, NUMBER, STRING, FUNCTION, OBJECT, ARRAY.
AL est un langage dynamique, il ne nécessite pas de manipulation des types. Le trans-typage ce fait automatiquement, selon le besoin et les opérateurs utilisés.

```
SYNTAX :
nil
true | false
nan | infinity | number
'"' string '"'
array | function | object
ident '[' expr ']'
```

Les opérateurs binaires sont les suivant :

symbole	déscription
&&	(booléen) et : true (vrai) et true (vrai) donne true (vrai)
||	(booléen) ou : true (vrai) ou false (faux) donne true (vrai)
==	est égale
!=	est différent
<	est plus petit que
>	est plus grand que
<=	est plus petit ou égale à
>=	est plus grand ou égale à

+	(number) plus
-	(number) moins
*	(number) multiplie
/	(number) divise
%	(number) modulo
..	(string) concatène
->	(function) puis

Les opérateurs unaires sont les suivants :

symbole	description
!	(boolean) not (non) : !true (non vrai) donne false (faux)
-	(number) moins : -i est équivaut à 0 - i
++	incrementation : i++ équivaut à i = i + 1 ou i += 1
--	decrement : i-- équivaut à i = i - 1 ou i -= 1

exemple:

```
1
2
        ui.showLog();
```

```
3       set i = 100 / 2 + 4 * -2;
4       i++;
5       set b = true || false;
6       util.log("result : " .. i .. " and 1 or 0 : " .. b);
7
```

AL supporte également plusieurs valeurs constantes : true, false, nil, nan, infinity :
- true et false sont les deux valeurs booléenne
- nil représente la valeur null
- nan signifie "not a number" (résultat d'une opération non autorisée)
- infinity représente un nombre infinit (utiliser -infinity pour la valeur négative)

statement

Déclare une variable dans la portée courante.
Les variables déclarées peuvent être utilisées par la suite dans la même portée ou dans une portée imbriqué la la portée courante.

SYNTAX : 'set' ident [symbol expr] ';'

Les symboles sont les suivants :

symbole	déscription
=	égale i = n;
+=	plus égale, équivaut à i = i + n

-=	moins égale, équivaut à i = i - n
*=	multiplie égale, équivaut à i = i * n
/=	divise égale, équivaut à i = i / n
%=	modulo égale, équivaut à i = i % n
..=	concatène égale (pour les strings), équivaut à s = s .. t
->=	puis égale (pour les functions), équivaut à f = f -> g

La déclaration est elle même une expression.

exemple:

```
1
2       ui.showLog();
3       set i = 10;
4       i += 20;
5       util.log("result: " .. i);
6
```

while

While est la première instruction impérative. Elle boucle tant que sa condition renvoie true (vrai).

SYNTAX : 'while' '(' condition ')' block

exemple:

```
1
2       ui.showLog();
3       set i = 10;
4       while (i-- > 0) {
5         util.log("result: " .. i);
6       }
7
```

do

Boucle jusqu'à ce que la condition renvoie false (faux).

SYNTAX : 'do' block 'until' '(' condition ')'

exemple:

```
1
2       ui.showLog();
3       set i = 10;
4       do {
```

```
5           util.log("result: " .. i);
6         } until (i-- <= 0)
7
```

loop

Boucle le nombre de fois indiqué par le paramètre limite.

SYNTAX : 'loop' (limit) block

exemple:

```
1
2         ui.showLog();
3         set a = 0;
4         loop (10) {
5           util.log("loop: " .. a);
6           a++;
7         }
8
```

for

La boucle for initialise une variable et répète le bloc d'instruction tant que la progression de la variable respecte la condition.

SYNTAX : 'for' ([initialization] ';' [condition] ';' [progression]) block

exemple:

```
1
2       ui.showLog();
3       for (set i = 0; i < 10; i++) {
4         util.log("loop: " .. i);
5       }
6
```

if

IF exécute un bloc d'instruction si sa condition est true (vrai) sinon, c'est le bloc Else qui est exécuté. Elseif exécuté si la condition précédente n'est pas true (vrai) et si la sienne l'est.

SYNTAX : 'if' '(' condition ')' block {['elseif' '(' condition ')' block]} ['else' block]

exemple:

```
1
2       ui.showLog();
3       for (set i = 0; i<4; i++) {
4         if (i == 0) {
5           util.log("i is 0");
6         } elseif (i == 1) {
7           util.log("then 1");
8         } else {
9           util.log("then others");
10        }
11      }
12
```

En langage AL, IF est aussi un expression. Elle peut être utilisée pour décrire une valeur conditionnelle.

exemple:

```
1
2       ui.showLog();
3       set i = 0;
4       set s = if (i==0) "zero" elseif (i == 1) "one" else "other";
5       util.log ("if i=" .. i .. ", s=" .. s);
6
```

function

Définit un bloc d'instruction réutilisable dans une portée propre avec des paramètre en entrée et un paramètre de sortie.

La particularité d'AL, c'est que ses fonctions (ainsi que ses objets) sont considérés comme des valeurs. Ce sont des expressions et donc, peuvent se terminer par un point-virgule.

SYNTAX : 'function' ['(' [arg [{',' arg}]] ')'] '{' instruction '}'

exemple:

```
1
2
3
4
5
6
```

```
set f = function (x) {
  ui.message("x parameter is " .. x);
};
f (10);
```

lambda

Une expression lambda est une fonction avec une écriture simplifiée. Elle ne peut-être déclarée que comme paramètre d'une autre fonction. Elle a été créée pour simplifier l'écriture des foncteurs de la programmation fonctionnelle.

SYNTAX : ['(' [arg [{',' arg}]] ')'] '{' instruction '}'

exemple:

```
1
```

```
2       util.pulse({
3           algo.go(10);
4       }, 50);
5
```

object

Définit un ensemble de déclarations réutilisables dans leur portée propre.
Les déclarations peuvent être : des attributs (variables), des méthodes (fonctions) et des objets imbriqués.

SYNTAX : 'object' '(' {parent {[',' parent]}} ')' '{' declarations '}'

exemple:

```
1
2       ui.showLog ();
3       set o = object () {
4         set a = 0;
5         set b = "my b attribut";
6       };
7       util.log ("o.a = " .. o.a);
8       util.log ("o.b = " .. o.b);
9
```

Toute fonction déclarée dans un objet est appelé méthode. Les méthodes constituent les comportements de l'objet.
Une méthode désignée pour accéder à un attribut est appelé un accesseur.

Une méthode désignée pour modifier un attribut est appelé un mutateur.

exemple:

```
set o = object () {
  set a = 0;
  // the a setter
  set setA = function(a) {
    this.a = a;
  };
  // the a getter
  set getA = function() {
    return this.a;
  };
  // a method
  set doubleA = function() {
    this.a = this.a * this.a;
  };

};

o.setA(2);
ui.message("o.a parameter is " .. o.getA());
o.doubleA();
ui.message("and its double is " .. o.getA());

```

Les objets peuvent être dupliqués en conservants la même structure interne. Cela s'appel cloner un objet.
En AL les objets sont construits et peuvent être clonés. Il existe deux façons pour cloner; l'instruction new duplique l'objet et ses états actuels et la méthode clone.
Cette dernière permet d'attribuer de nouvelles valeurs aux états de l'objet en les lui passant en paramètre.

exemple:

```
1
2         ui.showLog();
3         set o = object () {
4           set a = 7;
5           set b = "my attribute b";
6           set toString = function () {
7             return "o {a=" .. a .. ", b=" .. b .. "};"
8           };
9         };
10        set p = o.clone (8, "another parameter");
11        util.log (o);
12        util.log (p);
13
```

This est une référence de l'objet à lui même.

exemple:

```
1
2           ui.showLog();
3         set o = object () {
4           set a;
5           set setA = function (a) {
6             this.a = a;
7           };
8           set getA = function () {
9             return this.a;
10          };
11        };
12        o.setA (7);
13        util.log (o.getA());
14
```

Supers[n] est un tableau qui permet d'accéder à la liste des super objets (héritage multiple).

exemple:

```
1
2       set q = object() {
3         set ret7 = function () {
4           return 7;
5         };
6       };
7
8       set p1 = object() {};
9
10      set p2 = object(q) {};
11
12      set o = object (p1, p2) {
13        set test = function () {
14          return this.supers[1].supers[0].ret7();
15        };
16      };
17
18      ui.message("o.test result is " .. o.test());
19
```

array

Définit un tableau d'éléments ou un dictionnaire (liste associative) indéxé par n'importe quelle expression.
Lors de la définition de tableaux imbriqués, le mot clé 'array' n'est obligatoire que pour le premier tableau (le root). Il devient optionnel ensuite.

```
SYNTAX :
item : [ expr ':' ] expr
array : [ 'array' ]1 '{' item [{ ',' item }] '}'
```

exemple:

```
1
2       ui.showLog();
3       set a = array {7, 8, 9, 10, 11};
4       for (set i=0; i < a.length(); i++) {
5         util.log("a[" .. i .. "] is " .. a[i]);
6       }
7       // or more elegent
8       a.each (function (item, index) {
9         util.log("a[" .. index .. "] is " .. item);
10      });
11
```

Les tableaux peuvent être utilisés comme des tableaux associatifs.
Ils gardent leur comportement de tableau, mais leurs valeurs sont référencées par une expression clé :

exemple:

```
1       ui.showLog();
        set a = array {"a" : 7, "b" : 8, "c" : 9, "d" : 10, "e" : 11};
2       a.each (function (item, index, key) {
          util.log("a[" .. key .. "] is " .. item);
3       });

4       util.log ("Find a[c] = " .. a["c"]);

5

6

7

8
```

9

Le framework Algoid

Un langage c'est une somme de structures de contrôle et de primitives qui permettent de manipuler les états.
Le framework est une librairie additionnelle (composé de fonctions et d'objets) qui met à disposition une somme d'outils.

Une particularité du langage AL est que chaque fois qu'une valeur est utilisée, AL met à disposition des méthodes pour agir sur elle.
Cet idiome a été inspiré de Smalltalk (cascade) et de Python (magic methods).

type

En AL, toutes les variables héritent de l'objet type. Lorsqu'une variable est déclarée, AL créer un objet approprié qui est aussi un objet type. Type à des propriétés et des méthodes qui sont donc communes à tous les types de variables.

property

`*.getType ()`

Renvoie le type AL de la donnée. Voir Types AL pour la référence complète.

`*.is (type)`

Vérifie que la donnée soit du type indiqué comme paramètre. Voir Types AL pour la référence complète.

`*.isNull ()`

Vérifie si le type de la variable est VOID (égale à nil).

`*.ifNull (value)`

Vérifie si le type de la variable est VOID (égale à nil). Si tel est le cas, la fonction retourne la valeur en paramètre.

`*.ifNotBreak (type)`

Vérifie que la donnée soit du type indiqué comme paramètre. Si tel n'est pas le cas, la fonction renvoie nil et termine la chaîne d'appel de fonctions. Voir Types AL pour la référence complète.

`*.ifNullBreak ()`

Vérifie si le type de la variable est VOID (égale à nil). Si tel est le cas, la fonction retourne la valeur en paramètre.

`*.equals (b)`

Vérifie si boolean est égale à b.

`*.toString ()`

Renvoie l'expression de la donné sous forme de texte. Dans le cas d'un type complexe (comme array, function ou object), renvoie l'ensemble des valeurs du type sous forme de texte.

method

```
*.add (item [, index])
```

Index est un paramètre optionnel.
Si index est absent, il crée un tableau avec la valeur en première position et y ajoute un élément à sa fin.
Sinon, il crée le tableau et ajoute l'élément à la position indiquée.

```
*.addAll (array [, index])
```

Index est un paramètre optionnel.
Si index est absent, il crée un tableau avec la valeur en première position et y ajoute tous les éléments à la fin.
Sinon, il crée le tableau et ajoute les éléments à la position indiquée.

event

```
*.onChanged ( function(value){} )
```

Un évènement qui se lève chaque fois que la valeur est modifiée dans le programme.

void

Quand une variable sans valeur est déclaré, AL créer un objet VOID. Cet objet à des propriétés et des méthodes. Voir l'objet Type pour les méthodes et les propriétés communes aux types.

boolean

Quand un booléen est déclaré, AL créer un objet. Cet objet à des propriétés et des méthodes. Voir l'objet Type pour les méthodes et les propriétés communes aux types.

property

`boolean.not ()`

Retourne la valeur inverse du booléen.

`boolean.and (b)`

Retourne le résultat de l'opération boolean && b.

`boolean.or (b)`

Retourne le résultat de l'opération boolean || b.

`boolean.xor (b)`

Retourne le résultat de l'opération (boolean || !b) && (!boolean || b). En d'autres termes, boolean différent de b

```
boolean.ifTrue (function () {})
```

Execute la fonction en paramètre is la valeur booléeene est vrai.

Equivalent fonctionel de l'instruction If.

```
boolean.ifFalse (function () {})
```

Execute la fonction en paramètre is la valeur booléeene est fausse.
Equivalent fonctionel de l'instruction Else.

```
boolean.whileDo (function () {})
```

Execute la fonction en paramètre tant que la valeur booléeene est vrai.
Equivalent fonctionel de l'instruction While.

```
boolean.untilDo (function () {})
```

Execute la fonction en paramètre tant que la valeur booléeene est vrai.
Equivalent fonctionel de l'instruction While.

number

Quand un nombre est déclaré, AL créer un objet. Cet objet à des propriétés et des méthodes. Voir l'objet Type pour les méthodes et les propriétés communes aux types.
Attention, les parenthèses doivent être employés avec ce type parce que le point est réservé aux décimales. ex : (7).isNumber();

```
number.isInteger ()
```

Vérifie si le type de la variable est de type NUMBER et entière.

`number.isReal ()`

Vérifie si le type de la variable est de type NUMBER et réel.

`number.isNan ()`

Vérifie si la valeur n'est pas un nombre, càd si le résultat de l'opération est invalide.

`number.isInfinite ()`

Vérifie si la valeur est infinie.

method

`number.toInteger ()`

Retourne la valeur entière du nombre.

`number.minus ()`

Retourne la valeur inverse du nombre, équivalant de -n.

`number.increment ([n])`

Retourne le nombre incrémenté, Équivalant de number + n. Si n n'est pas paramétré, il est égale à 1

`number.decrement ()`

Retourne le nombre décrémenté, Équivalant de number + n. Si n n'est pas paramétré, il est égale à 1

`number.addition (n)`

Equivaut à l'opération number + n.

```
number.substract (n)
```
Equivaut à l'opération number - n.

```
number.multiply (n)
```
Equivaut à l'opération number * n.

```
number.divide (n)
```
Equivaut à l'opération number / n.

```
number.modulo (n)
```
Equivaut à l'opération number % n.

```
number.greaterThan (n)
```
Equivaut à l'opération number > n.

```
number.smallerThan (n)
```
Equivaut à l'opération number < n.

```
number.greaterOrEquals (n)
```
Equivaut à l'opération number >= n.

```
number.smallerOrEquals (n)
```
Equivaut à l'opération number <= n.

```
number.between (min, max)
```
Retourne vrai si number est entre min et max. C'est l'Équivalant de number >= min && number <= max.

```
number.decodePoint ()
```

Renvoie le caractère correspondant au code codepoint.

```
number.loopFor (function (index) {} [, init, [, increment]])
```

Equivaut à l'instruction impérative for (set index = init; index < number; index += increment). Exécute la fonction depuis 0 (ou init, optionel) jusqu'au nombre en incrémentant de 1 (ou increment, optionel) chaque fois.

string

Quand une chaîne de caractère est utilisée, AL créer un objet STRING. Cet objet à des propriétés et des méthodes. Voir l'objet Type pour les méthodes et les propriétés communes aux types.

property

```
string.isEmpty ()
```

Vérifie si la chaîne est vide ("").

```
string.length ()
```

Retourne le nombre de caractères de la chaîne.

```
string.getChar (index)
```

Retourne le caractère à la position index dans la chaîne.

method

`string.contains (subString)`

Retourne vrai si la chaine contiens la sous-chaine.

`string.concat (string)`

Concatène la chaîne de caractères avec celle en paramètre.

`string.indexOf (subString, index)`

Retourne la position de la sous-chaîne contenue dans la chaîne.
Index est optionnel et indique à quelle position la recherche est commencé.

`string.count (subString)`

Retourne le nombre de sous chaîne trouvés dans la chaîne.

`string.upper ()`

Retourne l'équivalent en majuscule de la chaîne.

`string.lower ()`

Retourne l'équivalent en minuscule de la chaîne.

`string.append (substring [, index])`

Retourne le résultat de la concaténation entre la chaîne et la sous-chaîne.
Index est optionnel et spécifie la position où la sous-chaîne doit être insérée.
Équivalant de l'opérateur ".."

```
string.appendSep (substring , separator)
```

Retourne le résultat de la concaténation entre la chaîne et la sous-chaîne précédée d'un séparateur si elle n'était pas vide.

```
string.subString (begin [, end])
```

Renvoie la coupe de la chaîne en une sous-chaîne entre le début et la fin.
End, le paramètre de fin est optionnel. Il indique la fin de la sous-chaîne, s'il n'est pas indiqué, alors c'est la fin de la chaîne qui est prise.

```
string.subStringOf (begin [, end])
```

Renvoie la coupe de la chaîne en une sous-chaîne entre les sous-chaîne de début et de fin.
End est optionnel.

```
string.replace (from, to)
```

Renvoie la chaîne dont la sous-chaîne from a été remplacé par la sous-chaîne to.

```
string.replaceAt (subString, index)
```

Renvoie la chaîne dont la sous-chaîne a été remplacé à la position index.

```
string.remove (index [, length])
```

Renvoie la chaîne dont le caractère à la position à été remplacé.
Length est optionnel, if assigné il indique le nombre de caractère à supprimer.

```
string.split (separator)
```

Divise la chaîne de caractère en un tableau de chaînes selon le séparateur choisi.
Utiliser le séparateur "" pour convertir une chaîne en tableau de caractères.

```
string.splitAt (array)
```

Divise la chaîne de caractère en un tableau de chaînes selon les indexes spécifiés dans le tableau (array) choisi.

```
string.trim ()
```

Renvoie la chaîne dont les éspaces inutiles au début et à la fin de la chaîne ont été supprimés.

```
string.create (subString, count)
```

Créer une chaîne en dupliquant une sous-chaine de 0 à "count" fois.

```
string.encodePoint ()
```

Créer un tableau dont les valeurs sont les codespoints unicode de la chaîne.

functional method

```
string.each (function (char [, index]) {} [, step])
```

Itère sur tous les caractères de la chaîne.
La fonction exécutée doit définir un paramètre pour récupérer le caractère à chaque itération de la boucle.
Le second paramètre index est optionnel et fournis la position du caractère dans la chaîne.Step est optionnel, il représente le nombre de caractère à sauter avant la prochaine itération.

array

Quand un tableau est déclaré, AL créer un objet. Cet objet à des propriétés et des méthodes. Voir l'objet Type pour les méthodes et les propriétés communes aux types.

property

```
array.isEmpty ()
```

Vérifie si le tableau est vide ({}).

```
array.length ()
```

Retourne le nombre d'éléments du tableau.

```
array.getItem (identity)
```

Obtient l\élément du tableau à la position indiquée par identity.
Identity peu-t-être la position ou la clé de l'élément.

```
array.setItem (identity, item)
```

Définit la valeur du tableau à la position indiquée par identity.
Identity peu-t-être la position ou la clé de l'élément.

```
array.getFirst ()
```

Obtient le premier élément du tableau.

```
array.getLast ()
```

Obtient le dernier élément du tableau.

prototype

```
array.clone ()
```

Duplique le tableau vers un nouveau.
Chaque modification faite à un clone n'affecte pas l'objet original.

method

```
array.clear ()
```

Enlève tous les éléments du tableau.

```
array.contains (item)
```

Teste si l'élément est contenu par le tableau.

```
array.remove (identity)
```

Supprime un élément du tableau à la position de identity.
Identity peu-t-être la position ou la clé de l'élément.

```
array.pop ([identity])
```

Identity est un paramètre optionnel, il peu-t-être la position ou la clé de l'élément.
Si identity est absent, retourne le dernier élément du tableau et le supprime.
Si identity est spécifié, retourne l'élément à la position et le supprime du tableau.
Utiliser array.add() et array.pop() pour avoir un comportement de pile LiFo (Last in First out, ou en français DEPS Dernier entré Premier sortie). Utiliser array.add() et array.pop(0) pour avoir un comportement de file FiFo (First in First out, ou en français PEPS Premier entré Premier sortie)

```
array.indexOf (item)
```

Retourne la position de l'élément dans le tableau.

```
array.count (item)
```

Retourne le nombre d'élément d'une certaine valeur trouvée dans le tableau.

```
array.swap(identity1, identity2)
```

Intervertie les deux éléments aux positions identity1 et identity2.
Identity1 et 2 peuvent être les positions ou les clés des éléments.

```
array.decodePoint ()
```

Crée une chaîne de caractères à partir de sa représentation codepoint.

functional method

```
array.find (function (item [, index [, key]] [, pos]) {})
```

Trouve l'élément du tableau pour lequel la fonction retourne la valeur vraie.
La fonction exécutée doit définir un paramètre pour récupérer l'élément à chaque itération de la boucle.
Le second paramètre index est optionnel et fournis la position de l'élément dans le tableau.
Le troisième paramètre est également optionnel, il fournis la clé de l'élément.
Pos est un paramètre également optionnel de la méthode find. Elle indique à partir de quel indice la recherche doit commencer.

```
array.create (count, function ([index]) {})
```

Créer un tableau de n éléments. Chaque élément est calculé par la fonction passée en paramètre.

```
array.each (function (item [, index [, key]]) {})
```

Itère sur tous les éléments du tableau.
La fonction exécutée doit définir un paramètre pour récupérer l'élément à chaque itération de la boucle.

Le second paramètre index est optionnel et fournis la position de l'élément dans le tableau.
Le troisième paramètre est également optionnel, il fournis la clé de l'élément.

```
array.eachOnRow (row, function (item [, index [, key]]) {})
```

Dans un tableau à deux dimensions, itère sur tous les éléments de la ligne du tableau.
La fonction exécutée doit définir un paramètre pour récupérer l'élément à chaque itération de la boucle.
Le second paramètre index est optionnel et fournis la position de l'élément dans le tableau.
Le troisième paramètre est également optionnel, il fournis la clé de l'élément.

```
array.eachOnCol (col, function (item [, index [, key]]) {})
```

Dans un tableau à deux dimensions (une table), itère sur tous les éléments de la colonne du tableau.
La fonction exécutée doit définir un paramètre pour récupérer l'élément à chaque itération de la boucle.
Le second paramètre index est optionnel et fournis la position de l'élément dans le tableau.
Le troisième paramètre est également optionnel, il fournis la clé de l'élément.

```
array.eachItem (function (item [, index [, key]]) {})
```

Dans un tableau à plusieurs dimensions (un arbre), itère sur tous les éléments du tableau, si l'élément est un tableau, il va itérer sur les éléments du tableau récursivement.
La fonction exécutée doit définir un paramètre pour récupérer l'élément à chaque itération de la boucle.
Le second paramètre index est optionnel et fournis la position de l'élément dans le tableau.
Le troisième paramètre est également optionnel, il fournis la clé de l'élément.

```
array.filter (function (item [, index [, key]]) {})
```

Filtre les éléments du tableau sur la valeur de retour d'une fonction.
La fonction doit retourner un booléen.
Si la valeur du booléen est vrai, l'élément est conservé, sinon il est supprimé du tableau.

La fonction exécutée doit définir un paramètre pour récupérer l'élément à chaque itération de la boucle.
Le second paramètre index est optionnel et fournis la position de l'élément dans le tableau.
Le troisième paramètre est également optionnel, il fournis la clé de l'élément.

```
array.sort ([function (item1, item2){}])
```

Renvoie un tableau trié sur les éléments du tableau.
La fonction est optionnelle.

```
array.min ([function (item1, item2){}])
```

Retourne la valeur minimale contenue dans le tableau.
La fonction est optionnelle. Voir sort pour plus de détails;

```
array.max ([function (item1, item2){}])
```

Retourne la valeur maximale contenue dans le tableau.
La fonction est optionnelle. Voir sort pour plus de détails;

```
array.join (function (item1, item2 [, index [, key]]){})
```

Joint tous les éléments du tableau ensemble.
La fonction est nécessaire pour savoir comment joindre séquencielement les éléments.
C'est util pour concaténer un tableau en un string ou pour ajouter tous les nombre d'un tableau.

```
array.merge (array, function (item1, item2, index [, key]){})
```

Renvoie le résultat de la fusion de deux tableaux. Chaque élément du premier tableau est fusionné avec l'élément correspondant du second tableau.
La fonction détermine le comportement de la fusion en retournant la valeur souhaitée pour chaque élément.
Note: si le second tableau est trop court, les éléments seront répétés depuis le début jusqu\à la complétion.

function

Quand une fonction est déclarée, AL créer un objet. Cet objet à des propriétés et des méthodes. Voir l'objet Type pour les méthodes et les propriétés communes aux types.

property

```
function.parameterExists (name)
```

Vérifie que le paramètre existe dans la définition de la fonction.

```
function.getParametersNames ()
```

Retourne un tableau contenant tous les noms des paramètres définit par la fonction.

```
function.setParameter (name, value))
```

Assigne la valeur au paramètre de la fonction dont le nom correspond à name. Le paramètre est créé s'il n'existe pas. Utilisé pour assigner les paramètres avant l'appelle de la fonction (util lors du passage d'une fonction à une autre sans copie des paramètres).

```
function.setParameters ({ values })
```

Assigne les valeurs aux paramètres de la fonction. Attention, les valeurs des paramètres doivent être regroupés dans un tableau (entre {}). Utilisé pour assigner tous les paramètres avant l'appelle de la fonction (util lors du passage d'une fonction à une autre sans copie des paramètres).

method

```
function.removeParameter (name)
```

Supprime dynamiquement un paramètre de la définition de la fonction.

functional method

```
function.concat (function)
```

Retourne une fonction résultat de la concaténation des deux fonctions. Les paramètres sont fusionnés et les traitements s'exécutent en suivant. Equivaut à l'opérateur ->

```
function.decorate (function)
```

Renvoie une nouvelle fonction résultat de la décoration de la fonction par une fonction décoratrice. L'éxécution de la fonction décorée est encapsulé dans la fonction décoratrice. Nécessaire au paradigme Aspect.

object

Quand un objet est déclaré, AL créer un meta-objet. Cet objet à des propriétés et des méthodes. Voir l'objet Type pour les méthodes et les propriétés communes aux types.

prototype

```
object.clone ()
```

Duplique l'objet vers un nouveau.
Chaque modification faite à un clone n'affecte pas l'objet original.
C'est l'équivalent de l'opérateur new.

property

```
object.attributeExists (name)
```

Vérifie que l'attribut existe dans la définition de l'objet.

```
object.getAttributesNames ()
```

Retourne un tableau contenant tous les noms des attributs définit par l'objet.

```
object.isA ()
```

Vérifie que l'objet soit une instance ou un sous objet de celui en paramètre.

```
object.setAttribute (name, value)
```

Assigne ou ajoute (si absent) dynamiquement un attribut à la définition de l'objet. L'attribut peut-être un attribut, une méthode ou un objet imbriqué.

method

```
object.toString = function () {}
```

Remplace le text par défaut de l'object.

```
object.merge (object)
```

Renvoie un objet résultat de la fusion de deux objets. Les définitions sont fusionnées ensemble par

addition des attributs et des méthodes.

```
object.removeAttribute (name)
```

Supprime dynamiquement un attribut de la définition de l'objet. L'attribut peut-être un attribut, une méthode ou un objet imbriqué.

al

property

```
al.allObjects
```

Retourne tous les objets et les méthodes qui peuvent être utilisés dans l'API Algoid.

```
al.allLocalObjects
```

Retourne tous les objets et les méthodes qui peuvent être utilisés dans la portée courante.

```
al.clock
```

Retourne le temps en seconde depuis lequel Algoid a été lancé.

al.order

method

```
al.order.ascending (item1, item2)
```
Trie l'ordre du tableau de façon ascendante.

```
al.order.descending (item1, item2)
```
Trie l'ordre du tableau de façon ascendante.

```
al.order.random (item1, item2)
```
Inverse l'ordre du tableau.

```
al.order.reverse (item1, item2)
```
Inverse l'ordre du tableau.

al.combine

```
al.combine.sum (item1, item2)
```
Combine les éléments d'un tableau ensemble par sommage.

```
al.combine.product (item1, item2)
```
Combine les éléments d'un tableau ensemble par produit.

```
al.combine.concat (item1, item2, index, key, separator)
```
Combine les éléments d'un tableau ensemble par concaténation.

al.types

property

`al.types.VOID`

Le type AL VOID. Représente une valeur nulle.

`al.types.BOOL`

Le type AL BOOL. Représente une valeur booléenne (true, false).

`al.types.NUMBER`

Le type AL NUMBER. Représente une valeur numérique (1, 2, 3.5, 7 ....).

`al.types.STRING`

Le type AL STRING. Représente une chaîne de caractère ("Hi, I am algoid !").

`al.types.ARRAY`

Le type AL ARRAY. Représente un tableau de données ({true, 5, "Hi"}).

`al.types.FUNCTION`

Le type AL FUNCTION. Représente une fonction.

`al.types.OBJECT`

Le type AL OBJECT. Représente un objet.

math

constant

`math.E`

La constante mathématique e.

`math.PI`

La constante mathématique PI.

method

`math.abs (number)`

Retourne la valeur absolue (positive) du nombre.

`math.acos (factor)`

Calcule l'arc cosinus d'un angle en degré.

```
math.aim (number, number)
```

Calcule l'angle pour suivre une cible depuis les coordonnées (0, 0) du plan.

```
math.asin (factor)
```

Calcule l'arc sinun d'un angle en degré.

```
math.atan (factor)
```

Calcule l'arc tangente d'un angle en degré.

```
math.ceil (number)
```

Renvoie la plus petite valeur entière directement suppérieur au nombre : 10.9 = 11

```
math.cos (angle)
```

Calcule le cosinus d'un angle en degré.

```
math.dbl (number)
```

Calcule le nombre à la puissance 2.

```
math.diagonal (number, number)
```

Calcule la longueur de la diagonale (hypothénuse) d'un rectangle.

```
math.exp (number)
```

Calcule la valeur exponentielle du nombre.

```
math.floor (number)
```

Renvoie la partie entière du nombre : 10.9 = 10

```
math.log (number[, base])
```

Renvoie le logarithme naturel du nombre. Is la base est assignée, renvoie le logarithme du nombre de la base donnée.

```
math.max (number, number)
```

Retourne le plus grand des deux nombres.

```
math.min (number, number)
```

Retourne le plus petit des deux nombres.

```
math.pow (number, power)
```

Calcule le nombre à la puissance n.

```
math.random (factor)
```

Génère un nombre aléatoire entre 0 et factor à l\exception de celui-ci.

```
math.round (number)
```

Arrondi le nombre réel en un nombre naturel : 10.9 = 11

```
math.sin (angle)
```

Calcule le sinun d'un angle en degré.

```
math.sqrt (number)
```

Calcule la racine carrée du nombre.

`math.tan (angle)`

Calcule la tangente d'un angle en degré.

API root

Quelques fonctions utiles.

`eval(script)`

Exécute le script AL passé en paramètre.

`exists(variable)`

Retourne vrai si la variable a été déclarée et qu'elle est accessible.

`include(path)`

Inclue (lit une seule fois) un script AL or ALG dans la portée courante.

`load(path)`

Charge un script AL or ALG dans la portée courante.

`print(text)`

Ecrit le text pris en paramètre, dans la sortie standard.

game

Le point d'entrée d'un jeu.

factory

```
game.createFont(path, size, color)
```

Crée une nouvelle police de caractère.Crée une nouvelle police de caractère. Voir l'objet font pour une référence complète.Crée une nouvelle police de caractère.Crée une nouvelle police de caractère. Voir l'objet font pour une référence complète. Voir l'objet game.color pour une référence complète.

```
game.createSound(path)
```

Crée un nouveau son.Crée un nouveau son. Voir l'objet sound pour une référence complète.

```
game.createStage([gravityX[, gravityY]])
```

Crée un nouveau stage dans la pile avec une gravité (physique).Crée un nouveau stage dans la pile avec une gravité (physique). Voir l'objet stage pour une référence complète.

property

```
game.getDebug()
```

Renvoie si le mode de rendu debug est activé.

`game.getRender()`

Renvoie si le rendu est activé.

`game.hasNext()`

Retourne si le stage courant est suivi d'un autre stage dans la pile (en vue de créer un transition).

`game.hasPrevious()`

Retourne si le stage courant est précédé d'un autre stage dans la pile (en vue de créer un transition).

`game.setDebug()`

Active ou non le mode de rendu debug

`game.setRender()`

Active ou non le rendu

method

`game.goBack()`

Va au stage précédent et lance une animation de transition.

`game.goNext()`

Va au stage suivant et lance une animation de transition.

game.color

Un ensemble de constantes de couleurs. Les couleurs personnalisées peuvent être écrit sous la forme d'une chaîne de caractères au format "aarrggvv", aa pour les deux digit hexadécimaux de la composante alpha, rr, gg et bb respectivement, les composantes rouge (red), verte (green) et bleue (blue).

enum

`game.color.BLACK`

La constante de couleur BLACK (000000ff).

`game.color.BLUE`

La constante de couleur BLUE (0000ffff).

`game.color.CLEAR`

La constante de couleur CLEAR (00000000).

`game.color.CYAN`

La constante de couleur CYAN (00ffffff).

`game.color.DARK_GRAY`

La constante de couleur DARK_GRAY (3f3f3fff).

`game.color.GRAY`

La constante de couleur GRAY (7f7f7fff).

`game.color.GREEN`

La constante de couleur GREEN (00ff00ff).

`game.color.LIGHT_GRAY`

La constante de couleur LIGHT_GRAY (bfbfbfff).

`game.color.MAGENTA`

La constante de couleur MAGENTA (ff00ffff).

`game.color.MAROON`

La constante de couleur MAROON (7f0000ff).

`game.color.NAVY`

La constante de couleur NAVY (00007fff).

`game.color.OLIVE`

La constante de couleur OLIVE (7f7f00ff).

`game.color.ORANGE`

La constante de couleur ORANGE (ffc600ff).

`game.color.PINK`

La constante de couleur PINK (ffadadff).

`game.color.RED`

La constante de couleur RED (ff0000ff).

`game.color.TEAL`

La constante de couleur TEAL (007f7fff).

`game.color.WHITE`

La constante de couleur WHITE (ffffffff).

`game.color.YELLOW`

La constante de couleur YELLOW (ffff00ff).

game.ease

Un ensemble de constantes de formule d'animation.

`game.ease.BACK_IN`

La formule d'animation BACK_IN.

`game.ease.BACK_INOUT`

La formule d'animation BACK_INOUT.

`game.ease.BACK_OUT`

La formule d'animation BACK_OUT.

`game.ease.BOUNCE_IN`

La formule d'animation BOUNCE_IN.

`game.ease.BOUNCE_INOUT`

La formule d'animation BOUNCE_INOUT.

`game.ease.BOUNCE_OUT`

La formule d'animation BOUNCE_OUT.

`game.ease.CIRC_IN`

La formule d'animation CIRC_IN.

`game.ease.CIRC_INOUT`

La formule d'animation CIRC_INOUT.

`game.ease.CIRC_OUT`

La formule d'animation CIRC_OUT.

`game.ease.CUBIC_IN`

La formule d'animation CUBIC_IN.

`game.ease.CUBIC_INOUT`

La formule d'animation CUBIC_INOUT.

`game.ease.CUBIC_OUT`

La formule d'animation CUBIC_OUT.

`game.ease.ELASTIC_IN`

La formule d'animation ELASTIC_IN.

`game.ease.ELASTIC_INOUT`

La formule d'animation ELASTIC_INOUT.

`game.ease.ELASTIC_OUT`

La formule d'animation ELASTIC_OUT.

`game.ease.LINEAR`

La formule d'animation LINEAR.

`game.ease.QUAD_IN`

La formule d'animation QUAD_IN.

`game.ease.QUAD_INOUT`

La formule d'animation QUAD_INOUT.

`game.ease.QUAD_OUT`

La formule d'animation QUAD_OUT.

`game.ease.SINE_IN`

La formule d'animation SINE_IN.

`game.ease.SINE_INOUT`

La formule d'animation SINE_INOUT.

`game.ease.SINE_OUT`

La formule d'animation SINE_OUT.

`game.ease.STRONG_IN`

La formule d'animation STRONG_IN.

`game.ease.STRONG_INOUT`

La formule d'animation STRONG_INOUT.

`game.ease.STRONG_OUT`

La formule d'animation STRONG_OUT.

game.key

Un ensemble de constantes de touche clavier.

`game.key.ALT`

La constante de touche ALT.

`game.key.ALTGR`

La constante de touche ALTGR.

`game.key.CTRL`

La constante de touche CTRL.

`game.key.DEL`
La constante de touche DEL.

`game.key.DOWN`
La constante de touche DOWN.

`game.key.END`
La constante de touche END.

`game.key.ENTER`
La constante de touche ENTER.

`game.key.HOME`
La constante de touche HOME.

`game.key.LEFT`
La constante de touche LEFT.

`game.key.PAGE_DOWN`
La constante de touche PAGE_DOWN.

`game.key.PAGE_UP`
La constante de touche PAGE_UP.

`game.key.RIGHT`

La constante de touche RIGHT.

`game.key.SHIFT`

La constante de touche SHIFT.

`game.key.SPACE`

La constante de touche SPACE.

`game.key.TAB`

La constante de touche TAB.

`game.key.UP`

La constante de touche UP.

actor

Actor est l'objet principal d'ALGEA. Il est composé par une image et peut être bougé ou animé selon les objet tweens ou la physique.

L'objet actor doit être créé par la méthode de fabrication : layer.createActor(). Voir : layer.createActor() pour une référence complète.

exemple:

```
1   // initialize stage and layer
2   set stage = game.createStage(0, -10);
3   set layer = stage.createLayer();
4
5   // create ground
6   set x = 0;
7   loop(200) {
8     set ground = layer.createActor("PlanetCute/Grass Block");
9     ground.move(x, 50);
10    ground.typeStatic().hitSquare(75);
11    x = x + ground.getWidth();
12  }
13
14  // create hero
15  set hero = layer.createActor("PlanetCute/beetleship");
16  hero.move(500, 500);
17  hero.typeDynamic().hitCircle(15);
```

method

`actor.applyForce(forceX, forceY)`

Applique une force linéaire sur l'acteur (utiliser setMaxVelocity pour limité la vélocité).

`actor.applyForceX(forceX)`

Applique une force linéaire sur l'acteur sur l'axe X (utiliser setMaxVelocity pour limité la vélocité).

`actor.applyForceY(forceY)`

Applique une force linéaire sur l'acteur sur l'axe Y (utiliser setMaxVelocity pour limité la vélocité).

`actor.bullet(value)`

Fixe le fait que l'acteur ne puisse pas recouvrir les autres durant la simulation.

`actor.enablePhysic()`

Rends actif la simulation pour l'acteur.

`actor.fixeRotation()`

Rends inactive la rotation de l'acteur.

`actor.flipX()`

Applique l'effet miroir de l'acteur sur l'axe X.

`actor.flipY()`

Applique l'effet miroir de l'acteur sur l'axe Y.

`actor.getAlpha()`

Retourne la transparence de l'acteur (0 est transparent, 0.5 est translucide et 1 est opaque).

`actor.getAngularVelocity()`

Retourne la velocité angulaire (rotation à vitesse constante).

`actor.getDensity()`

Retourne la densité de l'acteur.

`actor.getFriction()`

Retourne le facteur de frottement de l'acteur.

`actor.getGravityScale()`

Retourne le facteur de gravité de l'acteur.

`actor.getGroupId()`

Retourne l'identifiant de groupe (util pour tester les collisions).

`actor.getHeight()`

Retourne la hauteur de l'acteur.

`actor.getMass()`

Retourne la masse de l'acteur en kilogrammes.

`actor.getMaxVelocity()`

Retourne la vélocité maximum à laquelle l'acteur peut bouger (util pour les forces).

`actor.getName()`

Retourne le nom de l'acteur (le nom de son image).

`actor.getRotation()`

Retourne la rotation de l'acteur (en degrés).

`actor.getWidth()`

Retourne la largueur de l'acteur.

`actor.getX()`

Retourne la position de l'acteur sur l'axe x.

```
actor.getXVelocity()
```

Retourne la velocité (déplacement à vitesse constante) de l'acteur sur l'axe x.

```
actor.getY()
```

Retourne la position de l'acteur sur l'axe y.

```
actor.getYVelocity()
```

Retourne la velocité (déplacement à vitesse constante) de l'acteur sur l'axe y.

```
actor.hitCircle(radiusMargin)
```

Créer une fixture circulaire. La marge détermine la réduction de la taille à appliquer pour couvrir l'image.

```
actor.hitPolygone(points)
```

Créer une fixture polygonale. Les coordonnées du polygone sont représentées par un tableau de paires d'x, y.

```
actor.hitSquare([topMargin[, bottomMargin[, leftMargin[, rightMargin]]]])
```

Créer une fixture rectangulaire. La marge détermine la réduction de la taille à appliquer pour couvrir l'image.

```
actor.impulse(x, y)
```

Applique une impulsion linéaire à l'acteur.

```
actor.impulseX(x)
```

Applique une impulsion linéaire à l'acteur sur l'axe X.

`actor.impulseY(y)`

Applique une impulsion linéaire à l'acteur sur l'axe Y.

`actor.isActive()`

Retourne si la simulation et la détection de colisions ont étés désactivés pour l'acteur.

`actor.isBullet()`

Retourne le fait que l'acteur puisse recouvrir ou pas les autres durant la simulation.

`actor.isContacting()`

Retourne vrai si l'acteur est en collision avec quelque chose.

`actor.isContactingBottom()`

Retourne vrai si l'acteur est en collision avec quelque chose par le bas.

`actor.isContactingLeft()`

Retourne vrai si l'acteur est en collision avec quelque chose par la gauche.

`actor.isContactingRight()`

Retourne vrai si l'acteur est en collision avec quelque chose par la droite.

`actor.isContactingTop()`

Retourne vrai si l'acteur est en collision avec quelque chose par le haut.

`actor.isDisable()`

Rends inactif la simulation pour l'acteur.

```
actor.isFixeRotation()
```
Retourne si la capacité de rotation a été désactivés pour l'acteur.

```
actor.lowerLevel()
```
Place l'acteur au à un niveau en dessous.

```
actor.lowerToBack()
```
Place l'acteur tout en bas (au dessous de tous les autres).

```
actor.move(x, y)
```
Positionne l'acteur sur l'écran.
L'acteur est positionné par son centre.
La position (0, 0) est le coin inférieur gauche de l'écran.

```
actor.moveBy(x, y)
```
Positionne l'acteur sur l'écran de façon relative à sa position précédente.

```
actor.outDisable()
```
Conserve l'acteur s'il dépace de l'écran.

```
actor.outKeep()
```
Conserve l'acteur s'il dépace de l'écran.

```
actor.outRemove()
```
Conserve l'acteur s'il dépace de l'écran.

```
actor.raiseLevel()
```

Place l'acteur au à un niveau au dessus.

```
actor.raiseOnTop()
```

Place l'acteur au top (au dessus de tous les autres).

```
actor.release()
```

Rend la main au simulateur pour calculer la position.

```
actor.releaseRotation()
```

Rends active la rotation de l'acteur.

```
actor.rotate(degree)
```

Fix l'angle de l'acteur (en degrés).

```
actor.say(font, text)
```

Crée une bulle de dialogue au dessus de l'acteur et y écrit le text.

```
actor.scale(factor)
```

Augmente ou réduit la taille de l'acteur (inférieur à 1 pour réduire et supérieur pour agrandir).

```
actor.scaleX(factor)
```

Augmente ou réduit la largeur de l'acteur (inférieur à 1 pour réduire et supérieur pour agrandir).

```
actor.scaleY(factor)
```

Augmente ou réduit la heuteur de l'acteur (inférieur à 1 pour réduire et supérieur pour agrandir).

```
actor.setActive(value)
```

Rends inactif la simulation et la détection de colisions pour l'acteur.

`actor.setAlpha()`

Modifie la transparence de l'acteur (0 est transparent, 0.5 est translucide et 1 est opaque).

`actor.setAngularVelocity(angle)`

Change la velocité angulaire (rotation à vitesse constante).

`actor.setDensity(factor)`

Applique un facteur de frottement à l'acteur.

`actor.setFriction(factor)`

Applique un facteur de frottement à l'acteur.

`actor.setGravityScale(factor)`

Applique un facteur de frottement à l'acteur.

`actor.setGroupId(id)`

Assigne l'identifiant de groupe (util pour tester les collisions).

`actor.setHeight(value)`

Spécifie la hauteur de l'acteur.

`actor.setMass(mass)`

Applique la masse de l'acteur en kilogramme.

```
actor.setMaxVelocity(velocity)
```
Fixe la vélocité maximum à laquelle l'acteur peut bouger (util pour les forces).

```
actor.setRestitution(value)
```
Applique un facteur de restitution à l'acteur.

```
actor.setSensor(value)
```
Rends inactif la simulation pour l'acteur Mais garde son type d'interaction physique.

```
actor.setSleepingAllowed(value)
```
Permet à l'acteur de se mettre en sommeil lorsqu'il n'est pas en train de bouger.

```
actor.setVelocity(x, y)
```
Change la velocité (déplacement à vitesse constante) de l'acteur sur les axes x et y.

```
actor.setWidth(value)
```
Spécifie la largeur de l'acteur.

```
actor.setX(x)
```
Change la position de l'acteur sur l'axe x.

```
actor.setXVelocity(x)
```
Change la velocité (déplacement à vitesse constante) de l'acteur sur l'axe x.

```
actor.setY(y)
```
Change la position de l'acteur sur l'axe y.

```
actor.setYVelocity(y)
```

Change la velocité (déplacement à vitesse constante) de l'acteur sur l'axe y.

```
actor.typeDynamic()
```

Applique le type Dynamique à l'acteur lors de la simulation physique (Affecté par la gravité et les autres acteurs). Idéal pour le jouer et les énemies.

```
actor.typeKinematic()
```

Applique le type Cinématique à l'acteur lors de la simulation physique (N'est pas affecté par l'environnement mais peut se déplacer). Idéal pour les plateformes en mouvement.

```
actor.typeSensor()
```

Rends inactif la simulation pour l'acteur (N'est pas affecté par l'environnement et n'affecte rien en retour, mais peut se déplacer et recevoir les évènements de collisions). Parfait pour les objets comme les pièces, bonus etc.

```
actor.typeStatic()
```

Applique le type Statique à l'acteur lors de la simulation physique (N'est pas affecté par l'environnement et ne peut pas se déplacer). Idéal pour les murs et le sol.

event

```
actor.onContact(function (self, other, x, y))
```

Exécute la fonction définie lorsque l'acteur entre en contact avec un autre. Self renvoie l'acteur lui-même, other, l'acteur avec lequel il est rentré en contact.

```
actor.onContactWith(with, function (self, other, x, y))
```

Exécute la fonction définie lorsque l'acteur entre en contact avec un certain autre acteur ou groupe d'acteurs (définit par l'attribut groupId). With peut être un simple acteur/groupId or un tableau d'acteurs/groupIds.

```
actor.onEndContact(function (self, other, x, y))
```

Exécute la fonction définie lorsque l'acteur n'est plus en contact avec un autre. Self renvoie l'acteur lui-même, other, l'acteur avec lequel il est rentré en contact.

```
actor.onEndContactWith(with, function (self, other, x, y))
```

Exécute la fonction définie lorsque l'acteur n'est plus en contact avec un certain autre (défini par le paramètre with). With peut être un simple acteur or un tableau d'acteurs.

```
actor.onPulse(function (self, delta, loopCount), delay[, count[, function (self,
delta, loopCount)]])
```

Exécute la fonction définie de façon répété selon le delai souhaité.

Count indique le nombre de repétition avant l'arrêt et la seconde fonction, la fonction de callback appelé après le dernier appel.

actorAnim

ActorAnim est une acteur composé par un tableau d'images. L'image peut être animé en jouant la séquence du tableau.

L'objet actorAnim hérite de l'objet actor donc, toutes les méthodes de l'objet actor lui sont

accessibles. Voir : actor pour une référence complète.

L'objet actorAnim doit être créé par la méthode de fabrication : layer.createActorAnim(). Voir : layer.createActorAnim() pour une référence complète.

method

`actorAnim.getCurrentFrame()`

Retourne l'image courante de l'animation.

`actorAnim.getFrameDuration()`

Retourne le temps d'une image de l'animation.

`actorAnim.gotoFrame()`

Va à une image spécifique de l'animation.

`actorAnim.play()`

Joue l'animation contenue dans l'image fragmentée.

`actorAnim.repeat()`

Configure l'animation en mode répétition.

`actorAnim.reverse()`

Inverse le sens de lecture de l'animation.

`actorAnim.sequence()`

Configure l'animation en mode séquence (s'arrête à la fin de l'animation).

```
actorAnim.setFrameDuration(duration)
```
Indique le temps d'une image de l'animation.

```
actorAnim.stop()
```
Arrête (met en pause) l'animation contenue dans l'image fragmentée.

```
actorAnim.yoyo()
```
Configure l'animation en mode yoyo.

button

Button est un control graphique.

L'objet button doit être créé par la méthode de fabrication : layer.createButton(). Voir : layer.createButton() pour une référence complète.

```
button.getText()
```
Retourne le text du bouton.

```
button.move(x, y)
```
Change la position du bouton.

```
button.setText(message)
```
Modifie le text du bouton.

event

```
button.onTap(function (x, y, button))
```
Exécute la fonction définie lorsque l'utilisateur tape (ou clique) sur l'acteur.

```
button.onTouch(function (x, y, button))
```
Exécute la fonction définie lorsque l'utilisateur touche (ou drag) sur l'acteur.

```
button.onTouchPulse(delay, function (x, y, button))
```
Exécute la fonction définie de façon répété lorsque l'utilisateur touch l'acteur.

```
button.onUp(function (x, y, button))
```
Exécute la fonction définie lorsque l'utilisateur relâche l'acteur.

buttonImage

ButtonImage est un control graphique.

L'objet buttonImage doit être créé par la méthode de fabrication : layer.createButtonImage(). Voir : layer.createButtonImage() pour une référence complète.

method

```
buttonImage.move(x, y)
```

Change la position du bouton.

event

```
buttonImage.onTap(function (x, y, button))
```

Exécute la fonction définie lorsque l'utilisateur tape (ou clique) sur l'acteur.

```
buttonImage.onTouch(function (x, y, button))
```

Exécute la fonction définie lorsque l'utilisateur touche (ou drag) sur l'acteur.

```
buttonImage.onTouchPulse(delay, function (x, y, button))
```

Exécute la fonction définie de façon répété lorsque l'utilisateur touch l'acteur.

```
buttonImage.onUp(function (x, y, button))
```

Exécute la fonction définie lorsque l'utilisateur relâche l'acteur.

checkBox

CheckBox est un control graphique.

L'objet checkBox doit être créé par la méthode de fabrication : layer.createCheckBox(). Voir : layer.createCheckBox() pour une référence complète.

method

```
checkBox.getText()
```
Retourne le text du bouton.

```
checkBox.move(x, y)
```
Change la position du bouton.

```
checkBox.setText(message)
```
Modifie le text du bouton.

event

```
checkBox.onTap(function (x, y, button))
```
Exécute la fonction définie lorsque l'utilisateur tape (ou clique) sur l'acteur.

```
checkBox.onTouch(function (x, y, button))
```
Exécute la fonction définie lorsque l'utilisateur touche (ou drag) sur l'acteur.

```
checkBox.onTouchPulse(delay, function (x, y, button))
```
Exécute la fonction définie de façon répété lorsque l'utilisateur touch l'acteur.

```
checkBox.onUp(function (x, y, button))
```

Exécute la fonction définie lorsque l'utilisateur relâche l'acteur.

font

Font est une police de caractère et sa taille pour écrire.

L'objet font doit être créé par la méthode de fabrication : game.createFont(). Voir : game.createFont() pour une référence complète.

exemple:

```
1  // create font in game
2  set font = game.createFont("emulogic", 12, game.color.WHITE);
3
4  // initialize stage and layer
5  set stage = game.createStage(0, 0);
6  set layer = stage.createLayer();
7
8  // use font with a label
9  set label = layer.createLabel(font, "Hi ALGEA!", 50, 50);
```

label

Label est un control graphique.

L'objet label doit être créé par la méthode de fabrication : layer.createLabel(). Voir : layer.createLabel() pour une référence complète.

method

```
label.getText()
```

Retourne le text du label.

```
label.move(x, y)
```

Change la position du bouton.

```
label.setText(message)
```

Modifie le text du label.

event

```
label.onTap(function (x, y, button))
```

Exécute la fonction définie lorsque l'utilisateur tape (ou clique) sur l'acteur.

```
label.onTouch(function (x, y, button))
```

Exécute la fonction définie lorsque l'utilisateur touche (ou drag) sur l'acteur.

```
label.onTouchPulse(delay, function (x, y, button))
```

Exécute la fonction définie de façon répété lorsque l'utilisateur touch l'acteur.

```
label.onUp(function (x, y, button))
```

Exécute la fonction définie lorsque l'utilisateur relâche l'acteur.

layer

Les layers sont un ensemble de calques superposés, comme des papiers calques les uns sur les autres.

exemple:

```
1  // initialize stage and layer
2  set stage = game.createStage(0, -10);
3  set layer = stage.createLayer();
4
5  // create background
6  layer.createRepeatedImage("PlanetCute/Sky");
```

factory

```
layer.createActor(path[, x, y])
```

Crée un nouvel acteur graphique et physique dans le calque.

```
layer.createActorAnim(path, divWidth, divHeight[, x, y])
```

Crée un nouvel acteur animé dans le calque.

Les paramètres divWidth and divHeight représentent le nomvre d'image par colonne et par ligne du cadrillage.

```
layer.createButton(font, path, text[, x, y])
```

Crée un nouveau bouton dans le claque.

```
layer.createButtonImage(path[, x, y])
```

Crée un nouveau bouton image dans le claque.

```
layer.createCheckBox(font, text[, x, y])
```

Crée une nouvelle check box dans le claque.

```
layer.createLabel(font, text[, x, y])
```

Crée un nouveau champ text dans le claque.

```
layer.createPad(path[, x, y])
```

Crée un nouveau joypad dans le claque.

```
layer.createParticle(path[, x, y])
```

Crée un nouvel effet de particules dans le claque.

```
layer.createRepeatedImage(path)
```

Crée une nouvelle image qui se répète (pour le background) dans le claque.

```
layer.createVerticalLayout(path[, x, y])
```

Crée un nouveau conteneur d'objet à disposition vertical dans le claque.

```
layer.createWall(x, y, width, height)
```

Crée un nouveau mur virtuel (juste pour la physique) dans le claque.

method

```
layer.follow(actor)
```

Suit l'acteur en bougeant (scroll) le calque.

```
layer.getHeight()
```

Retourne la hauteur du calque (Représente la bounding box de tous les acteurs contenus par le calque).

```
layer.getWidth()
```

Retourne la largeur du calque (Représente la bounding box de tous les acteurs contenus par le calque).

```
layer.remove(actor)
```

Enlève l'acteur du calque.

```
layer.scroll(x, y)
```

Scroll la caméra du layer.

```
layer.scrollX(x)
```
Scroll la caméra du layer.

```
layer.scrollY(y)
```
Scroll la caméra du layer.

event

```
layer.onScroll(function (x, y))
```
Exécute la fonction définie lorsque le calque bouge (scroll).

particle

L'objet particle est une animations de particules. Utiliser 2d particle editor pour en créer.

method

```
particle.move(x, y)
```
Change la position du bouton.

repeatedImage

RepeatedImage est utilisé pour créer comme un papier peint qui se répète.

L'objet repeatedImage hérite de l'objet actor donc, toutes les méthodes de l'objet actor lui sont accessibles. Voir : actor pour une référence complète.

exemple:

```
1  // initialize stage and layer
2  set stage = game.createStage(0, -10);
3  set layer = stage.createLayer();
4
5  // create background
6  layer.createRepeatedImage("PlanetCute/Sky");
```

sequence

L'objet sequence définit une suite d'animation.

`sequence.add(tween)`

Ajoute une nouvelle animation dans la séquence.

```
sequence.pause()
```
Met la séquence en pause.

```
sequence.repeat()
```
Répète la séquence de façon automatique.

```
sequence.repeatYoyo()
```
Répète la séquence d'animations de façon automatique comme un yoyo (va et vient).

```
sequence.resume()
```
Redémarre la séquence (après une pause).

```
sequence.start()
```
Démarre la séquence.

event

```
sequence.onDone(function ())
```
Exécute la fonction définie lorsque la séquence d'animations est finie.

sound

Sound est un son ou une musique à jouer.

L'objet sound doit être créé par la méthode de fabrication : game.createSound(). Voir : game.createSound() pour une référence complète.

exemple:

```
1  set music = game.createSound("Loop/Vibraphone")
2  music.playLoop();
```

method

`sound.pause()`

Met le son en pause.

`sound.play([volume [, pitch [, pan]]])`

Joue le son. Le volume est dans l'interval 0 à 1. Le pitch représente la rapidité (<1 lentement, 1 normal et >1 rapide). Le panning control la position des hauts parleurs: -1 à gauche, 0 au centre et 1 à droite.

`sound.playLoop([volume [, pitch [, pan]]])`

Joue le son et le repete. Le volume est dans l'interval 0 à 1. Le pitch représente la rapidité (<1 lentement, 1 normal et >1 rapide). Le panning control la position des hauts parleurs: -1 à gauche, 0 au centre et 1 à droite.

`sound.resume()`

Après la pause, continue le son.

`sound.stop()`

Arrête le son.

stage

Crée un nouveau calque dans le stage.

L'objet stage doit être créé par la méthode de fabrication : game.createStage(). Voir : game.createStage() pour une référence complète.

exemple:

```
1  // initialize stage and layer
2  set stage = game.createStage(0, -10);
3  set layer = stage.createLayer();
```

factory

`stage.createLayer()`

Crée un nouveau calque dans le stage.

`stage.createSequence()`

Crée une nouvelle séquence d'animation (animation composée).

```
stage.createTiledLayer(path)
```
Crée un nouveau calque à base de tiled map dans le stage.

Les tiled map doivent êtres créées avec le logiciel Tiled map editor, qui ce trouve à l'adresse map editor

```
stage.createTween(getters, setter, duration, targets)
```
Crée un nouvelle animation.

Getters est/sont la fonction ou l'animation va trouver la valeur initial de l'objet à animer

Setter est la fonction qui sera appelé pour régulièrement pour animer l'objet.

Targets est/sont la valeur à atteindre par l'animation.

Getters et Targets peuvent recevoirent un tableau. Le nombre d'argument doit correspondre au nombre d'attribut de la fonction setter. Ainsi, l'animation prend en compte les méthodes à plusieur arguments.

method

```
stage.getHeight()
```
Retourne la hauteur du stage (Représente la taille de l'écran, ne pas confondre avec la méthode layer.getHeight()).

```
stage.getWidth()
```
Retourne la largeure du stage (Représente la taille de l'écran, ne pas confondre avec la méthode

layer.getWidth()).

event

```
stage.onAccelerometer(function (x, y, z))
```

Exécute la fonction définie lorsqu'une l'accéléromètre de l'appareil detecte un mouvement.

```
stage.onCompass(function (x, y, z))
```

Exécute la fonction définie lorsqu'une la boussole de l'appareil detecte un changement d'azimut.

```
stage.onKey(function (char))
```

Exécute la fonction définie lorsqu'une touche clavier est pressée.

```
stage.onKeyPulse(function (char))
```

Exécute la fonction définie lorsqu'une touche clavier est pressée et à intervale régulier tant que la touche reste enfoncé.

```
stage.onKeyUp(function (char))
```

Exécute la fonction définie lorsqu'une touche clavier est relachée.

```
stage.onTap(function (x, y, button))
```

Exécute la fonction définie lorsque l'utilisateur tape (ou clique) sur l'acteur.

```
stage.onTouch(function (x, y, button))
```

Exécute la fonction définie lorsque l'utilisateur touche (ou drag) sur l'acteur.

```
stage.onTouchPulse(delay, function (x, y, button))
```

Exécute la fonction définie de façon répété lorsque l'utilisateur touch l'acteur.

```
stage.onUp(function (x, y, button))
```

Exécute la fonction définie lorsque l'utilisateur relâche l'acteur.

touchPad

TouchPad est un control graphique.

L'objet touchPad doit être créé par la méthode de fabrication : layer.createTouchPad(). Voir : layer.createTouchPad() pour une référence complète.

method

```
touchPad.getPrecentX()
```

Retourne le pourcentage de déplacement du knob sur l'axe X en pourcentage (de 0 à 1).

```
touchPad.getPrecentY()
```

Retourne le pourcentage de déplacement du knob sur l'axe Y en pourcentage (de 0 à 1).

```
touchPad.move(x, y)
```

Change la position du bouton.

event

```
touchPad.onTap(function (x, y, button))
```

Exécute la fonction définie lorsque l'utilisateur tape (ou clique) sur l'acteur.

```
touchPad.onTouch(function (x, y, button))
```

Exécute la fonction définie lorsque l'utilisateur touche (ou drag) sur l'acteur.

```
touchPad.onTouchPulse(delay, function (x, y, button))
```

Exécute la fonction définie de façon répété lorsque l'utilisateur touch l'acteur.

```
touchPad.onUp(function (x, y, button))
```

Exécute la fonction définie lorsque l'utilisateur relâche l'acteur.

tween

L'objet tween définit une animation.

exemple:

```
1  // initialize stage
2  set stage = game.createStage(0, 0);
```

```
set layer = stage.createLayer();

// create actor
set cloud = layer.createActor("Platformer/Cloud1");
cloud.move(500, 200);
cloud.typeSensor().hitSquare();

// simple anim
stage.createTween(cloud.getY, cloud.setY, 2, 500).repeatYoyo().start();

```

Il peut être utilisé pour animer des méthodes avec plusieurs paramètres comme move.

exemple:

```
// multi parameter animation
// initialize stage
set stage = game.createStage(0, 0);
set layer = stage.createLayer();

// create actor
set cloud = layer.createActor("Platformer/Cloud1");
cloud.move(500, 200);
cloud.typeSensor().hitSquare();

// simple anim
stage.createTween(
    array{cloud.getX, cloud.getY}, cloud.move, 2, array{700, 500}
  ).repeatYoyo().start();
```

method

`tween.pause()`

Met l'animation en pause.

`tween.repeat()`

Répète l'animation de façon automatique.

`tween.repeatYoyo()`

Répète l'animation de façon automatique comme un yoyo (va et vient).

`tween.resume()`

Redémarre l'animation (après une pause).

`tween.setEase(formula)`

Spécifie la formule de progression de l'animation.

`tween.start()`

Démarre l'animation.

event

`tween.onDone(function ())`

Exécute la fonction définie lorsque l'animation est finie.

verticalLayout

VerticalLayout est un conteneur de control graphiques.Il organise ses enfant verticalement.

L'objet verticalLayout doit être créé par la méthode de fabrication : layer.createVerticalLayout().
Voir : layer.createVerticalLayout() pour une référence complète.

exemple:

```
1   // create stage
2   set stage = game.createStage(0, 0);
3   set layer = stage.createLayer();
4
5   // create layout
6   set vl = layer.createVerticalLayout(stage.getWidth() / 2, stage.getHeight() /
7   2);
8
9   // create buttons with event
    vl.addActor (
10    layer.createButton(font, "orange", "Space battle")
        .onTap( {load("SpaceBattle")} )
11    )

12  vl.addActor (
      layer.createButton(font, "orange", "Jumper")
13      .onTap( {load("Jumper")} )
      )
14

15

16

17
```

method

`verticalLayout.addActor(actor)`

Ajoute un acteur dans le layout verticale.

`verticalLayout.getSpace()`

Retourne l'espace entre les éléments.

`verticalLayout.move(x, y)`

Change la position du bouton.

`verticalLayout.setSpace(size)`

Change l'espace entre les éléments.

Forme de Backus-Naur d'Algoid Language

```
// expressions et précédence des opérateurs
assign ::= var ident ( "=" | "+=" | "-=" | "*=" | "/=" | "%=" | "..=" | "->=" ) expr [ ";" ]
expr ::= concat | assign
concat ::= or [ ( ".." | "->" ) or ]
or ::= and [ "| |" and ]
and ::= compar [ "&&" compar ]
compar ::= oper [ "==" | "!=" | "<" | "<=" | ">" | ">=" oper ]
oper ::= mult [ "+" | "-" mult ]
mult ::= unary [ "*" | "/" | "%" unary ]
unary ::= ( ["-" | "!" | new ] ident) | ( ident [ "++" | "--" ] )

// identifiants
ident ::= call [ "." call ]
```

```
call ::= index "(" [ expr [ { "," expr } ] ] ")"
index ::= value [ { "[" expr "]" } ]

// valeurs
boolean ::= true | false
number ::= nan | infinity | ("0"-"9") [ { ("0"-"9") } ]
string ::= """ [ { .. caractère ASCII .. } ] """
value ::= "(" expr ")" | object | lambda | function | array | if | this | supers | ident | nil | boolean |
number | string

// structures
assign ::= set ident "=" expr [ ";" ]
lambda ::= [ "(" [ ident ] [ { "," ident } ] ")" ] "{" body "}"
function ::= function "(" [ ident ] [ { "," ident } ] ")" "{" body "}"
return ::= return expr [ ";" ]
array ::= array<1*> [ "(" ")" ] "{" [ item [ { "," item } ] ] "}"
item ::= [ expr ":" ] expr
object ::= object "(" [ ident [ "," ident ] ] ")" "{" [ { assign } ] "}"

// impératifs
if ::= if "(" expr ")" body [ { elseif "(" expr ")" body } ] [ else body ]
loop ::= loop "(" [ expr ] ")" body
for ::= for "(" [ assign ] ";" [ expr ] ";" [ expr] ")" body
while ::= while "(" expr ")" body
until ::= do body until "(" expr ")"
break ::= break [ ";" ]

// autres
comment ::= ( "//" .. caractère ASCII .. .. fin de ligne .. ) | ( "/*" .. caractère ASCII .. "*/")
body ::= instruction | "{" [ { instruction } ] "}"
instruction ::= comment | return | loop | for | while | until | break | expr [ ";" ] | ";" | body
```