AOZ STUDIO

GUIDE DE L'UTILISATEUR

1. QUE LA MAGIE COMMENCE

Bonjour, bienvenue !

Ravi de vous accueillir dans le monde magique d'AOZ Studio.

Si vous ne connaissez rien à la programmation, mais que vous voulez publier une application, un jeu, ou le site web de vos rêves, vous êtes au bon endroit. Si vous êtes déjà un pro et désirez développer beaucoup plus vite, vous êtes au bon endroit. Si vous désirez commencer à programmer tout de suite, vous êtes au bon endroit, sinon vous pouvez commencer par lire : *AOZ Studio -Introduction.*

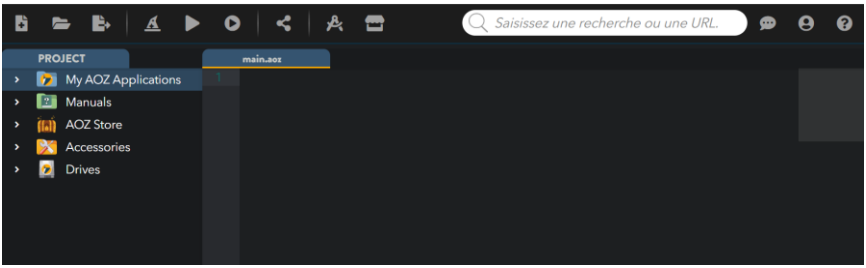
INSTALLATION D'AOZ STUDIO

Très bonne nouvelle : seule l'installation de l'application « AOZ Studio » sur votre PC ou MAC est nécessaire. Tout est inclus dans AOZ Studio : une fois l'installation terminée, vous pouvez commencer à programmer. Donc je confirme, il n'y a pas 20 pages sur la façon d'installer et de configurer l'éditeur, le compilateur, les librairies, SDK, API...

Pour obtenir la dernière version d'AOZ Studio, c'est là : www.aoz.studio

VOTRE PREMIER PROGRAMME

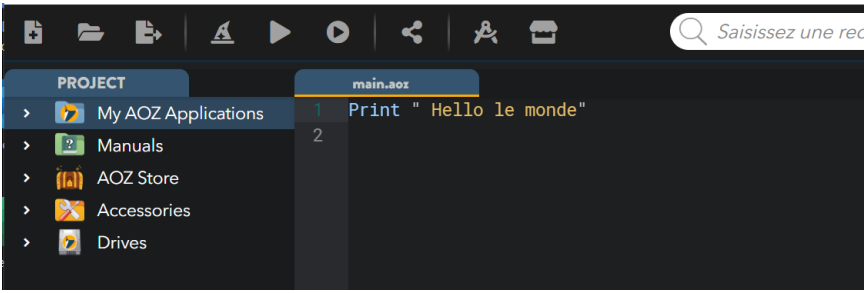
Lancez le programme AOZ Studio depuis votre PC ou MAC.



1°) Regardez le grand espace vide dans AOZ Studio, tapez là votre première ligne de code (allez, on n'a pas peur !) :

Print "Hello le monde"

Comme ceci :



2°) Découvrez tout en haut la barre des boutons :

3°) et cliquez sur le bouton **RUN** (ou sur la touche de fonction F2)

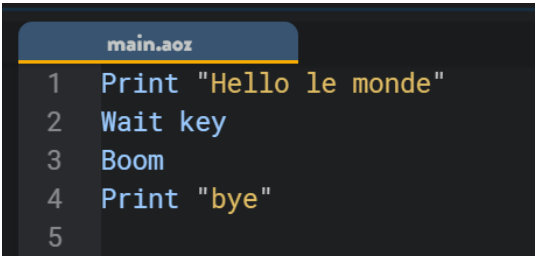
Attendez quelques secondes, cliquez, et vous pouvez voir s'afficher dans une nouvelle fenêtre le résultat de votre programme : "Hello le monde".

Félicitations ! Vous êtes maintenant un programmeur.

Si vous pensez que c'est magique, vous avez raison. Ce que vous venez de faire c'est de donner à votre ordinateur une instruction pour afficher quelque chose sur l'écran. Pour ce faire, vous avez utilisé l'instruction **Print**, (oui je sais les instructions sont en Anglais, comme dans la plupart des langages de programmation).
Print est une instruction de la langue AOZ que votre ordinateur comprend et qui lui dit d'afficher ce qui est entre guillemets.

Ok, on continue.

Tapez ces lignes de programme :



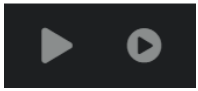
Vous remarquerez qu'AOZ Studio essaie de vous aider en proposant de l'aide sur les instructions que vous tapez, et il va même afficher les différents éléments de votre code dans des couleurs différentes. Par exemple, les instructions, comme **Print**, sont affichées en bleu, les valeurs comme « bye » en jaune.

Avez-vous des questions sur votre nouveau programme ?

- L'instruction **Wait Key**, dit à l'ordinateur (ou au smartphone) d'attendre jusqu'à ce qu'une touche soit pressée (Wait = attendre, Key = touche)
- L'instruction **Boom**, devinez...

Donc, ce programme va imprimer « Hello le monde », attendre que l'on appuie sur une touche, puis jouer le son boom et poliment dire au revoir. Nous conseillons de toujours programmer poliment.

Pour voir ce programme fonctionner cliquez sur l'un des deux boutons **RUN**, oui en fait, nous en avons deux en haut de l'écran d'AOZ Studio.



Il y en a deux car notre programme peut fonctionner dans :

- un navigateur Web (comme Chrome, Edge)

  ou dans
- AOZ Studio, comme nous venons de le faire

Donc cliquez sur RUN, attendez quelques secondes, et votre programme s'exécutera automatiquement.

Maintenant que vous connaissez les principes – de base, nous allons pouvoir faire beaucoup plus que Boom. Mais d'abord il nous faut voir quelques trucs ennuyeux ; promis c'est rapide !

2. VUE D'ENSEMBLE

Ce chapitre va vous donner un aperçu d'AOZ Studio et de sa langue AOZ. Bon OK, si vous êtes vraiment impatient ou un programmeur confirmé vous pouvez passer aux chapitres suivants pour créer votre propre programme.

MOTS CLÉS ET SYNTAXE

La langue AOZ, se compose d'instructions ou mots clés, et comme toute autre langue elle utilise un ensemble de règles appelées syntaxe. Connaître les mots clés et la syntaxe c'est tout ce dont vous avez besoin pour créer vos merveilles.
Le vocabulaire AOZ est constitué de plus de 800 mots clés. Quant à la syntaxe, elle est simple, pas de conjugaisons, d'irrégularités. De plus AOZ Studio est là pour vous aider, il vous suggérera des instructions et vous guidera pour leur syntaxe.

BASIC

AOZ est basée sur un langage très connu : le Basic, qui signifie *Beginner's All-purpose Symbolic Instruction Code*.

Toutes les instructions sont définies dans la langue AOZ et tout est modulaire, ce qui est juste une autre façon de dire que tout ce dont vous avez besoin pour vos créations est intégré et que vous pouvez les combiner, les organiser comme vous le souhaitez.

AOZ utilise le standard Basic comme base mais l'a très largement étendu dans ses capacités (y compris avec la "programmation objet".)

SIMPLE ET PUISSANT
AOZ est facile à apprendre, mais il est surtout très puissant, ce qui le rend également très efficace pour les professionnels endurcis. Vous pourrez créer le jeu de vos rêves, l'application mobile ou un site Web en une fraction du temps nécessaire avec d'autres outils.
AOZ permet à votre ordinateur de gérer de grandes quantités de données et de pixels à l'écran, vos applications s'exécuteront très rapidement.

SOYONS SÉRIEUX
Si votre intérêt n'est pas dans les jeux mais se porte sur le développement d'autres types d'applications, alors vous ne serez pas déçu. AOZ permet de faire beaucoup de choses, y compris dans le domaine des bases de données, des utilitaires, des interfaces utilisateur.

MODERNE, UNIVERSEL ET RAPIDE
Vos applications AOZ parleront aux ordinateurs, smartphones, comme à tous les appareils avec un navigateur Web, en utilisant des technologies modernes comme les dernières bibliothèques Node.js et JavaScript.

EXTENSIBLE
Si les 800 instructions d'AOZ ne vous suffisent pas AOZ Studio vous permet de créer vos propres instructions et fonctions (ensemble d'instructions permettant d'exécuter une tâche),. Vos nouvelles instructions et fonctions peuvent ensuite être utilisées dans vos applications, publiées sur Internet et réutilisées par d'autres. Vous pouvez programmer vos nouvelles instructions en AOZ lui-même, en utilisant rien d'autre que de simples instructions AOZ, ou en JavaScript (c'est un langage de programmation qui permet d'implémenter des mécanismes complexes sur une page web).

TRANSPILER

AOZ Studio intègre un transpiler : un traducteur intelligent qui prend le code d'une langue et le convertit en une autre langue. Dans notre cas, le transpiler AOZ prend le code de la langue AOZ, et le traduit en quelque chose appelé JavaScript et quelque chose appelé code HTML5 (c'est ce qui fonctionne sur les navigateurs Web).

Votre code AOZ une fois transpilé devient ultra-rapide et portable, il pourra fonctionner sur tous les navigateurs Web de n'importe quelle machine compatible comme les PC, MAC, les smartphones. On verra plus tard que vous pourrez également faire fonctionner vos programmes comme des applications autonomes (.exe pour PC, .apk pour Android…).

Donc, pour résumer, votre programme créé en langue AOZ est converti par le transpileur d'AOZ Studio en JavaScript, le langage le plus couramment utilisé. Il était important pour nous de vous aider à passer d'une langue à l'autre : vous pourrez mélanger du AOZ et du Javascript, mais nous parions que vous utiliserez surtout AOZ, comme beaucoup de développeurs professionnels qui programment ainsi deux fois plus rapidement.

GRATUIT OU PAYANT

Nous sommes si gentils que nous avons fait une version gratuite d'AOZ Studio (au passage AOZ Studio est open source hors le transpiler). Cela signifie que vous pouvez l'utiliser gratuitement. Nous espérons que vous l'aimerez tellement que vous voudrez bien nous payer une licence extrêmement modique, quasi rien ; parce que nous devons nourrir les magiciens d'AOZ Studio, faire les potions magiques…

- La version gratuite est complète et vous pouvez créer un nombre illimité de programmes. Elle comprend de la publicité et la limite est que vous seul êtes autorisé à utiliser vos programmes.

- Le paiement de la licence (version payante) permet de ne pas avoir d'annonces publicitaires et de partager vos créations, de les publier, ou les vendre à qui vous voulez, sur toutes machines compatibles, et sans nous payer de redevances. Après tout, vous devez garder les récompenses de votre travail.

Et nous sommes si gentils qu'AOZ Studio est gratuit pour les organisations humanitaires, d'éducation pour les enfants, à but non lucratif.

Si vous êtes un développeur expérimenté, n'oubliez pas que le code AOZ est Open Source (Domaine Publique). Vous pouvez créer vos propres branches, optimiser le code, insérer vos propres idées et réalisations. Vous serez plus que bienvenu pour aider au développement d'AOZ Studio.

Peu importe que vous soyez un codeur expérimenté, paresseux, ou débutant ; vous êtes le bienvenu sur le DISCORD d'AOZ Studio (c'est une application chat).

Vous y rencontrerez une grande communauté de magiciens bienveillants qui vous aideront, et vous pourrez échanger directement avec l'équipe d'AOZ Studio :
https://discord.com/invite/JmFyFRA

Découvrez la chaîne Youtube AOZ Studio

3. NOS PREMIERS PROGRAMMES

Afficher des images avec Actor

Vous avez hâte de faire bouger des images à l'écran ?

Nous vous comprenons, et c'est pourquoi nous avons créé l'instruction **Actor,** de sorte que vous soyez le réalisateur et dirigiez vos acteurs.

Il existe plusieurs instructions AOZ pour manipuler des images, mais l'instruction Actor est la plus puissante. Elle simplifie l'affichage des images, gère les animations, contrôle les collisions, elle vous permettra de faire des choses magiques.

Un « Actor » est un personnage, mais plus globalement un élément graphique. Par exemple, dans un jeu, c'est le personnage que le joueur contrôle, ou un ennemi, c'est aussi un élément du paysage, une image... En d'autres termes, c'est un élément visuel, essentiel pour un jeu comme pour une application professionnelle. Commençons donc par là.

Notes pour les programmeurs plus expérimentés :

- Si vous souhaitez visualiser ou déplacer une image, animée ou non, vous pouvez utiliser soit l'instruction BOB, qui est compatible AMOS, soit l'instruction Actor dont nous parlons dans ce chapitre.
- Si vous souhaitez programmer en mode Objet (OOP), l'instruction BOB est également compatible OOP avec des paramètres similaires à Actor.

Voyons cet Actor

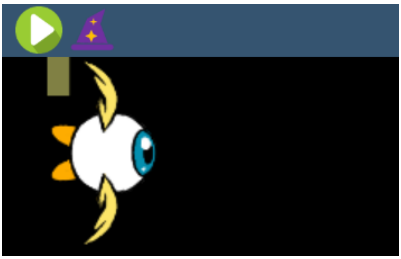
L'instruction Actor comprend de nombreuses options, que l'on appelle paramètres et qui ont eux-mêmes de nombreuses propriétés ou valeurs, c'est ce qui en fait sa puissance.

Mais ne vous inquiétez pas, la plupart des paramètres ont des valeurs par défaut. Ainsi vous pouvez ne modifier que les paramètres qui vous intéressent.

C'est parti alors ! Après avoir effacé toutes les lignes des programmes qui étaient à l'écran, tapez le code suivant :

Actor "magicien", Image$="magic.png"

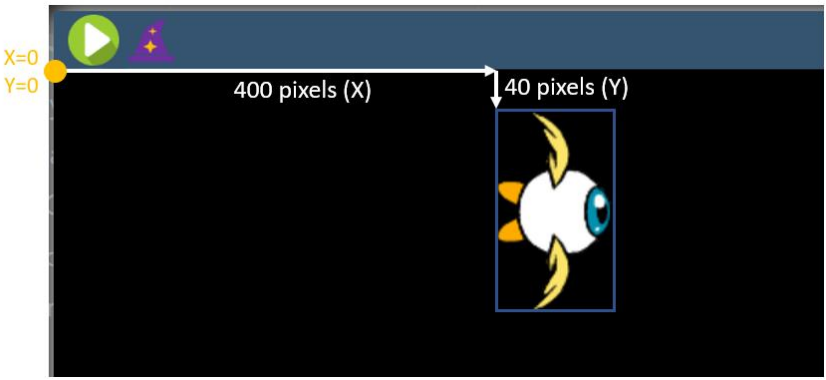
Maintenant examinons ce programme en cliquant sur le bouton **RUN** pour afficher l'image « magic» sur l'écran.



Vous avez créé un actor, son nom est « magicien » et son image est le fichier « magic.png». C'est aussi simple que ça !

Maintenant, ajoutez deux paramètres appelés X et Y au code:

Actor "magicien", Image$="magic.png", X=400, Y=40



Examinons de plus près les différents paramètres utilisés dans cet exemple :

« nom de l'actor »
Le nom de l'actor, qui est « magicien » dans cet exemple, est le premier paramètre, et le seul que vous devez spécifier à chaque fois. Les autres paramètres de l'instruction Actor sont facultatifs. Vous pouvez lui donner un nom, ou un numéro, par exemple :

Actor 10, Image$="magic.png", X=400, Y=40

Nous verrons plus tard qu'il peut être commode d'utiliser des nombres pour déplacer plusieurs actors en même temps. Vous êtes libre de choisir des noms ou des nombres, mais évitez d'utiliser des espaces et des accents. Soit dit en passant, si le nom est déjà utilisé par un autre actor, alors c'est l'autre, le premier défini, qui sera affecté par vos paramètres.

Image$ = "nom de l'image"
L'image définit l'apparence ou le « costume » que l'actor portera. C'est littéralement son image.

Une fois l'image définie dans l'instruction actor, cette image sera toujours utilisée pour cet actor et vous n'avez plus à le préciser. Et si aucune image n'est définie, alors l'instruction Actor ne montrera rien.

Vous pouvez utiliser de nombreux formats d'images comme PNG, BMP, JPEG.

X- position horizontale, position verticale Y

Ces deux paramètres définissent la position d'affichage de l'actor à l'écran, les valeurs de X et Y sont en pixels (le plus petit point affiché sur l'écran). Dans notre exemple, l'image apparaît en position 400 (horizontale), 40 (verticale) à partir du coin supérieur gauche de l'écran.

Rappelez-vous, ces paramètres sont facultatifs, tout comme les autres. Si X ou Y ne sont pas définis, les dernières valeurs de X ou Y seront utilisés. Par exemple si vous avez placé un actor à 50, 50 dans votre programme et que vous n'attribuez pas de valeurs à X et Y dans une nouvelle instruction **Actor**, la position de départ restera toujours 50,50.
Note : par défaut si X et Y ne sont pas définis la position de départ de l'actor sera 0,0. Oui je sais c'est logique.

AOZ s'adapte à votre logique

AOZ vous permet de choisir l'ordre de votre potion magique. Si vous préférez :

Actor 10, X=400, Y=40, Image$="magic.png"
Plutôt que :
Actor 10, Image$="magic.png", X=400, Y=40
La magie fonctionnera toujours !

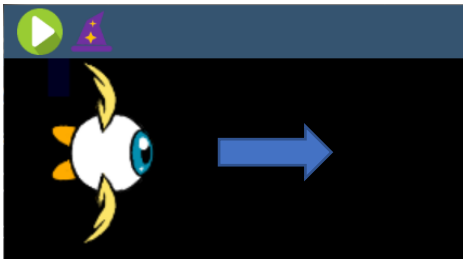
DÉPLACER L'ACTOR

Maintenant que nous savons afficher un actor, changeons notre code pour déplacer l'actor à travers l'écran.

Nous allons changer notre programme et ajouter un nouveau paramètre, appelé **EndX** comme ceci :

```
Actor "magicien", X=0, Y=0, Image$="magic.png", EndX=1980
```

Exécutez le programme en cliquant sur RUN (ou appuyez sur la touche de fonction F2), et votre actor se déplace de gauche à droite de l'écran.



Note : pour réaliser la même chose dans la plupart des autres langages, cela nécessite 20 à 100 lignes de code. Chaque langage a ses caractéristiques, ses forces et ses faiblesses, AOZ a été conçu pour offrir une puissante simplicité de programmation.

Détaillons notre programme. Nous avons défini le nouveau paramètre **EndX** =1980. **EndX** indique à notre actor qu'il doit se déplacer de sa position actuelle (X,Y) à une position finale définie par **EndX** (EndY si on le défini). Lorsque cette position est atteinte, le mouvement de l'actor s'arrête.

N'oubliez pas, vous pouvez mettre vos paramètres dans l'ordre que vous voulez, ainsi vous pouvez également écrire :
Actor "magicien", Image$="magic.png", X=0, EndX=1980, Y=0

Pour résumer : le point de départ et le point final du mouvement est automatiquement pris en charge par l'instruction **Actor**, et commence immédiatement. Dans ce cas de X=0, Y=0 jusqu'à ce que la position X soit égale à la valeur définie par le paramètre **EndX,** qui est ici 1980.

Nous verrons plus loin que le paramètre **duration** modifie la vitesse de mouvement.

À vous de jouer 😊 :

- Arrêtez l'actor au milieu de l'écran.
- Déplacez l'actor verticalement en utilisant le même principe que le code ci-dessus.
- Déplacez l'actor de la droite de l'écran vers la gauche.
- Déplacez l'actor en diagonale.
- Utilisez l'image lucie.png au lieu de magic.png

PAUSE, DO…LOOP, IF…THEN

Voici quelques autres concepts importants à connaitre pour un programmeur.

Par défaut, le mouvement de l'actor démarre automatiquement, mais dans certains cas, vous voudrez le contrôler. Par exemple, pour le mettre en pause quand vous le souhaitez. Modifions donc notre code :

```
Actor "magicien", X=10, Y=0, Image$="magic.png", EndX=1980
Do
  A$ = Inkey$
  If A$ = "t" Then Actor "magicien", ActionMove$="pause"
  If A$ = "y" Then Actor "magicien", ActionMove$="play"
Loop
```

Exécutez le programme (RUN). Notre actor s'affiche à l'écran et se déplace. Maintenant, appuyez rapidement sur la touche **t** sur votre clavier : le mouvement s'arrête. Appuyez ensuite sur la touche **y** : le mouvement reprend.
Recommencez, appuyez sur **t** pour faire une pause, puis **y** pour continuer. Puis **t** ….
Stop, vous avez compris !
Vous oui, mais pas par votre ordinateur qui va répéter la boucle indéfiniment sans se fatiguer.
Vous le savez déjà chez AOZ Studio, nous sommes très gentils avec les ordinateurs et les magiciens, alors nous avons de vous

donner le pouvoir d'arrêter un programme en appuyant simultanément sur les touches **Ctrl** et **c** (Les touches Ctrl sont situées en bas, à gauche et à droite de votre clavier.)

Analysons ce programme :

L'instruction **Do ... Loop**

- **Lignes 2 et 6** : le Do ... Loop. C'est une boucle. Il s'agit d'une instruction en deux parties pour indiquer que le morceau de programme entre le Do et le Loop doit être répété en boucle. En d'autres termes, dans la boucle la partie du code se répète pour toujours jusqu'à ce que le programme, ou l'utilisation de Ctrl plus c, décide de mettre fin à cette folie.

- **Ligne 4 et 5 :** un nouveau paramètre est défini pour l'actor : **ActionMove$**. Cela permet d'affecter le mouvement de l'actor. Il y a deux états possibles : **"play"**, c'est la valeur par défaut, et **"pause**", qui interrompt le mouvement.

- **Ligne 3 :** la variable **A$ stocke** la valeur de la touche pressée, retournée par la fonction **Inkey$**
  Note : Si aucune touche n'est pressée, **Inkey$** retourne une valeur vide ou "". Si la touche appuyée est la barre d'espace, la fonction **Inkey$** renvoie un espace ou " ". (Notez qu'il y a un vide/espace entre les deux guillemets.)
  Si **Inkey$** retourne "T" ou "Y" en majuscules et non "t" ou "y", si le clavier est en capitales (touche CAPS-LOCK).

L'instruction **If...Then...Else**

- **Lignes 4 et 5** : examinons maintenant en détail une instruction très importante composée de trois mots, dont seul le mot If est obligatoire.

La célèbre et indispensable instruction **If...Then...Else** :

- **Si** (If en Anglais) **A$** contient la lettre "t" c'est que cette touche a été pressée -> alors nous changeons l'**ActionMove$** de

l'instruction **Actor** avec la propriété **« pause »** qui arrête l'animation.

- **Si A$** contient un espace ("y") alors nous changeons l'**ActionMove$** de l'instruction **Actor** avec la propriété **« play »** qui relance l'animation.

Nous avons maintenant couvert rapidement deux instructions très importantes : la boucle **Do...Loop** et le test **If...Then...Else**.

Vous ne comprenez pas tout ? Quelques essais devraient vous aider et nous reviendrons sur tout cela plus tard.

À vous de jouer 😊 :

- Modifiez les lettres du clavier qui pausent et jouent l'animation.

Comme vous pouvez le voir, avec AOZ, il est très facile de déplacer et contrôler les actors sur l'écran. Si vous avez plusieurs dizaines d'actors à bouger, l'instruction **Actor** fournit les paramètres pour rendre cela facile à gérer.

Continuons et essayons quelque chose de plus compliqué.

Changez le programme comme suit :

```
Actor "magicien", Y=100, Image$="magic.png"
PX=0
Do
  PX = PX + 16
  Actor "magicien", X = PX
  Wait Vbl
Loop
```

Exécutez le programme (RUN). L'actor passe progressivement de gauche à droite de l'écran.

Je sais ce que vous pensez. Vous vous dites : *mais ce code arrive au même résultat que lorsque nous avons utilisé la ligne simple* :
Actor "magicien", Image$="magic.png", Y=100, EndX=1980

Alors pourquoi rendre les choses compliquées quand on peut faire simple ?

Parce qu'il est important de comprendre ce qui se passe derrière une instruction et les valeurs des paramètres.
L'utilisation d'une commande puissante comme Actor ne doit pas vous conduire à ignorer les mécanismes qui lui permettent de produire ses effets.

Étudions ce programme :

- **Ligne 1 :** notre actor est positionné à l'écran (en 0, 100).
- **Ligne 3 et 7 :** la boucle Do ... Loop, vous vous souvenez, indique que nous répétons le code entre le Do et le Loop en continu.
- **Ligne 4 :** la valeur de la variable PX lorsque le programme atteint la ligne 3 pour la 1ère fois est égale à 0 (valeur par défaut). Cette ligne ajoutera 16 à PX, qui vaudra alors 16 (nouveau PX = 0 + 16).
- **Ligne 5 :** nous appelons l'instruction **Actor** pour mettre à jour le paramètre X avec la valeur de la variable PX, maintenant X vaudra 16.

Remarque : comme nous avons déjà défini la valeur de Y et de l'image dans la ligne 1, il n'est pas nécessaire de les redéfinir ici, donc nous gardons la même valeur de Y=100.
Ligne 6 : Wait Vbl est une petite instruction nécessaire pour lisser l'animation en la synchronisant avec votre écran. Nous en parlerons plus tard (cf le chapitre Vitesse et Wait VBL).
Ligne 7 : La boucle **Do... Loop** se termine par le **Loop** indiquant que nous terminons là cette boucle et revenons au début de celle-ci (c.-à-d. à la ligne 2) pour exécuter à nouveau le même code.

En résumé, cette boucle ajoute 16 à la variable PX à chaque fois. Ainsi, notre actor avancera de 16 pixels en X (horizontalement), sans changer Y (verticalement) à chaque passage dans la boucle.

À vous de jouer :
- Déplacez le magicien plus lentement et plus vite (je vous aide : en changeant le paramètre de la ligne PX = PX + 16).
- Déplacez le magicien en diagonale.
- Puis déplacez le dans toutes les directions avec 4 touches en utilisant **Inkey$.**

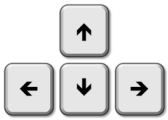
DÉPLACER UN ACTOR AVEC LE CLAVIER

Encore plus de magie. Vous allez automatiquement déplacer votre actor avec les touches fléchées du clavier en utilisant une seule ligne de code.

Effacez tout le programme précédent et tapez :

Actor "magicien", Image$="magic", Control$="Keyboard"

Exécutez ce programme (RUN) et vous verrez l'actor se déplacer dans les différentes directions lorsque vous appuierez sur les 4 touches fléchées de votre clavier.



Si vous êtes un débutant, vous penserez que c'est évident, mais sachez que normalement il faudrait un programme compliqué pour y parvenir. Avec AOZ cette simple ligne suffit.

Pour y parvenir, nous avons ajouté le paramètre **Control$**, qui, détermine comment contrôler l'actor. Ici avec le clavier, mais vous pourriez faire la même chose pour un joystick, une souris ou un écran tactile.

Avec AOZ, vous pouvez programmer léger, simple, en utilisant les paramètres par défaut, puis ajuster ce qui est spécifique.

Remarque : par défaut, les touches utilisées par le **Control$="Keyboard"** sont **les 4 touches fléchées** avec le clavier Français AZERTY et les touches **Q S** et **Z D** (et pour clavier Anglais QWERTY **A S** et **W D)**.

Maintenant, prenons plus de contrôle pour faire des choses plus précises. Voici un nouvel exemple :

Actor "magicien", Image$="magic.png", Control$="ArrowRight: offsetX = 4; ArrowLeft: offsetX = -4"

Run le programme (bouton RUN). À l'écran, notre actor est statique. En appuyant sur les touches fléchées droite et gauche du clavier, notre actor se déplace.

Nous voyons que nous avons défini, deux touches dans le **Control$** : **ArrowRight** et **ArrowLeft**. Ce sont des propriétés, des noms uniques, associés à ces touches (droite et gauche).

Étudions notre programme :

L'actor est positionné aux coordonnées 0.0 qui sont les valeurs par défaut.

Nous définissons les valeurs (ou propriétés - après j'arrête de le répéter promis) du paramètre **Control$** avec une chaîne de texte. Rappelez-vous, une chaîne de texte, que l'on nomme également chaîne de caractères, commence et se termine par **"** dans AOZ. Dans cette chaîne nous assignons les valeurs des deux touches (**ArrowRight** et **ArrowLeft** ou flèche droite et flèche gauche). Et pour chacune, nous avons établi les valeurs à utiliser pour le **offsetX** qui indique le nombre de pixels du déplacement horizontal. Une valeur négative a été définie pour la touche gauche afin de déplacer l'actor vers la gauche de l'écran.
Rappel : en haut à gauche de l'écran c'est la position X=0, Y=0.

Pour résumer, vous pouvez assigner des propriétés à une touche comme suit :
"Code de la touche : propriété1=xxxx; propriété2=yyyy;,..."
Ex: **"ArrowRight: offsetX = 4 ; ArrowLeft: offsetX = -4"**

Chaque nom de control (ex Keyboard, ArrowRight) est séparé par un point-virgule et leurs propriétés sont séparés par une virgule. Le format est comme suit :
<nom du control (Ex Keyboard)**> : propriété1=valeur1, propriété2=valeur2, propriété3=valeur3 ; <nom du control> : propriété1=valeur1, propriété2=valeur2, ...**

À vous de jouer 😊 :

- Changez la vitesse du mouvement de votre actor.
- Ajoutez les propriétés ArrowUp: offsetY = -4 ; ArrowDown: offsetY = 4 pour déplacer l'actor dans les 4 directions.
- Inverser le mouvement : touche gauche pour se déplacer à droite, et touche droite pour se déplacer à gauche.

On peut toujours rendre les choses plus compliquées...

Eh bien, pour continuer notre série « faisons compliqué quand on peut faire simple », nous allons maintenant contrôler notre actor sans utiliser le paramètre **Control$**. C'est possible, voici comment :

```
PX=0 : PY=0
Actor "magician", Image$="magic.png"
Do
   If Key State( 37 ) = True Then PX = PX - 4
   If Key State( 39 ) = True Then PX = PX + 4
   If Key State( 38 ) = True Then PY = PY - 4
   If Key State( 40 ) = True Then PY = PY + 4
   Actor "magician", X = PX, Y = PY
   Wait Vbl
Loop
```

Exécuter le programme (RUN).

En utilisant les touches fléchées du clavier, vous découvrez que vous contrôlez très bien le mouvement de l'actor 😊.

Note : Nous verrons plus tard les N° des touches du clavier, utilisés ici avec KeyState().
Analysons notre programme :

- **Lignes 4 à 7** : Ces 4 lignes permettent de tester chacune des touches fléchées du clavier.
  La fonction **Key State (valeur de la touche)** devient vraie quand la touche correspondante est pressée sur le clavier.

  Si (**If**) c'est le cas, le code qui suit le **Then** sera exécuté.
  Vous pouvez bien sûr appuyer sur deux touches simultanément. Gauche et Haut, par exemple, pour un déplacement en diagonale.

  Remarque : à l'occasion, essayez le petit programme "ScanCode Tester" qui se trouve dans le dossier AOZ Studio "Accessories", ou consultez également le code des touches du clavier sur le site https://keycode.info/.

- **Ligne 8 :** met à jour notre actor en attribuant les nouvelles valeurs des variables PX et PY aux paramètres X et Y. Comme d'habitude, nous ne mettons à jour que les paramètres qui changent. Rappelez-vous qu'il n'est pas utile de modifier les paramètres qui ne changent pas, comme par exemple ici l'image "magic.png".

Vous avez vu comment contrôler notre actor au clavier en utilisant **Control$** ou **Key State**, et ce n'est que le début. Vous allez pouvoir faire des merveilles avec vos actors grâce à toutes sortes de comportements. Le moment venu, reportez-vous au chapitre **ALLER PLUS LOIN AVEC ACTOR**.

DÉPLACER UN ACTOR AVEC LE JOYSTICK

Nous avons vu combien il était simple d'utiliser les touches par défaut ou d'assigner des touches du clavier pour manipuler notre actor. Mais comment faire avec un Joystick ? Est-ce plus compliqué ?

Comme toujours avec AOZ, vous pouvez faire simple avec les paramètres par défaut et faire ce que vous voulez en modifiant les paramètres et les propriétés.

La version simple du mouvement avec le Joystick est :

Actor "magicien", Image$="magic.png", Control$="Joystick"

Connectez un joystick ou une manette de jeu à votre ordinateur, exécutez le programme et déplacez votre actor. Grand ou pas ?

Et la version la plus contrôlée :

Actor "magicien", Image$="magic.png", Control$= "JoyRight0: offsetX = 10; JoyLeft0: offsetX = -10"

Ici, le contrôle de l'actor est défini pour que la vitesse sur l'axe des X soit +10 vers la droite, et -10 vers la gauche.

Control$="JoystickN" (notez le N dans la syntaxe). Nous numérotons les joysticks car vous pouvez en avoir plusieurs connectés à votre PC ou appareil.

Je résume les propriétés :

- **Joystick (**ou **Joystick0)** : pour toutes les directions du 1er joystick connecté
- **JoystickN** : pour le second (**Joystick1)**, troisième, ... joystick connecté
- **JoyLeftN :** vers la gauche.
- **JoyRightN :** vers droite.
- **JoyUpN :** vers le haut.
- **JoyDownN :** vers le bas.
  (Pourquoi je dis cela, c'est tellement évident).
- **JoyButtonN (**N° du bouton**) :** ou N est le numéro du bouton d'action du joystick. C'est nécessaire car il y a de nombreux boutons sur les nombreux Joysticks, Gamepads, Controleurs des simulateurs de vol, de sorte que chacun a besoin d'un numéro.

Remarque : **Joystick, Joystick0, JoyLeft0** utilisent le premier joystick connecté… Oui, la numérotation commence à 0 et non à 1, c'est comme ça, une vieille habitude de l'informatique.

À votre tour 😊 :

- Changez la vitesse de 10 à 6
- Ajoutez les propriétés JoyUp0: offsetY = -6; JoyDown0: offsetY = 6 pour déplacer l'actor dans les 4 directions
- Doubler la vitesse du mouvement de votre actor
- Inversez les directions

Vous verrez plus loin dans ce guide, dans le chapitre **ALLER PLUS LOIN AVEC ACTOR** encore plus de possibilités pour manipuler vos actors, : comment gérer les collisions entre plusieurs actors, comment faire des animations etc.

4. INTRODUCTION AUX ANIMATIONS

Si vous êtes arrivé jusqu'ici vous avez probablement déjà commencé à jouer avec les paramètres de l'instruction **Actor**. Peut-être avez-vous même pensé à quelques idées de jeu, ou d'applications. Nous espérons que vous commencez à sentir la puissance d'AOZ Studio entre vos mains et les possibilités créatives fantastiques que vous avez dorénavant.

LOOKAT$

Nous allons parler des Hotspots et d'une nouvelle instruction amusante : **LookAt$**

Voici un tout nouveau programme afin que vous puissiez comprendre ce que **LookAt$** fait dans la vie.

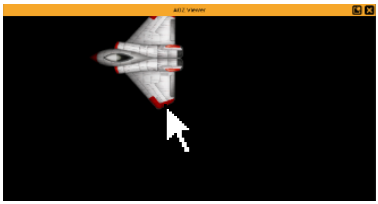
```
Actor "ship", X=Screen Width/2, Y=Screen Height/2,
Image$="ship.png", LookAt$="Mouse"
```

Exécutez le programme : Notre actor est affiché à l'écran comme vous pouviez vous y attendre. Maintenant déplacez votre souris, et remarquez comment votre actor semble regarder dans sa direction !

Etudions ce programme :

Ligne 1: Nous affichons l'actor au centre de l'écran qui est la largeur et la hauteur que nous donne AOZ (selon votre écran) divisée par 2. Le paramètre **LookAt$** a la propriété **"Mouse"**. Cela dit à l'actor de s'orienter de manière à toujours faire face au pointeur de la souris.

Screen Width et **Screen Height** sont 2 fonctions qui donnent la taille de l'écran en pixels. Par défaut, cette taille est de 1920x 1080 (full HD, soit environ 2 millions de pixels affichés), mais vous pouvez modifier la résolution de l'écran.



Mais, mais…, notre actor n'est pas du tout centré, et il semble qu'il tourne autour de quelque chose. C'est bizarre.

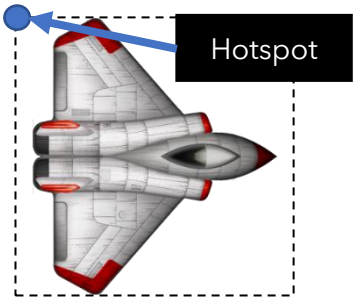
LES HOTSPOTS

Pour faire des animations, vous devez comprendre le principe du hotspot (et des feuilles Sprites qui seront vues plus tard dans le chapitre Animation).

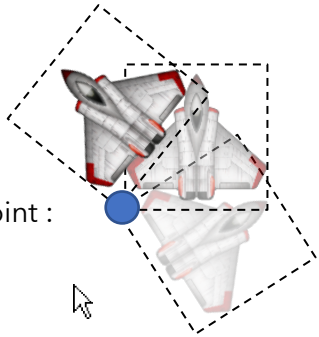
Le hotspot ou « point d'ancrage » est un point invisible, comme le centre de gravité de l'actor. Le hotspot sert à :

- Placer l'actor précisément à l'écran en X et Y
- Fixer le centre de rotation de l'acteur
- Centrez un zoom sur l'actor
-

Par défaut, le hotspot est placé en haut et à gauche de l'image de chaque actor.



Comme il est situé là, la rotation de notre actor est autour de ce point :



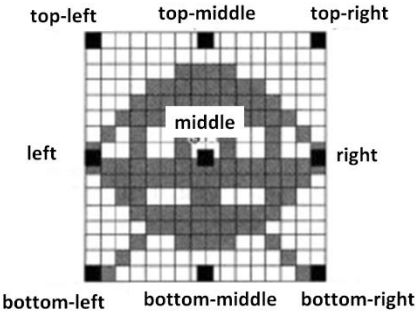
Donc, pour obtenir le résultat de rotation attendu dans notre programme, nous avons besoin de déplacer le hotspot de notre actor au centre de celui-ci. Rien de plus simple avec AOZ, nous allons utiliser une nouvelle instruction qui nous permettra de déplacer le hotspot au bon endroit. Et cette instruction, vous n'auriez jamais deviné, est appelé **Hotspot$**.

Modifions la ligne de notre code comme ceci :

```
Actor "ship", X=Screen Width/2, Y=Screen Height/2,
Image$="ship.png", LookAt$="Mouse", Hotspot$="middle"
```

Exécutez le programme et voyez que le problème de rotation est résolu. Votre actor est parfaitement centré, et s'oriente autour de son centre. Ce paramètre **Hotspot$** défini son emplacement dans l'image. Ici, en lui donnant la valeur **middle** (milieu) cela indique qu'il se positionne au centre de l'image.

Le diagramme suivant détermine la valeur du hotspot :



Le point chaud est réglé à partir du coin supérieur-gauche de l'image. Nous voyons que le point central de l'image est défini par le la valeur **"middle"**.

Lorsque l'actor est ainsi affiché le décalage du hotspot est ajouté par AOZ automatiquement aux coordonnées X, Y ; ce qui est bien agréable, parce que l'on à rien à faire. J'aime ça.

Remarque : Il est parfaitement légal de positionner le hotspot en dehors de l'écran actuel. Cela peut être utilisé par exemple pour un jeu ou l'actor apparaît ou disparaît de l'écran.

N'oubliez pas que le Hotspot est un point d'ancrage (ou point chaud), c'est-à-dire que les positions X et Y sont les positions du hotspot de votre actor. Par exemple si vous modifiez votre ligne de code en mettant X=0 et Y=0 :

Actor "ship", X=0, Y=0, Image$="ship.png", LookAt$="Mouse", Hotspot$="middle"

Vous verrez que votre actor sera partiellement caché par le bord de l'écran parce que le point affiché à 0,0 est en fait le centre de l'image et pas son coin supérieur gauche.
Pour retrouver la position du hotspot d'origine, il faut replacer le Hotspot en haut à gauche, soit la valeur **"top-left"**.

Le Hotspot n'est peut-être pas le sujet principal de ce paragraphe, mais il est important de le comprendre.

Alors maintenant, revenons à ce paramètre **LookAt$**.

LookAt$ indique à l'actor qu'il doit regarder dans une direction spécifique. Nous avons demandé à l'actor de suivre le pointeur de la souris en définissant la propriété = **"mouse"**.

Regardons les différentes propriétés de **LookAt$:**

- **Mouse :** Comme nous venons de le voir, l'actor suit le pointeur de souris.
- **N:** L'actor regarde sans fin l'autre actor (c'est bien pour les actors amoureux). N est le numéro ou le nom de l'acteur à suivre.
- **X,Y:** L'actor regarde un point fixe sur l'écran. Si l'actor bouge, il suivra toujours ce point. X et Y sont les coordonnées du point à suivre.

Exemples:

- - LookAt$="Mouse"
- - LookAt$= "MonAmoureux"
- - LookAt$="100,350"

AUTO$ (AUTO$)

Allons encore plus loin et changeons votre code comme ceci :

Actor "ship", X=Screen Width/2, Y=Screen Height/2, Image$="ship.png", LookAt$="Mouse", Hotspot$="middle", Auto$="forward=4"

Exécutez le programme. Votre actor continue à regarder le bout de votre souris, mais cette fois en le poursuivant !

Auto$ est un autre grand paramètre d'actor, il détermine ce qui doit être fait automatiquement selon ses propriétés : ici **forward=4** déplacera automatiquement l'actor (vers le pointeur de la souris).

DÉFILEMENT AVEC ACTOR

Tapez ce petit programme de 3 lignes s'il vous plaît:

Actor 2, X=0, Y=0, EndX= -1920, Duration=10000, LoopMove=True, Image$="bg.png"

Actor 1, X=0, Y=880, EndX= -1920, Duration=7000, LoopMove=True, Image$="ground.png"

Actor "magic", X=Screen Width/2, Y=Screen Height/2, Image$="magic.png", LookAt$="Mouse", Hotspot$="middle", Auto$="forward=8"

Note : vous pouvez copier/coller ces lignes directement dans AOZ Studio pour éviter les fautes de frappe. Il y a 3 lignes dans ce programme qui commencent chacune par Actor.

Allez on fait un petit RUN, et…

Étonnant ce que vous arrivez à faire avec 3 lignes de code !
Vous êtes un magicien, je vous l'avais pourtant dit, vous ne m'aviez pas cru ?

Nous avons ajouté un nouveau paramètre : **LoopMove**, lorsque la valeur de celui-ci est "**true**" ce qui veut dire "vraie", dans cet exemple l'image de l'actor défilera en boucle, il répètera le mouvement de X,Y à EndX, EndY.

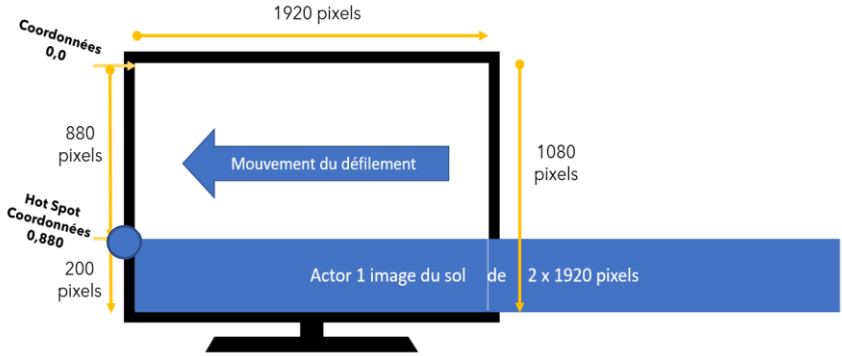
Nous faisons défiler horizontalement 2 actors, nommés 1 et 2, avec chacun son image :
Notez que l'image de la forêt (Actor 2) s'affiche depuis le coin en haut à gauche (X=0, Y=0), alors que le sol (Actor 1) est placé à la position 0,880 (X=0, Y=880) pour commencer. Il finit à -1920 (EndX = **-**1920) et son mouvement dure 700ms, plus rapide que la forêt.

POURQUOI -1920 ?
Pour comprendre vous devez savoir que :

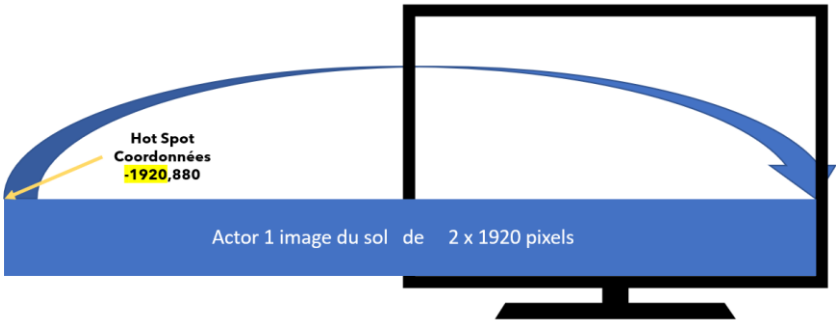
- La résolution horizontale de l'écran est de 1920 pixels. 1920*1080 sont les valeurs par défaut dans AOZ Studio, mais vous pouvez les changer.
- Dans cet exemple les deux images ont une taille de 3840 pixels soit 2 *1920 de long.

Si **LoopMove** est **True**, cela dit à l'instruction **Actor** que lorsque l'animation demandée de l'image sera terminée il faudra recommencer.

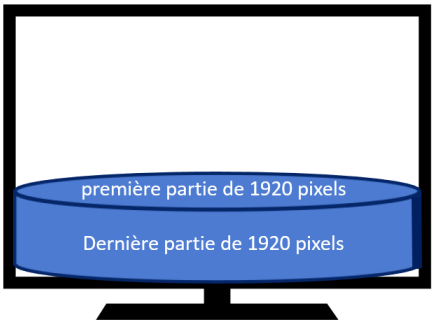
Tu te souviens du hotspot ? J'espère bien, c'était le chapitre précédent. Le hotspot de l'image du sol est en haut à gauche de l'image, donc pour le déplacer vers la gauche l'EndX doit être négatif (ici -1920).



Lorsque la moitié de l'image (1920 pixels) a disparu à gauche de l'écran, l'instruction **Actor** va commencer à boucler :



Et la boucle continue, comme une roue qui tourne.



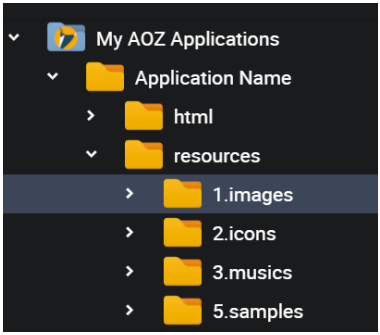
C'est assez pour le moment, il y aura beaucoup d'autres choses magiques à découvrir plus tard. À ce stade je vous propose de continuer à explorer AOZ Studio.

OÙ SONT MES IMAGES ?

À ce stade, vous vous poser sans doute la question :
Mais où sont enregistrées toutes les images que j'utilise dans ces exemples ? Comment puis-je ajouter une image ?

Voici les réponses.

Chaque programme que vous créez a son propre dossier « resources » dans lequel vous trouverez le sous-dossier : « 1.images ». C'est là que vous déposerez vos nouvelles images.

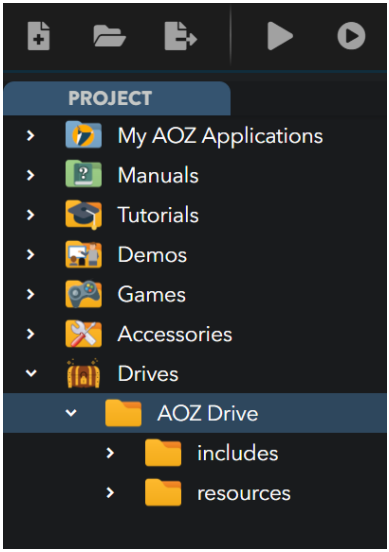


Une fois qu'une image, un son,... se trouve là, votre programme y aura un accès direct et pourra l'utiliser juste par son nom.

- Vous pouvez déplacer et copier des sons, de la musique, des images de votre ordinateur directement dans les dossiers de ressources de votre application affichés dans AOZ Studio.
  Vous pouvez trouver ces dossiers sur votre disque dur (par défaut) :
  Documents/Mes applications AOZ/**votre programme**/resources/

- Vous pouvez aussi copier un fichier d'un autre répertoire d'AOZ Studio en le faisant glisser dans le dossier correspondant alors que la touche Ctrl est pressée.

OK, j'ai besoin de vous dire un secret. Les images, les sons que nous utilisons dans ce guide de l'utilisateur proviennent d'un dossier secret… Quoi ? Oui je sais, nous avons créé un dossier secret de ressources partagées, le «AOZ Drive».



Il comprend déjà les images génériques que nous utilisons dans ce guide de l'utilisateur (au passage merci Romain d'avoir dessiné nos personnages, Lucie, Magic et tous les autres).

Grâce à ce système de ressources partagées d'AOZ Studio lorsque vous créez de nouvelles applications vous n'avez même pas besoin de copier les images que vous utilisez souvent (qui se trouvent dans le dossier partagé global AOZ Drive) dans vos dossiers Applications/ressources, AOZ Studio le fait pour vous.

Dis autrement, si vous enregistrez une image dans ce dossier global AOZ Drive, vous pourrez utiliser cette image (ou son) dans toutes vos applications sans rien faire. Pratique pour les images que vous utilisez fréquemment comme votre logo, votre photo,...

Ce dossier secret, je peux maintenant révéler où il se trouve :
"C: /AOZ Studio/AOZ Drive/ressources/
(si vous avez installé AOZ Studio dans C:/)

Comment cela fonctionne ?

Lorsque vous exécutez votre application (RUN), AOZ Studio cherche et copie les ressources dont vous avez besoin (celles utilisées dans votre programme) de l'AOZ Drive dans le dossier ressources de votre application. Et quand elles sont dans le dossier ressources de votre application votre programme peut les utiliser directement (et même sans extension ex "lucie" au lieu de "lucie.png").

AOZ utilise plusieurs formats d'images, .PNG, .GIF, .SVG sont ceux que nous conseillons d'utiliser car ils gèrent la transparence, bien utile au-dessus des arrière-plans. Mais vous pouvez aussi utiliser .JPEG, .BMP, .iff ou .ilmb

Remarque : une autre méthode existe pour utiliser des images dans un programme avec l'instruction Load Asset. Cette instruction charge un fichier (image, son, vidéo) directement du répertoire Resources/Assets de votre application ou à partir d'un chemin donné (voir le chapitre : Dossier par défaut).

OK faisons une pause avec Actor et la manipulation des images, cette instruction **Actor** est superpuissante et il faudra du temps pour la maîtriser.
Je vous propose maintenant de vous familiariser avec la programmation plus classique et quelques instructions clés pour commencer, puis nous allons ensemble écrire un jeu !

5. PLUS DE MAGIE

Un petit récap avant de continuer ?

Vous savez maintenant utiliser :

- **Actor** pour :
  - afficher des images **Image$**
  - les positionner précisément avec **Hotspot$**
  - déplacer des images en indiquant leurs coordonnées avec : **X, Y, EndX, EndY**
  - les animer automatiquement avec **Auto$**
  - choisir comment les piloter avec **Control$**
    au joystick avec : **JoyStick**

  et les commandes associées :
  - déclencher leur déplacement avec **ActionMove$**
  - les orienter automatiquement avec **LookAt$**
  - les faires défiler en boucle avec **LoopMove**

- **Print** pour afficher un message à l'écran.

… et vous avez commencé à utiliser :

- **If… Then** pour définir des conditions et agir en conséquence
- **Do… Loop** pour répéter des actions.
- **InKey$** pour récupérer une touche pressée par l'utilisateur

Si vous vous demandez les différences entre les fonctions :

• **Inkey $** : renvoie la touche sous forme de chaîne ou de chaîne vide si la mémoire du clavier est vide.

• **Key State (n)** : vérifie si la touche avec l'ID **n** est actuellement enfoncée et renvoie true si c'est le cas.

Pas mal pour un début ! On continue dans ce chapitre avec :

- **Input** pour saisir du texte au clavier de l'utilisateur
- **Goto** pour aller à une partie de votre programme
- **If... Then... Else** pour compléter ce qui précède, bien maîtriser les conditions et déclencher des actions en conséquence.

C'est parti !

Input pour saisir au clavier

Effacez tout votre code précédent et tapez :

```
Input "Quel est votre nom" ; NOM$
Print "Bonjour" ; NOM$
Input "Quel âge avez-vous ?" ; age
jours = age*365
Print "Wow ! En jours c'est " ; jours
Bell
Print "Vous êtes assez vieux pour être un magicien !"
Print "Au revoir" ; NOM$
Bell 2
```

Input est une excellente instruction pour demander quelque chose. Ex : **Input NOM$**.

Ici, nous combinons **Input** avec un affichage :

Input "Quel est votre nom" ; NOM$
Affichera d'abord « Quel est votre nom » puis attendra la saisie de la réponse, et ce que vous répondrez en tapant sur votre clavier sera stocké dans la variable **NOM$.**

Input "Quel âge avez-vous ?" ; age
La valeur que vous saisirez sera stockée dans la variable « age ».

jours = age*365
Cette ligne calculera le nombre de jours correspondant et le stockera dans la variable « jours ».

Vous avez peut-être remarqué que certains noms de variables se terminent par un $, comme **NOM$**, mais d'autres n'ont pas ce $ comme **age**.

Cette convention permet à AOZ de traiter spécifiquement les variables numériques et les chaînes de caractères : on appelle ça **le typage des variables**, et vous devrez respecter cette convention dans vos programmes pour éviter un message bloquant en rouge "Erreur : incompatibilité de type".

Goto pour sauter à une partie de votre code

Votre programme est constitué d'une liste d'instructions, qui s'exécutent généralement l'une après l'autre. Mais vous aurez parfois besoin d'aller directement à une portion particulière de votre code.

Vous avez besoin pour cela de définir une **étiquette** pour repérer l'endroit du code où vous voulez aller.

Dans AOZ, vous définissez une étiquette par un nom suivi de deux points. Exemple : « LAGYM : ».
Vos noms peuvent contenir n'importe quelles lettre (majuscule ou minuscule) ou nombre, ainsi qu'un « underscore » pour en améliorer la lisibilité pour pouvoir écrire « LA_GYM : »

Une fois une étiquette définie, vous pouvez demander à votre programme d'y accéder grâce à l'instruction **Goto**.

Quand on mélange ça avec des conditions, ça devient vraiment… magique !

If… Then… Else pour définir des conditions

Si (**If**) quelque chose est vrai alors (**Then**) l'ordinateur fera une action, sinon (**Else**) il fera quelque chose d'autre.

Copiez / collez puis exécutez ce petit programme et regardez l'ordinateur décider quelle heure il est :

```
NUIT=1
MAINTENANT=1
Print "Quelle heure est-il maintenant..."
Wait 2
If MAINTENANT = NUIT Then Goto LIT
If MAINTENANT <> NUIT Then Goto FOOT

LIT:
Print "Je pense qu'il faut se coucher"
End

FOOT:
Print "Où est la balle ?"
End
```

Maintenant, changez la valeur de NUIT par exemple NUIT=0, et exécuter ce programme à nouveau.

Bon, ce n'est qu'un exemple : l'intérêt de Goto deviendra évident lorsque vous aurez une même portion de code utilisée à plusieurs endroits de votre programme : l'étiquette vous permettra de ne pas répéter plusieurs fois les mêmes instructions, et facilitera la maintenance de votre programme.

Si vous êtes sûr d'avoir compris comment votre ordinateur a pris sa décision *Then Goto* le paragraphe suivant, *Else* essayez à nouveau !

En parlant de **Else** (sinon), modifiez le programme comme suit, puis changez les valeurs de NUIT.

```
NUIT=0
MAINTENANT=1
Print "Quelle heure est-il maintenant ?"
Wait 3
If MAINTENANT = NUIT Then Goto LIT Else Goto FOOT

LIT:
Print "Je pense qu'il faut se coucher"
End

FOOT:
Print "Où est la balle ?"
End
```

Et voilà : « **If MAINTENANT = NUIT Then Goto LIT Else Goto FOOT** », ça se lit bien, se comprend aisément, et s'écrit en une ligne.

Votre premier jeu

Que diriez-vous de créer encore plus de magie en écrivant un jeu simple pour tester ce que vous avez appris jusqu'à présent ? C'est un petit jeu de logique. Mais avant de jouer vous devez connaitre ces symboles mathématiques :

=	signifie « est égal à »
<>	signifie « est différent de »
>	signifie « est plus grand que »
<	signifie « est inférieur à »

Encore une chose : on a jusqu'à présent écrit des conditions simples, qui s'expriment en une ligne, comme **If MAINTENANT = NUIT Then Goto LIT Else Goto FOOT .**

Mais vous aurez souvent besoin d'avoir plusieurs lignes d'instruction dans une condition… et vous aurez besoin de l'instruction **Endif** pour marquer la fin de votre bloc conditionnel.

Ça donne des choses comme ça :

```
If [condition] then
        instruction 1
        instruction 2
        …
Endif
```

ou

```
If [condition] then
        instruction 1
        instruction 2
        …
else
        instruction 3
        instruction 4
        …
Endif
```

OK, lorsque vous êtes prêt tapez le code suivant de ce jeu et aller trouver une victime pour y jouer avec vous.

Il s'agit d'un jeu de devinette. Vous connaissez toutes les instructions utilisées, sauf **Cls** qui vous permet d'effacer l'écran.

```
Print "Jouons au chiffre secret"
Print "Demandez à votre victime de fermer les yeux"
Wait 2
Input "Tapez un numéro secret entre 1 et 10 "; A
Cls

Print "Demandez à votre victime de regarder"
Wait 2


SECRET:
Input "Victime, trouvez le numéro secret " ; B
If B=A
        Print "OUI. FÉLICITATIONS !"
Else
        If B<A
                Print "Faux, c'est plus" : Goto SECRET
        Else
                Print "Et non c'est moins" : Goto SECRET
        End If
End If
```

Vous voyez les deux points : cela sépare deux instructions et évite d'aller à la ligne, vous auriez pu taper avec le même résultat :

```
If B<A
    Print "Faux, c'est plus"
    Goto SECRET
Else
    Print "Et non c'est moins"
    Goto SECRET
End If
```

Suivez bien les étapes du programme, remarquez notamment comment l'étiquette SECRET: permet de signaler l'endroit du code où vous voulez aller tant que le chiffre secret n'a pas été trouvé.

Notez les imbrications logiques des deux conditions.

Bon, ce qui précède fonctionne bien, mais vous pouvez obtenir le même résultat en bien moins de lignes, en utilisant des conditions sur une ligne, qui ne nécessitent plus de Endif.

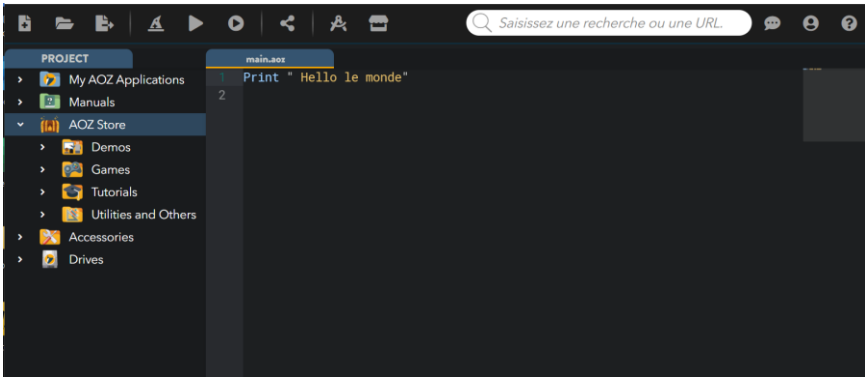
Remplacez la partie SECRET: par ce qui suit :

```
SECRET:
Input "Victime, trouvez le numéro secret " ; B
If B=A Then Print "OUI. FÉLICITATIONS !" : End
If B<A Then Print "C'est plus" Else Print "C'est moins"
Goto SECRET
```

Plus simple non ? On dit "optimisé". **If** vous ne comprenez pas tout **Then** "J'ai échoué" **Else** "Vous êtes brillant."

6. LE MODE EDIT

Trouver votre chemin dans AOZ Studio a été rendu aussi simple que possible.



Les dossiers des projets (Project) sont sur la gauche,
La zone principale -le grand écran de travail- est sur la droite, et
Les boutons d'actions sont sur le menu en haut.

Depuis le début de ce guide utilisateur, vous vous demandez probablement ce que vous pouvez faire avec tous ces boutons en haut de l'écran.

Les boutons du menu

Chacun d'eux fournit une action en un clic :

 Créer une nouvelle application AOZ

 Charger une application AOZ qui existe déjà

 Enregistrer une application AOZ et l'exporter pour la partager avec d'autres

 Passer en mode AOZ Direct pour tester des bouts de programmes

 Exécuter cette application dans un navigateur Web

 Exécuter cette application dans la visionneuse AOZ

 Publier cette application pour que le monde en profite

 Composer l'interface des pages avec le DESIGNER

 L'AOZ Store, la bibliothèque partagée des programmes

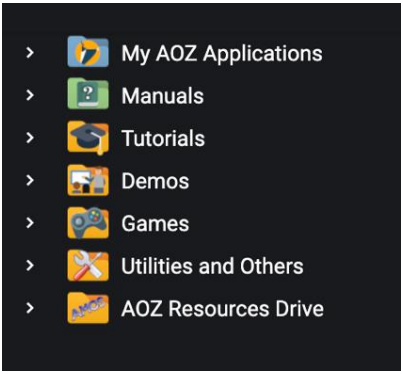
 Lire vos messages AOZ

 Votre compte et paramètres utilisateur

 Consulter la documentation

Les dossiers Projets

Les dossiers de projets sont tous soigneusement empilés sur le côté gauche de l'écran. A l'intérieur de chaque dossier on trouve toute une série de sous-dossiers, et à l'intérieur de chaque sous-dossier une foule de trucs magiques.

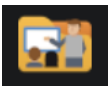


Vous pouvez déplacer, supprimer, renommer et copier. Note si cela ne fonctionne pas parfaitement, selon votre version, nous savons…
Regardons chaque dossier à tour de rôle.

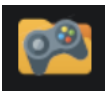
Toutes vos applications sont conservées dans ce dossier par défaut, avec les noms que vous leur avez donnés. Chaque dossier d'application peut contenir des sous dossiers de ressources telles que des images, des icônes, de la musique et des effets sonores.

Le dossier Manuels de référence est l'endroit où vous trouverez ce guide utilisateur.
Les anciens y trouverons le manuel AMOS Pro de l'ordinateur AMIGA ; AOZ est compatible avec son vénérable grand-père AMOS, mais avec beaucoup plus d'instructions.

Les tutoriels sont conservés dans ce dossier. Ils comprennent toute une série de leçons sur AOZ Studio.



Le dossier Démos contient des tas d'exemples, des effets spéciaux,… Explorez ces démos, examiner leur code et adaptez-les comme vous le souhaitez.



Jeux, c'est l'endroit où vous trouverez des jeux complets pour profiter, adapter…

Accessories/Utilities est le dossier pour les outils et ressources externes qui peuvent vous être utiles comme une visionneuse de polices, de joystick, de clavier etc..

MANUEL INTERACTIF ET AIDES

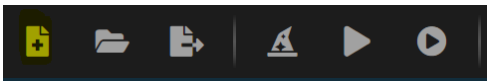
Le manuel interactif est appelé avec le bouton **(i)** en haut à droite de l'écran et également lorsque vous pressez la touche **F6** dans votre code (cliquez sur une instruction puis F5), c'est très utile.

Essayez également de cliquer sur une instruction puis **F5**

7. CRÉER UN PROJET

Préparation

Créons un nouveau projet. Cliquez sur la première icône de la ligne en haut de votre écran, qui dit **« Nouveau projet AOZ».**



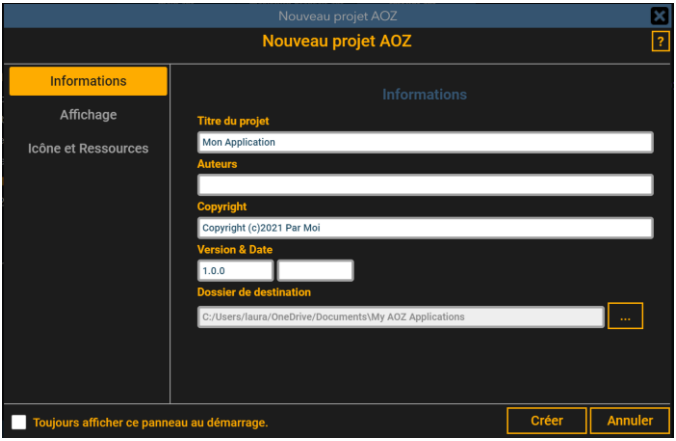
Une boîte de sélection apparaît avec un choix de modèles de projet (ou "templates") :

Il existe des modèles prêts à l'emploi
- "multiplateforme" : pour les écrans d'ordinateur, de smartphones, les sites Web,
- "Amiga" pour être compatible avec le format historique de cet ordinateur.
- "Personnalisable" pour configurer le projet vous-même.

Notez tout en bas à gauche la case à cocher pour ouvrir ce panneau à chaque fois.

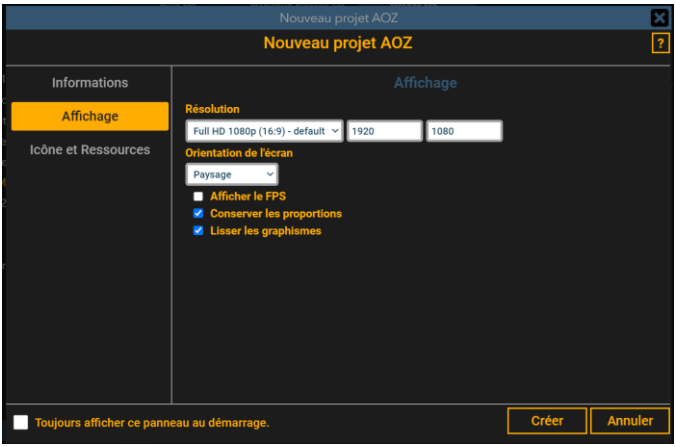
Supposons que vous souhaitez que votre nouvelle application AOZ s'exécute en tant que projet pour smartphones, sélectionnez le modèle Projet AOZ multiplateforme.

Vous pouvez maintenant remplir le formulaire Informations :



N'oubliez pas le titre du projet et de sélectionner le dossier de destination dans lequel vous souhaitez conserver votre projet (si celui par défaut ne vous convient pas).

Puis cliquez sur Affichage :



AOZ Studio vous aidera autant que possible à ce que votre application s'ajuste à l'appareil que vous souhaitez utiliser. Pour une application pour smartphone, vous pouvez choisir l'orientation de la page : portrait (vertical), paysage, ou les deux (auto détection).

- le FPS (pour Frame per Second) vous donnera en haut à gauche une indication de la vitesse d'affichage)
- Conserver les proportions : garde intact l'aspect de vos images, sans étirement ni déformation, quelque soit la taille de la fenêtre utilisée.
-Lisser les Graphismes : applique un lissage autour de vos images pour atténuer l'aspect pixelisé.

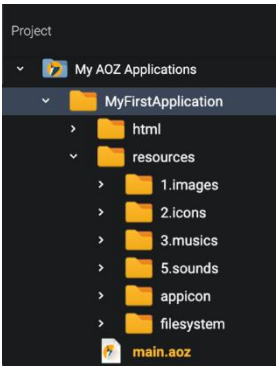
Puis cliquez sur Icône et Ressources pour choisir l'icône graphique de votre application si vous le souhaitez.

Concernant "Utiliser le dossier Assets par défaut" il vaut mieux laisser cette case cochée pour commencer. Si elle n'est pas cochée, votre programme pourra charger des assets, en Français des ressources (images, sons,…), depuis n'importe quel dossier de votre ordinateur en fournissant le chemin complet avec la commande Load Asset que nous verrons plus tard.

Puis cliquez sur le bouton Créer pour ouvrir le projet, vous êtes prêt à programmer.

Vos dossiers d'applications

Lorsque vous créez une nouvelle application, elle est enregistrée dans son propre dossier, là où tout les éléments nécessaires à son fonctionnement se trouvent également.



Par défaut, ce dossier ce trouve dans le répertoire "My AOZ Applications" sur votre ordinateur, avec donc les dossiers contenant toutes les images, icônes, l'audio, la vidéo, les fichiers de données.

Comme toujours, l'application est lancée en cliquant sur le programme : **main.aoz** ("main" peut être changé par le nom que vous avez donné à votre programme)

Bien, à ce stade de votre progression, nous allons commencer à vous donner le manche à balai et vous laisser voler de vos propres ailes. Préalablement nous avons besoin de dire un mot ou deux sur les erreurs de programmation et les bogues, et comment nous avons fait tout notre possible pour vous aider à y faire face. Passons donc au chapitre suivant.

Lucie dit : Salut, à toi de jouer !

8. ERREURS, BOGUES ET AIDE

La plupart des programmeurs se réfèrent aux erreurs et aux pépins dans leurs programmes comme des bogues ou « bugs ». Ces petits diables sont responsables de nous gâcher la magie. Ils peuvent être dus à une erreur de frappe, une instruction qui a été mal utilisée, une demande impossible ou mal formulée qui fait perdre la tête à l'ordinateur. Parfois, cela peut prendre des heures, voire des jours pour repérer où ces bugs se cachent et ce qu'ils font, mais AOZ est là, acharné à la tâche pour les débusquer avec vous.

AOZ vous aidera non seulement à repérer l'erreur, mais aussi essaiera d'expliquer quel est le problème et exactement où il se cache.

Si cela se produit pendant que vous programmez, alors vous pourrez corriger le bug immédiatement. Si cela se produit lorsque vous testez ou exécutez votre programme AOZ vous guidera directement vers toutes les lignes de code en infraction, l'une après l'autre, jusqu'à ce que cela soit parfait.

En dehors des erreurs de frappe simples, certaines des erreurs les plus courantes viennent de la façon dont vous écrivez les variables. Une variable est une quantité ou un nombre dont la valeur peut changer en fonction de ce que l'on y sauve, c'est une case mémoire avec un nom, un peu comme une boite aux lettres. Il est important de se rappeler que les variables dans AOZ, sont sensibles à la casse (les majuscules/minuscules). Une variable avec des lettres minuscules au lieu de majuscules ou l'inverse, sera considérée différemment. Par exemple :

stuff$, STUFF$, sTuFf$ sont pour AOZ 3 variables différentes.

L’aide instantanée

AOZ va essayer de vous aider à chaque étape. Lorsque vous avez commencé à créer votre magie en tapant des lignes de code, vous avez probablement remarqué que votre code est apparu dans différentes couleurs. Mots-clés, commentaires et variables apparaissent tous dans leur propre couleur distincte car c'est plus facile à lire. Si quelque chose n’est pas tout à fait clair alors AOZ va essayer de mettre en évidence le problème en changeant de couleur. De même, lorsque vous tapez les premiers caractères d’une commande AOZ tentera de vous faire gagner du temps et d’éviter les erreurs en vous montrant une liste d'instructions qu’il estime pouvoir utiliser ; déplacer vous dans la liste avec les touches fléchées.
Vous pouvez également **cliquer ou sélectionner l'instruction pour mettre en évidence l’élément qui vous intéresse et presser la touche F5 ou F6** (voir page suivante). AOZ vous en parlera automatiquement et vous aidera à mettre vos instructions dans le bon ordre et le bon format.

Bon sauf à venir chez vous et programmer pour vous, que pouvons-nous faire de plus ? Eh bien, nous pouvons faire beaucoup plus, par exemple, continuer ce guide utilisateur. Alors s’il vous plaît, lisez la suite.

L'aide sur erreur

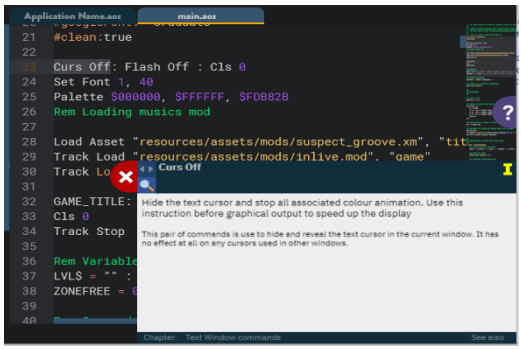
Si vous faites une erreur et il y a des erreurs qui arrêtent votre programme, ces erreurs apparaîtront sur leur propre mur de la honte dans la fenêtre de votre éditeur en bas, généralement avec un message utile montrant le numéro de ligne exacte.

Vous pouvez afficher ces erreurs en cliquant sur ce bouton en haut de l’écran ou en tapant **Ctrl+Shift+c**.

Il y a encore plus d'options de débogage si vous utilisez la visionneuse AOZ intégrée (l'AOZ Viewer) en pressant F2 et le chapeau magique en haut à gauche comme nous le verrons dans le prochain chapitre. Alors n'ayez jamais peur d'expérimenter.

Les aides contextuelles F5 & F6

Comme vous apprenez la langue d'AOZ vous devez savoir quelles instructions sont disponibles, et il y en a tellement (près de 800 et ça augmente tous les mois) que vous avez besoin d'être aidé. C'est pour cela que nous avons fait l'aide et le manuel contextuels.



AIDE (F5) : Si vous cliquez sur une instruction puis appuyez sur la touche de fonction F5, un popup apparaîtra et vous donnera tous les détails sur l'instruction et celles qui y sont liés. Voyez également les liens en bas de cette fenêtre. Pour fermer cliquez sur la croix à gauche.

MANUEL (F6) : Si vous sélectionnez une instruction et appuyez sur la touche de fonction F6, la recherche d'AOZ va faire son travail et tout vous dire sur ce qu'elle sait sur cette instruction. Pour fermer cliquez sur la croix à gauche.

L'Aide en mode direct

Et vous avez le mode direct, le petit chapeau magique, pour vérifier toutes sortes de choses, comme la valeur des variables de votre programme, nous allons le voir avec le prochain chapitre.

Aller plus vite

Nous avons déjà parlé d'optimisation, il y a beaucoup de façons d'accélérer un programme, par exemple vous pouvez éviter d'utiliser des graphismes de grandes tailles si vous n'en avez pas vraiment besoin. De même pour vos fichiers audio et vidéo. Certaines instructions vont plus vite que d'autres, en choisissant l'organisation, l'enchainement et les instructions cela aura un impact qui pourra être très important. L'organisation, la structuration du code on appelle cela l'algorithmie.

Pour les apprentis magiciens, vous aurez parfois le problème inverse avec un programme qui va trop vite ! Si vous avez besoin de temporiser votre programme, vous pouvez dire à votre ordinateur d'attendre aussi longtemps que vous le souhaitez. L'instruction **Wait 1** ralenti d'une seconde, et devinez **Wait 10** va moudre pendant 10 secondes. **Wait 0.5 ....**

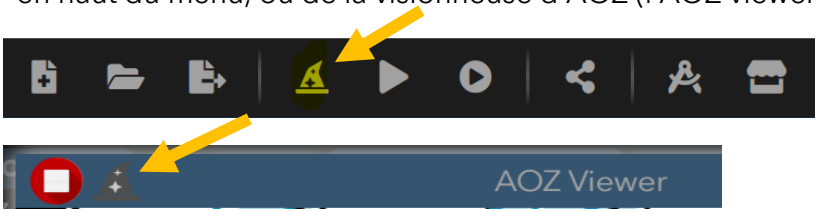
Impatient ! Nous allons revenir dans le royaume magique où nous faisons faire à des objets sur un écran des choses étonnantes, brillantes, effrayantes, déconcertantes, voir carrément folles.

Mais avant il y a une autre aide importante qui a besoin de son propre chapitre, le célèbre : Mode Direct.

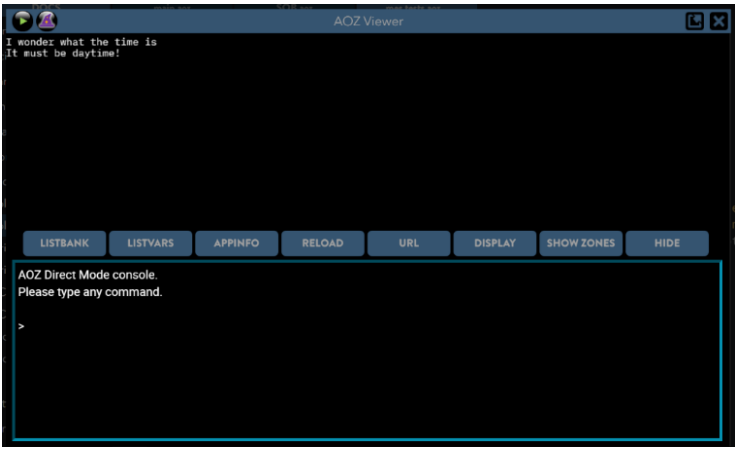
9. LE MODE DIRECT

Jusqu'à présent nous avons utilisé le grand écran et l'édition de notre code dans le mode appelé Mode Edition (ou Edit Mode). AOZ Studio dispose également d'un Mode Direct (Direct Mode), c'est l'endroit pour tester n'importe quoi. Cool.

Pour passer en mode Direct, appuyez sur le chapeau de magicien en haut du menu, ou de la visionneuse d'AOZ (l'AOZ viewer) :



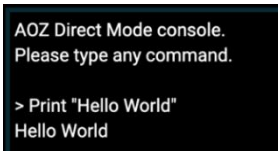
Dans les deux cas vous entrez alors en Mode Direct avec deux groupes de fonctionnalités supplémentaires : Les boutons bleus et dessous la console du Mode Direct.



Essayons, tapez n'importe quel programme et RUN (F2) pour le voir dans la visionneuse AOZ (comme vous l'avez fait jusqu'à présent), puis cliquez sur le chapeau magique pour entrer dans le Mode Direct de la visionneuse.

Vous verrez votre programme, OK très bien, mais aussi les boutons bleus, ils sont assez faciles à comprendre, il suffit d'essayer. Je pourrais en faire 2 pages, mais est-ce utile ?

et au-dessous la "console" d'AOZ.
Tapez vos instructions dans la console, comme ceci : **Print "Hello World"**



Comment ? Oui je pense que les boutons Bleus de Direct Mode sont explicites, et sinon vous êtes les bienvenus pour envoyer une plainte à givemeabreak!@kjh0$£%.com.

10. ALLER PLUS LOIN AVEC ACTOR

S'il vous plaît ne lisez pas ce chapitre avant d'avoir lu le premier chapitre sur l'instruction Actor. J'apprécie votre empressement, mais commençons par le commencement. Pourquoi croyez-vous que nous avons ainsi nommé ce chapitre !

LES DIFFÉRENTS CONTRÔLES POSSIBLES

Note : cette instruction et chapitre sont en cours de développements.

Nous avons vu qu'il était assez facile de contrôler un actor avec le paramètre **Control$**. Et vous pouvez attribuer plusieurs contrôles à un actor (par ex. le clavier, un joystick).

Pour rappel, chaque nom de control (ex Keyboard, ArrowRight) est séparé par un point-virgule et leurs propriétés sont séparés par une virgule. Le format est comme suit :
<nom du control (Ex Keyboard)**> : propriété1=valeur1, propriété2=valeur2, propriété3=valeur3 ; <nom du control> : propriété1=valeur1, propriété2=valeur2, ...**

Voici une description des différents types de contrôles disponibles avec l'instruction **Actor**.

Le Clavier

Exemple: **Control$= "ArrowLeft: angle=4"** Ici, en appuyant sur la flèche gauche, l'actor tournera d'un angle de 4 degrés.

Voici les propriétés de **Control$** :

- **offsetX**: valeur pour le mouvement horizontal de l'acteur.
- **offsetY**: valeur pour le mouvement vertical de l'acteur.
- **angle**: valeur pour la rotation de l'actor

- **hrev**: true/false. Si vrai (true) le retournement horizontal (effet miroir) de l'actor est activé
- **vrev**: true/false. Si vrai (true) le retournement vertical (effet miroir) de l'actor est activé
- **Image:** image à utiliser pour l'actor lorsque cette touche clavier est appuyée
- **anim**: Nom de l'animation à jouer pour l'actor lorsque cette touche est appuyée. Le paramètre **Actorsheet$** de l'actor doit être défini.
- **forward** : valeur numérique pour aller de l'avant. Le mouvement se déroule selon l'angle de rotation de l'actor.
- **backward** : valeur numérique pour reculer. Le mouvement se fait à l'angle opposé de rotation de l'actor.

Exemple:

```
Actor "magicien", X=Screen Width/2, Y=Screen Height/2,
Image$="magic.png", Control$ ="ArrowUp: angle=-8; ArrowDown:
angle=8; ArrowRight: forward=10; ArrowLeft: backward=10"

Actor "lucie", X=Screen Width/2, Y=Screen Height-230,
Image$="lucie.png", Control$="ArrowRight: OffsetX=16; ArrowLeft:
OffsetX=-16"
```

À votre tour 😊 :

- Remplacer la propriété backward par forward, mais toujours en reculant (pensez **-**)
- Essayez les différents paramètres **Control$**

Le Joystick

Comme nous l'avons vu dans l'un des premiers exemples du Chapitre 9 : "Déplacer un actor avec le Joystick", nous pouvons très simplement associer un joystick à un actor pour le contrôler. De nombreux joysticks peuvent être connectés à l'ordinateur, et ils sont numérotés par leur ordre de connexion.

Le premier joystick sera le 0 (zéro), le second sera le 1. Oui, nous savons que cela n'a pas de sens, cela vient de l'origine des ordinateurs personnels, à une époque où les gens étaient bizarres.

Essayez cet exemple de code (joystick branché) :

Actor "magicien", Image$="magic.png", Control$="Joystick"

Et cet exemple :
Actor "magicien", Control$ = "JoyLeft0: angle=-4; JoyRight0: angle=4; JoyUp0: forward=4; JoyDown0: backward=4", Image$="magic.png"

Remarque : si votre joystick est bien connecté à votre PC mais n'est pas reconnu, fermez et relancez AOZ Studio. Si cela ne fonctionne toujours pas vérifier si il est bien reconnu par votre ordinateur (par exemple jouez à un jeu).

Souris

Voyons maintenant comment contrôler un actor, en utilisant la souris (mouse en Anglais). C'est le contrôle le plus facile à définir, car il n'y a qu'une seule propriété :
Control$ = "Mouse"

Essayez le code suivant :
Actor "magicien", Control$="Mouse", image$="magic.png"

Notre actor est placé sous le pointeur de souris. Et si vous déplacez la souris sur l'écran, l'actor se déplace avec elle. Vous pouvez utiliser cette méthode pour remplacer le pointeur de la souris par une image personnalisée dans vos programmes AOZ.

Dans certains jeux, il peut être nécessaire de déplacer l'actor uniquement dans la direction horizontale (comme la raquette dans un jeu de casse briques) ou uniquement dans la direction verticale. Pas de problème, AOZ a tout prévu ! Changeons un peu le code précédent :

Actor "magicien", Control$="Mouse: Honly=true", Image$="magic.png"

Note : si vous avez bien retenu nos explications sur le typage des variables, vous vous demandez peut-être pourquoi ce n'est pas honly**$**, ou que true n'est pas **"**true**"**. Vous vous dites : « héhé ils se sont bien trompés chez AOZ Studio. Ça y est je suis le boss ».
En fait l'ordinateur ne reconnait que des valeurs 1 et 0 ; par convention true veut dire 1 et false veut dire 0. Ce sont des Alias de valeurs numériques, qui ne nécessitent donc ni guillemets, ni $ dans le nom du paramètre. Ça peut se discuter, mais essayez un peu d'utiliser 1 et 0 dans votre code et vous verrez que nos alias simplifient la vie. Alors, c'est qui le boss ?

Exécutez le programme. L'actor suit maintenant le pointeur de souris amicalement mais seulement sur l'axe vertical.

La souris a deux propriétés :

- **Honly** : True (Vrai) / False (Faux). Pour le suivi de la position horizontale.
- **Vonly** : True (Vrai) / False (Faux). Pour le suivi de la position verticale.

Dans l'exemple précédent la verticale est False (car par défaut la propriété est définie comme faux).
C'est intéressant pour un jeu de type Breakout où l'actor ne doit se déplacer que de gauche à droite en bas de l'écran.

Automatique

Ici la tour de contrôle à Major Tom : votre actor est peut-être sur pilote automatique.

En définissant le paramètre **Auto$** toutes les propriétés définies (placées immédiatement après **l'Auto$=**) seront automatiquement utilisées sans avoir besoin d'appuyer sur un bouton, d'agiter le joystick ou de déplacer la souris.

Regardez ce code:
Actor "magicien", X=0, Y=150, Auto$="offsetX=4, offsetY=4", Image$="magic.png"

Exécutez le programme pour voir que votre actor se déplace tout seul vers la droite de l'écran.

Voici l'exemple plus complet : ***Shoot-em-up*** Game Kit, que vous trouverez dans le dossier Tutoriel et qui utilise les animations d'Actor :

```
// On charge les graphismes pour les animations de sprites
Load Asset "sprites/graphics.sprite", "graphics"

// // On paramètre le vaisseau du joueur
Actor "player", X = 160, Y = 260, SpriteSheet$="graphics",
Anim$="idle", LoopAnim = True, LeftLimit = 0, RightLimit = 256
// On y ajoute les paramètres selon l'appareil : Smartphone ou Pc
  If Touch Screen = True  // Pour écran tactile on utilise la souris, que
horizontalement (honly)
 Actor "player", Control$ = "mouse:honly=true"
  Else   // Pour PC on peut utiliser le joystick et le clavier. On crée une
variable C$ pour le contrôle de l'Actor :

C$ = "joystick0:offsetx=2,offsety=2;" // Le 1er Joystick(0) déplace
l'Actor de 2 pixels (offsetx et y) en X et en Y dans toutes les directions

C$ = C$ + "joyleft0:anim=left1,offsetx=-2;"  // Spécifiquement le
Joystick vers la gauche prends l'animation "left1" dans
resources/Assets/sprites/graphics.sprite
```

```
C$ = C$ + "joyright0:anim=right1,offsetx=+2;"  // Idem vers la droite

C$ = C$ + "keyboard:offsetx=2,offsety=2;"  // Le clavier déplace
l'Actor de 2 pixels, par défaut cela n'affecte que les touches fléchées
et les touches a-z/d-e

C$ = C$ + "ArrowLeft:anim=left1,offsetx=-2;"  // Spécifiquement la
touche gauche prends l'animation "left1" dans
resources/Assets/sprites/graphics.sprite

C$ = C$ + "ArrowRight:anim=right1,offsetx=2;"  // Idem vers la droite

C$ = C$ + "none:anim=idle" // Si il n'y a aucune action clavier ou
joystick on prends l'animation "idle" du vaisseau droit (avec le réacteur
qui scintille)

Actor "player", Control$ = C$  // on Affecte tous ces contrôles à l'Actor
// Note: Au lieu de concaténer des C$ il est également possible
d'écrire tout cela en 1 ligne directement après le = de Control$
  End If
```

Cette chose, **Control$**, n'est pas super facile à comprendre, mais une fois que vous aurez joué avec, c'est de la pure magie ! Faites une pause et entrainez-vous un peu.

À votre tour 😊 :

- Comprenez la syntaxe, et notamment l'utilisation des " " avec Control$. Essayez de changer les propriétés et les valeurs.

COLLISIONS

Dans les films, quand un vaisseau spatial en percute un autre, ca fait boom (alors que c'est dans l'espace…). Pour savoir en AOZ quand les choses vont faire boom, vous devez détecter la collision entre les actors. Heureusement il est super facile de détecter les collisions avec AOZ. Vous disposez de deux moyens principaux : Actor Col et On Collision$.

Collisions avec Actor Col

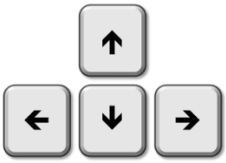
Et une fonction qui renvoie TRUE (ou 1) si les actors entrent en collision. Si aucune collision n'est détectée, la fonction renvoie FALSE (ou 0).

Tapons ce nouveau code pour tester si notre magicien rencontre l'actor(trice) la plus célèbre d'AOZ et en mettant en place la détection de collision pour savoir si cela va faire boom boom.

```
Actor "lucie", 600,150, Image$="lucie.png",Control$="keyboard:
OffsetX=12, offsetY=12"
Actor "magic", 960,540, image$="magic.png"
Do
  A = Actor Col ("lucie", "magic" )
  If A = True Then Boom
  Wait Vbl
Loop
```

Exécutez le programme. Déplacez votre actor avec le clavier et... Boom, c'est l'amour !

Remarquez que cela fonctionne très bien sans indiquer X=, Y= : **Actor "lucie",600,150,** … Je vous l'avais dit, chez AOZ Studio on fait tout pour rendre votre vie agréable.



Le programme est assez facile à comprendre je pense :

Ligne 4 : Ici, comme on utilise Actor avec son nom, et pas avec un numéro, il faut le préciser en mettant des "".

Mais cela fonctionne également avec des numéros par exemple pour tester la collision entre magic et l'actor 101 :
If Actor Col ("magic", 101) = True Then ....

Plus fort on peut tester entre un Actor et une image :
if Actor Col("magic", image$="ice") = True Then Goto DEAD

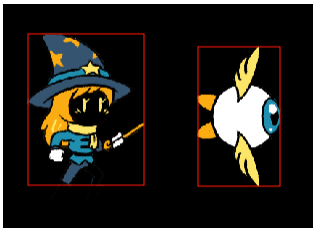
Group$ actors

Encore plus fort, on peut associer plusieurs Actors à un groupe (par exemple le groupe « monstre ») et tester un Actor et un groupe :

```
Actor "monstre1",  Image$ = "monstre1.png", Group$ = "monstre"
Actor "monstre2",  Image$ = "monstre2.png", Group$ = "monstre"
If Actor Col ("magic", Group$="monster" ) = True Then ....
```

Voyez en exemple le tutoriel (mini jeu) MagicBird. Il existe également un tuto vidéo sur la chaîne YouTube d'AOZ Studio.

Avec AOZ les collisions fonctionnent au pixel près ("pixel perfect") par défaut et non lorsque les boites (hit-box) entourant les actors se touchent, c'est donc beaucoup plus précis.



Collision avec l'instruction Actor

Nous venons de voir combien il est facile de tester une collision entre deux actors avec la fonction **Actor Col**.

L'instruction Actor vous permet également de tester une collision, ce, à l'aide du paramètre **OnCollision**. Il s'agit d'un paramètre spécial, utilisé comme un Auditeur, on dit "Listener" en informatique. Le travail d'un Listener est de surveiller un événement (dans ce cas une collision) et d'alerter le programme. Nous reviendrons à ce système de Listener au chapitre suivant.

Changeons notre programme comme suit :

```
Actor "magicien", Image$="magic.png", OnCollision$="COLLISION"
Actor "magicien", Control$ = "ArrowRight: offsetX=4; ArrowLeft:
offsetX=-4; ArrowUp: offsetY=-4; ArrowDown: offsetY=4"
Actor "lucie", 350, 350, Image$="lucie.png"
Do
 Wait Vbl
Loop
Procedure COLLISION [INDEX2$]
If INDEX2$ = "lucie" Then Boom : End
End Proc
```

Exécutez le programme. Déplacez l'actor en utilisant les touches fléchées du clavier pour entrer en collision avec Lucie ...Boom !

Regardons de plus près ce programme.

> **Ligne 1:** **OnCollision$** a été ajouté à l'instruction **Actor**. Sa propriété est une chaîne de caractères contenant le nom de la procédure qu'il faut lancer en cas de collision : **OnCollision$="COLLISION"**.

Note : Une procédure est un bloc de programme indépendant, qui sera appelé, ici, lorsqu'une collision se produit.

Ligne 3: nous ajoutons notre deuxième actor «lucie» à la position 350, 350.

Lignes 4 à 6 : une boucle simple pour maintenir le programme actif. (Pas indispensable mais apprenons ça en passant)

Wait Vbl, pour mémoire, est une instruction qui fluidifie l'affichage.

Ligne 7 : Début de la procédure appelée **COLLISION.** Pour créer une procédure vous devez mettre l'instruction **Procedure** suivie entre crochets du/des paramètre(s) pour communiquer avec la procédure (ici **INDEX2$**, dont le nom n'est pas choisi au hasard comme nous le verrons un peu plus tard et qui doit être en Majuscules, puis terminer par **End Proc**.

Donc le format est comme ceci :

Procédure COLLISION [INDEX2$]
....
....les instructions de votre procédure
....
End Proc

Lorsque vous insérez le nom de votre procédure n'importe où dans votre programme, il saute dans l'hyperespace quantique et exécute tout le code de la procédure, et quand il rencontre le **End Proc** l'exécution du programme revient juste après l'appel de la procédure.

LES PROCÉDURES ET LES FONCTIONS

Les procédures permettent aux programmes d'être divisés en parties indépendantes et réutilisables, je vous conseil fortement de les utiliser et ainsi organiser votre programme en parties ; vous apprécierez rapidement son utilité.

Une question qui vient rapidement est celle de la différence entre une Procedure et une fonction (Function) ?
A l'origine en BASIC (le langage de base d'AOZ) une Function avait une valeur de retour, alors qu'une Procedure n'en avait pas. Pour appeler une Procedure, il suffit d'invoquer le nom de la procédure et pour appeler une fonction, vous définissez une variable qui va contenir le résultat de la Function. En AOZ nous avons ajouté la possibilité aux Procedures de retourner également une valeur, comme une fonction.

La différence entre les deux se réduit maintenant à :
-Une Function peut être ajoutée au langage via une extension.
-Une Procedure qui n'a pas de valeur de retour peut être invoquée simplement par son nom, ou par Proc suivi du nom.

Example :

```
Print MyProc[2,3]
Print MyFunc(2,3)
ProcOnly[2,3]
Proc ProcOnly[2,3]
End

Procedure MyProc[a,b]
End Procedure[a*b]

Function "MyFunc",a,b
End Function(a*b)

Procedure ProcOnly[a,b]
   Print "pas de valeur de retour"
End Procedure
```

Note : un autre moyen de découper votre programme en morceaux est d'utiliser **INCLUDE** dont nous parlerons plus tard.

Dans ce programme, la procédure teste la variable nommée INDEX2$ qui contient le nom de l'actor en collision.

Rappelez-vous le nom de la procédure est fixé par le paramètre de **OnCollision$** (cf ligne 1).

Lorsque l'actor «magicien» entre en collision avec un autre actor la procédure COLLISION est automatiquement appelée et le nom de l'actor en collision est transmis dans la variable INDEX2$.

> **Ligne 8:** Nous testons si INDEX2$ est l'actor : «lucie», si c'est le cas, un effet sonore terrible "boom" est émis et le programme s'arrête
> **Ligne 9 :** Fin de la procédure, ce qui veut dire que le programme retourne juste après l'appel de la procédure et continue à la ligne suivante.

Collision coup de foudre

DES SOURIS ET DES ACTORS

Nous venons de voir qu'il était possible de contrôler un Actor avec le gamepad, le clavier ou la souris, afin de le déplacer sur l'écran.

On peut également rendre notre Actor sensible aux actions de la souris (ou au touché de l'écran tactile, c'est vu par AOZ de la même manière afin d'être compatible). Il pourra ainsi réagir lorsque l'on clique dessus (mais doucement on vient de dire qu'il est sensible), ou que le pointeur de la souris passe dessus.

OnMouse$

Pour cela, nous allons utiliser le paramètre **OnMouse$** de l'instruction Actor, il va vous faire gagner beaucoup de temps !

Voyons un exemple simple :

```
Actor "magic", X=100, Y=100, Image$="magic.png", OnMouse$="CLICK"
do
  Wait key
Loop

Procedure CLICK[EVENT$]
  If EVENT$="mouseclick" Then Print "Click!"
End Proc
```

Notre première ligne affiche l'Actor « magic » aux coordonnées 100,100. Le paramètre **OnMouse$** indique le nom de la procédure AOZ à appeler à chaque action de la souris sur cet Actor. La procédure AOZ qui sera appelée ici est CLICK.

Cette procédure, comme nous allons le voir, peut récupérer un certain nombre d'informations. Dans cet exemple la procédure CLICK récupère l'information EVENT$, qui contient le type d'action produit par la souris sur l'Actor.

Le contenu de notre procédure affiche le mot « Click ! » si la valeur de EVENT$ est « mouseclick » (correspondant à un click de la souris.)

EVENT$ peut prendre les valeurs suivantes :

- « **mouseclick** » : lorsque l'utilisateur clique sur l'Actor
- « **mousedown** » : lorsque l'utilisateur maintien un bouton de la souris enfoncé
- « **mouseup** » : lorsque l'utilisateur relève un bouton de la souris
- « **mousemove** » : lorsque l'utilisateur promène le pointeur de la souris sur l'Actor.
- « **dragdrop** » : lorsque l'utilisateur déplace l'Actor à l'aide de la souris

Modifions maintenant notre procédure CLICK_MAGIC ainsi :

```
Procedure CLICK[EVENT$, BUTTON, INDEX$, X, Y]
 Locate 1,1: Print "  Actor:"+INDEX$ + "  EVENT$:"+EVENT$ + "
BUTTON:"+Str$(BUTTON) + "X:"+Str$(X) + "Y:"+Str$(Y)
End Proc
```

Lorsque vous agissez sur « magic » avec la souris, des informations s'affichent en haut de l'écran.

La procédure CLICK_MAGIC propose plusieurs propriétés :

- **BUTTON** : est un entier correspondant au bouton de la souris qui est appuyé :
  - 0 : Pas de bouton appuyé
  - 1 : Le bouton gauche
  - 2 : Le bouton droit
  - 3 : Le bouton central ou molette (s'il existe)
- **INDEX$** : retourne le nom de l'Actor concerné par l'action de la souris. Si vous souhaitez également capturer les Actors identifiés par un numéro, vous pour utiliser INDEX.
- **X** et **Y** : retournent la position de la souris relative à l'intérieur de notre Actor.

Glisser / déposer

Modifions encore notre procédure :

```
Procedure CLICK[EVENT$, INDEX$, DRAGX, DRAGY ]
  If EVENT$="dragdrop" Then Actor INDEX$, X=DRAGX,
Y=DRAGY
End Proc
```

Ici, nous testons le type d'action retourné par **EVENT$**. Si l'action est « **dragdrop** », c'est que l'utilisateur a placé sa souris sur l'Actor, maintien un bouton de la souris enfoncé, et se déplace.

Les coordonnées du déplacement sont contenues dans 2 autres informations : DRAGX et DRAGY.

Notre procédure, à chaque appel, place alors notre Actor à ces nouvelles coordonnées.

Vous pouvez par exemple ainsi faire une fenêtre et la déplacer en une seule instruction.

11. ÉVÉNEMENTS

Une fois que l'instruction **Actor** est appelé, notre actor peut être « surveillé » constamment par plusieurs Listeners. Ils vérifient, scrutent l'activité de notre actor et alertent le programme qu'un événement s'est produit.

Un événement peut être :
- une collision
- un changement de position
- un changement d'image
- une touche appuyée

Important : Le nom d'un Listener commence toujours par **On...** et ce type de paramètre attend le nom de la procédure qui doit être appelé.

Dans notre programme précédent, lorsque nous avons défini **OnCollision$** nous avons demandé au Listener de collision d'aviser notre programme quand donc une collision se produit et d'appeler la procédure que nous avons appelée COLLISION en échangeant les informations demandées.

Dans notre exemple, c'est le nom de l'actor qui était transmis à la procédure. Mais, comme nous allons le voir dans les pages suivantes, vous pouvez transmettre bien d'autres informations (ou propriétés).

Notez que vous n'avez pas besoin de transmettre **toutes** les propriétés de l'actor à votre procédure AOZ, mais uniquement celles qu'elle devra traiter.

Vous disposez donc de plusieurs Listener chacun avec ses propriétés. Par convention ces listeners sont en Majuscules :

ONCOLLISION$

Ce listener alerte et informe sur les collisions entre Actors.

- **EVENT$** : nom de l'évènement (toujours "**on_collision**").
- **INDEX1** : valeur de l'indice de l'actor surveillé.
- **INDEX1$** : nom de l'actor surveillé.
- **INDEX2** : valeur de l'indice de l'actor entré en collision.
- **INDEX2$** : nom de l'actor entré en collision
- **IMAGE** : valeur de l'index de l'image de l'actor en collision
- **IMAGE$** : nom de l'image utilisée par l'actor en collision.
- **GROUP1$** : nom du groupe de l'actor surveillé
- **GROUP2$** : nom du groupe de l'actor en collision
- **ANIM1$** : nom de l'animation de l'actor surveillé
- **ANIM2$** : nom de l'animation de l'actor en collision avec

ONCONTROL$

Ce listener retourne les valeurs du **Control$** activé pour l'actor. Par exemple si **Control$="keyboard"** cela donnera des valeurs en rapport avec le clavier. "mouse" pour la souris,…

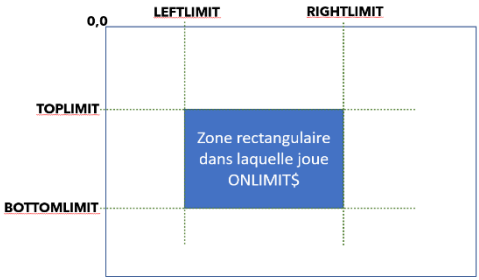
- **EVENT$:** nom de l'événement (toujours "**on_control**").
- **CONTROL$** : nom de l'instruction de cet événement.
- **INDEX** : valeur numérique de l'indice de l'actor surveillé.
- **INDEX$** : nom correspondant à l'actor surveillé.
- **BUTTON** : index du bouton de la souris qui a été activé.
- **ALTKEY** : renvoie vrai ou faux si la touche Alt a été pressée ou non.
- **CTRLKEY** : renvoie vrai ou faux si la touche Ctrl a été pressée ou non.
- **SHIFTKEY** : renvoie vrai ou faux si la touche Shift a été pressée ou non.
- **METAKEY** : renvoie vrai ou faux si le bouton « Windows » ou « CMD » (Mac) a été appuyé ou non.

- **MOUSEX** : valeur correspondante à la position horizontale de la souris.
- **MOUSEY:** valeur correspondante à la position verticale de la souris.

ONLIMIT$

Vous pouvez limiter le déplacement d'un actor à une zone rectangulaire, par exemple pour l'empêcher de sortir de l'écran. La procédure définie dans Onlimit$ est alors appelée lorsque l'actor touche l'un des 4 bords de cette limite, en passant les propriétés suivantes :

- **EVENT$** : nom de l'évènement (toujours "**on_limit**«).
- **INDEX** : valeur de l'indice de l'actor surveillé.
- **INDEX$** : nom de l'actor surveillé.
- **LIMIT$** : nom du bord ("left", "right", "top", "bottom")
- **leftlimit** : coordonnée X
- **rightlimit** : coordonnée X
- **toplimit** : coordonnée Y
- **bottomlimit** : coordonnée Y
-



ONANIMCHANGE$

SI vous avez animé l'actor (avec le paramètre Spritesheet$) **OnAnimChange$** appelle la procédure AOZ définie en lui transmettant les propriétés suivantes :

- **EVENT$** : nom de l'évènement «change» ou «complete». Si c'est «complete», c'est que l'animation est terminée.
- **INDEX** : valeur de l'indice de l'actor surveillé.
- **INDEX$** : nom de l'actor surveillé.
- **ANIM$** : nom de l'animation actuelle.

- **FRAME** : numéro de l'image actuellement en cours de lecture pour l'animation.
- **TOTALFRAMES:** le nombre d'images de l'animation.

ONCHANGE$

Ce Listener appelle la procédure AOZ définie chaque fois qu'il y a un changement sur l'actor concernant sa position, sa transparence, sa taille ou sa rotation.

- **EVENT$** : nom de l'évènement : «change» ou «complete». Si le nom est «complete» c'est que le mouvement de l'actor est terminé.
- **INDEX** : valeur de l'indice de l'actor surveillé.
- **INDEX$** : nom de l'actor surveillé.
- **X** : position X de l'actor.
- **Y** : position Y de l'actor.
- **ALPHA** : opacité de l'actor (aspect visuel).
- **SCALEX** : taux d'expansion/réduction horizontal
- **SCALEY** : taux d'expansion/réduction vertical
- **ANGLE** : angle de rotation de l'actor
- **HREV** : vrai si l'actor est tourné horizontalement, faux sinon.
- **VREV** : vrai si l'actor est tourné verticalement, faux sinon.
- **SKEWX** : distorsion horizontale de l'actor
- **SKEWY** : distorsion verticale de l'actor

Bien sûr, vous n'avez pas à surveiller chaque actor. Vous pouvez laisser vos actors vivre leur vie… et ne vous préoccuper que ce qui vous est utile, comme tout bon magicien qui se respecte !

Bon, vous n'avez surement pas tout retenu et vous vous posez des tas de questions ; je vous propose un exemple de mise en œuvre qui devrait vous permettre d'éclaircir tout ça.

COMMENT UTILISER LES LISTENERS

Voici un exemple d'utilisation du paramètre Listener pour un scrolling attaché aux touches droite-gauche du clavier :

```
Actor 1, Image$="bg.png", Y=0, Control$="ArrowLeft: offsetX=5;
ArrowRight: offsetX=-5", OnChange$="ON_CHANGE"
Do
 Wait Vbl
Loop
Procedure ON_CHANGE [INDEX$,X]
 If X > 0 Then Actor INDEX$, X = -1980
 If X < -1980 Then Actor INDEX$, X = 0
End Proc
```

Vous voyez avec cet exemple que:

- **OnChange$** appelle la procédure «ON_CHANGE» chaque fois que l'actor subit un changement de position (en raison du déplacement par le clavier)
- **Onchange$** appelle la procédure donnant en même temps les propriétés demandés, ici **INDEX$** et **X**

Ainsi, lorsque l'actor 1 est déplacé sur l'écran la procédure est appelée avec le nom de l'actor dans **INDEX$** et sa position **X** que le code dans la procédure peut utiliser, ici pour vérifier si nous atteignons la limite de 1980 pixels (rappelez-vous c'est la moitié de la taille du graphisme) et recommencer si c'est le cas.

-

INSTRUCTIONS UTILES AVEC ACTOR

Actor Reset

Réinitialise toutes les propriétés d'un actor. Exemple :

```
Actor Reset 1
Actor Reset "magic"
```

Actor Del

Supprime tous les actors, ou si le nom est spécifié un actor en particulier (qui ne s'affiche donc plus). Exemple :

```
Cls 0
Actor "forest", image$="bg"
Actor "Magic", X=100, Y=50, image$="magic"
Wait 2
Actor Del "Magic"  // efface seulement l'Actor avec le nom Magic
Actor Del          // efface tous les Actors
```

Le paramètre Visible d'Actor

Le paramètre Visible=True ou False affiche ou pas l'actor

```
Actor "magic", X=100, Y=100, Image$="magic.png"
Wait Key : Actor "magic", Visible=False
Wait Key : Actor "magic", Visible=True
```

Le paramètre Enable d'Actor

Si Enable=False (Enable=True ou False), alors, même si l'actor est contrôlé par Control$ et des animations, l'image restera affichée mais tous les comportements seront figés. Très pratique pour faire un bouton cliquable de votre interface.

```
Actor "magic", X=100, Y=100, Image$="magic.png",
Control$="keyboard", Enable=False
Do :  Wait Vbl : Loop
```

12. ANIMATIONS

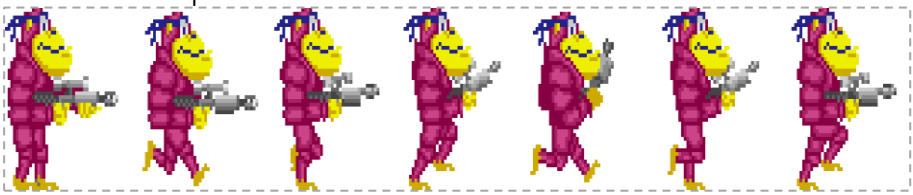
C'est très agréable d'afficher des images, mais il est certain que vous aurez envie de les animer.

Dans un jeu, les héros et les ennemis sont rarement immobiles, alors voyons comment animer vos images avec une technique bien connue du jeu vidéo : la feuille de sprites.

QU'EST-CE QU'UNE FEUILLE DE SPRITES ?

Une feuille de sprites, ou spritesheet, est une grande image contenant une séquence d'images successives qui trompent l'œil lorsqu'elles sont affichées l'une après l'autre, et donnent l'illusion d'un mouvement animé.

Voici un exemple :



L'instruction Actor offre plusieurs paramètres pour l'utilisation d'une spritesheet:

- **Spritesheet$** est le nom de la feuille de sprites à utiliser.
- **Anim$** est le nom de l'animation à jouer.

Regardez l'exemple dans Tutorials/2nd Game/shoot_em_up. Ce paragraphe est temporaire car des instructions spécifiques aux animations sont prévues ; plus à venir...

13. PARAMETRES D'ACTOR

Pour résumer ce dont nous avons discuté et plus encore, voici la liste complète des différents paramètres disponibles avec l'instruction **Actor** au moment de la rédaction de ce manuel.

- **X :** position horizontale de l'actor à l'écran. 0 par défaut
- **Y :** position verticale de l'actor à l'écran. 0 par défaut.
- **Z :** plan de l'actor à l'écran. 0 par défaut. Un Actor utilisant le plan 1 sera affiché devant tous les actors de plan 0. Plus le numéro de plan est élevé plus l'actor sera au premier plan.
- **StartX :** position horizontale de départ du mouvement.
- **StartY :** position verticale de départ du mouvement.
- **EndX :** position horizontale finale du mouvement.
- **EndY :** position vertical finale du mouvement.
- **Duration :** durée du mouvement de l'actor entre les deux points (Start et End).
- **Image$:** le numéro ou le nom de l'image à utiliser pour l'actor.
- **OnChange$:** pour assigner le nom de la procédure AOZ à appeler lorsque l'actor subit un changement (mouvement, forme, transparence...)
- **OnMouse$:** pour créer des actions sur le comportement de la souris

- **Transition$:** pour assigner le nom de l'effet à donner au mouvement de l'actor. Par défaut, il s'agit de la transition « linéaire ».

AOZ utilise la belle librairie CREATE.JS (c'est un excellent produit) pour les effets de **Transition$**.

Allez sur cette page pour visualiser les effets de transition à votre disposition :
https://www.createjs.com/demos/tweenjs/tween_sparktable

Ease Equations

backIn, backInOut, backOut,
bounceIn, bounceInOut,
bounceOut, circIn, circInOut,
circOut, cubicIn, cubicInOut,
cubicOut, elasticIn, elasticInOut,
elasticOut, linear/none, quadIn,
quadInOut, quadOut, quartIn,
quartInOut, quartOut, quintIn,
quintInOut, quintOut, sineIn,
sineInOut, sineOut,

Custom Eases

getBackIn, getBackInOut,
getBackOut, getElasticIn,
getElasticInOut, getElasticOut,
getPowIn, getPowInOut,
getPowOut

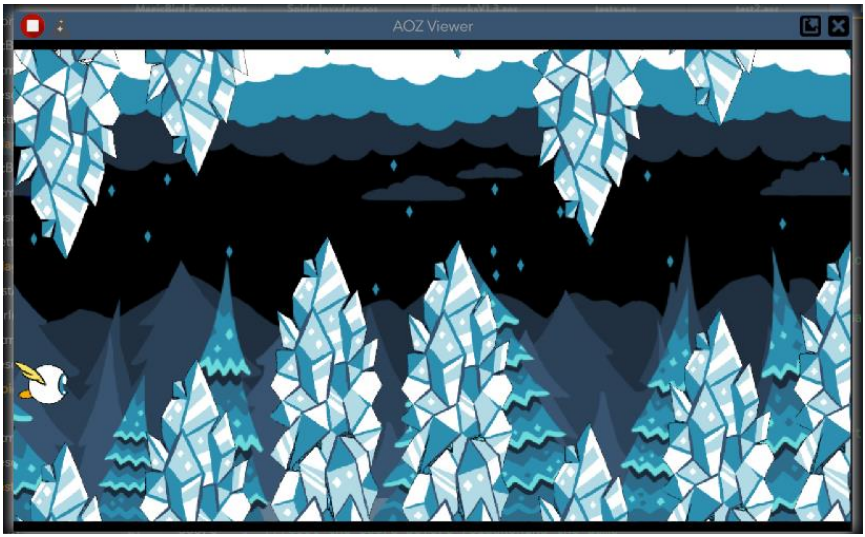
- **Scale :** taille de l'actor. 1.0 correspond à l'image dans sa taille d'origine, 1.2 correspond à une image agrandie de 20%, 0.5 correspond à une image réduite de moitié etc.
- **StartScale :** la valeur décimale de la taille de départ pour déplacer l'actor.
- **EndScale :** la valeur décimale de la taille à la fin pour le mouvement de l'actor.
- **ScaleX :** la valeur décimale de la largeur de l'actor. Par défaut 1.0 (taille normale)
- **StartScaleX :** la valeur décimale de départ de la largeur du mouvement de l'actor.
- **EndScaleX :** la valeur décimale à la fin du mouvement de la largeur l'actor.
- **ScaleY :** la valeur décimale de l'échelle de hauteur de l'actor. Par défaut 1.0 (taille normale)
- **StartScaleY :** la valeur décimale de la hauteur de départ du mouvement de l'actor.
- **EndScaleY:** la valeur décimale de la hauteur à la fin du mouvement de l'actor.
- **Angle :** la valeur en degrés de l'angle de l'actor. Par défaut 0.
- **StartAngle :** la valeur en degrés de l'angle initial pour le mouvement de l'actor.
- **EndAngle :** la valeur en degrés de l'angle final pour le mouvement de l'actor.
- **Alpha :** la valeur décimale de la transparence de l'actor. Par défaut 1.0 (totalement visible).
- **StartAlpha :** la valeur décimale de l'opacité initiale pour le mouvement de l'actor.
- **EndAlpha :** la valeur décimale de l'opacité finale pour le mouvement de l'actor.
- **SkewX :** la valeur de la distorsion (ou étirement) horizontale de l'actor
- **StartSkewX :** la valeur de départ de la distorsion horizontale du mouvement de l'actor.

- **EndSkewX :** la valeur de la fin de la distorsion horizontale du mouvement de l'actor.
- **Skew Y :** la valeur de la distorsion verticale de l'actor
- **StartSkewY :** la valeur du départ de la distorsion verticale du mouvement de l'actor.
- **EndSkewY :** la valeur de la fin de la distorsion verticale du mouvement de l'actor.
- **Visible :** True /False. Montre ou cache l'actor. Très utile pour cacher temporairement un actor, puis le réafficher.
- **Spritesheet$:** le nom de feuille de sprites à utiliser pour l'animation de l'actor
- **Anim$:** le nom de l'animation à jouer (disponible avec la feuille de sprites)
- **LoopAnim :** True/False jouer une animation en boucle ou non.
- **Hotspot$:** détermine la position du « point chaud » de l'actor. (Pour les valeurs à utiliser voir le chapitre Hotspot)
- **HotspotX :** valeur numérique qui détermine la position horizontale du « point chaud » de l'actor
- **HotspotY :** valeur numérique qui détermine la position verticale du « point chaud » de l'actor.
- **OnAnimChange$:** assigne le nom de la procédure qui sera appelé pour chaque changement de l'animation
- **OnCollision$:** nom de la procédure qui sera appelé lorsque l'actor entre en collision avec un autre actor.
- **LoopMove :** True/False Joue un mouvement en boucle.
- **ActionMove$:** "Play" / "Pause" le mouvement
- **Control$:** moyens de contrôles pour diriger l'actor ("keyboard", "mouse", "joystick").
- **OnControl$:** assigne le nom de la procédure qui sera appelé pour contrôler le mouvement d'un control$ (OnControl$ fonctionne avec Control$).
- **Hrev :** True/False. Symétrise (miroir horizontal) l'actor
- **Vrev :** True/False. miroir vertical de l'actor

- **LeftLimit :** Valeur définissant la limite à gauche du cadre de déplacement de l'actor
- **RightLimit :** Valeur définissant la limite à droite du cadre de déplacement de l'actor
- **TopLimit ou UpLimit :** Valeur définissant la limite supérieur (en haut) du cadre de déplacement de l'actor
- **BottomLimit ou DownLimit :** Valeur définissant la limite inférieur (en bas) du cadre de déplacement de l'actor
- **OnLimit$:** Assigne le nom de la procédure qui sera appelé lorsque l'actor atteint l'un des bords limite.
- **LookAt$:** Objet, actor ou point de l'écran que l'actor doit regarder. (Cf exemples en début de ce Guide).

14. FAISONS UN JEU

Vous en savez déjà assez pour faire un jeu : Magic Bird



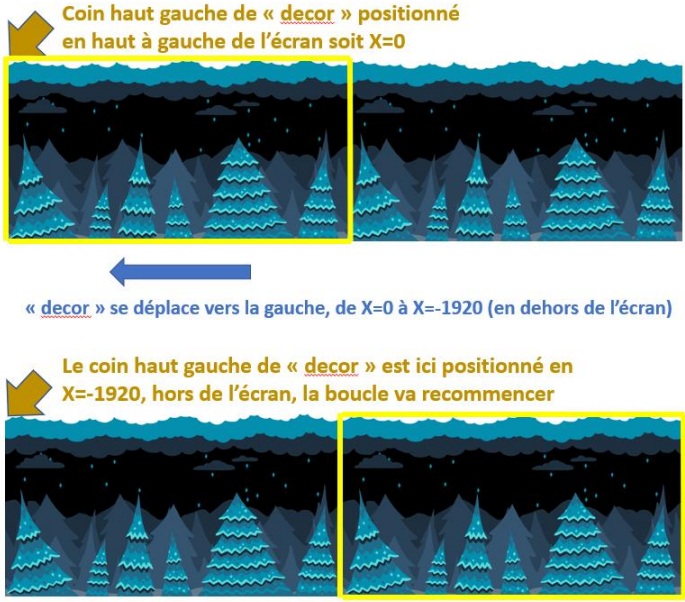
Le but du jeu est de faire voler l'oiseau Magic en évitant les stalactites. Pour jouer, un clic sur le bouton de la souris (ou l'appui sur l'écran du smartphone) permet à Magic de s'envoler.

Sans plus perdre de temps on va le programmer. Vous trouverez le code correspondant dans le dossier Tutorials, code que vous pouvez copier/coller pour les plus empressés. Pour les autres apprenons étape par étape. Donc, sans vous commander, tapez cette ligne de code (c'est une seule ligne) :

Actor "decor", Image$="bg.png", X=0, Y=0, EndX=-1920, LoopMove=True, Duration=20000

Si vous avez suivi les chapitres précédents rien ici ne doit vous étonner.
En utilisant l'instruction Actor nous créons donc un actor que l'on appelle ici "decor", et qui utilise l'image qui se nomme "bg.png". L'instruction susmentionnée affiche alors "decor" avec les paramètres que nous allons lui indiquer :

- on place au départ "decor" en haut à gauche de l'écran soit aux coordonnées X=0 et Y=0 et
- on dit à "decor" de déplacer son coin haut-gauche :
    - de la position X=0 (en haut à gauche de l'écran)
    - jusqu'à ce soit en EndX=-1920 et
- de boucler (LoopMove=True)
- avec la durée du mouvement Duration=20000. (note : 20000 millisecondes= 20 secondes pour une boucle)



Mais vous vous dites sans doute pourquoi il radote on a vu ca au début de ce guide utilisateur.

1. Alors faites RUN
2. Voyez le décor tourner en boucle, en boucle, en boucle, en boucle, en boucle, en boucle, en boucle

On continue, maintenant il nous faut un héros, qui mieux que Magic ?
Alors on ajoute en dessous de la 1er ligne cette nouvelle ligne :
Actor "magic", Image$="magicfly.png"

1. Faites RUN
2. Voyez le décor tourner en boucle et regardez Magic s'afficher en haut à gauche donc en X=0, Y=0. Normal on ne lui a rien dit, il prend les valeurs par défaut.

Je ne sais pas vous mais moi je n'aime pas voir Magic coincé la haut, alors je modifie le code comme suit en ajoutant une boucle Do…Loop pour déplacer Magic sur l'axe vertical des Y :

```
Actor "decor", Image$="bg.png", X=0, Y=0, EndX=-1920,
LoopMove=True, Duration=20000

Do
If Mouse Key = 0 then PY = PY+7 else PY = PY-15
Actor "magic", Y=PY, Image$="magicfly.png"
Wait vbl
Loop
```

Pas de panique j'explique. Et avant de vous plaindre sachez que pour faire la même chose dans un langage comme JavaScript, C#,… il faut au moins 5 fois plus de lignes de code.

Regardons ce code :

La boucle **Do...Loop** permet d'exécuter -sans s'arrêter- le code qui se trouve à l'intérieur. Ce code à l'intérieur modifie la variable PY selon l'appuis ou non sur le bouton de la souris.

Un petit rappel sur l'instruction **If...Then...Else** :
If *mon test* **Then** *je fais ca si c'est vrai* **Else** *je fais ca si c'est faux*
Exemple : If 2=2 Then Print"vrai" Else Print"faux"

Compris ? On continue avec notre jeu :

Explication de : **If Mouse Key = 0 then PY = PY+7 else PY = PY-15**

Si **Mouse Key = 0** cela veut dire que vous n'appuyez pas sur le bouton (Key) de la souris (Mouse) -> dans ce cas PY augmente de 7 pixels (PY=PY+7). Sur l'axe des Y quand cela augmente cela veut dire que l'on descend à l'écran (rappel Y=0 est tout en haut de l'écran).

Si **Mouse Key** n'est pas égal à 0 cela veut dire qu'à ce moment là vous n'appuyez pas sur le bouton de la souris alors on est dans le **else** -> PY monte de 15 pixels (PY=PY-15) sur l'axe des Y. En effet quand PY diminue c'est que l'on monte (vers le haut de l'écran).

Le résultat de cette ligne est que Magic descend tout seul de 7 pixels (quand on appui pas sur la souris) et remonte de 15 pixels si on appui sur la souris ou l'écran du smartphone.

Explication de : **Actor "magic", Y=PY, Image$="magicfly.png"**

Bon là vous avez compris, dans chaque itération de la boucle Magic se déplace aux nouvelles coordonnées PY et donc monte ou descend selon la ligne précédente.

Explication de : **Wait vbl**

Cette instruction permet de bien synchroniser l'affichage quand il est fréquemment modifié.

1. Faites RUN
2. Voyez le décor tourner en boucle et regardez Magic se déplacer de bas en haut avec l'appui sur la souris. Cool !

Maintenant nous allons mettre des obstacles, avec des stalactites qui vont se déplacer de droite à gauche (on parle de scrolling horizontal) et progressivement se rapprocher les uns des autres pour augmenter le niveau de difficulté. Ca vous va ? Non ? Pas de problème vous pourrez modifier tout ca, ce n'est qu'un exemple.

```
Actor "decor", Image$="bg.png", X=0, Y=0, EndX=-1920,
LoopMove=True, Duration=20000

For I= 1 to 50
  Actor I, X=400+(I*320), Y=-780+rnd(I*20), Auto$="offsetX=-5",
Image$="ice.png"
Next I

Do
If Mouse Key = 0 then PY = PY+7 else PY = PY-15
Actor "magic", Y=PY, Image$="magicfly.png"
Wait vbl
Loop
```

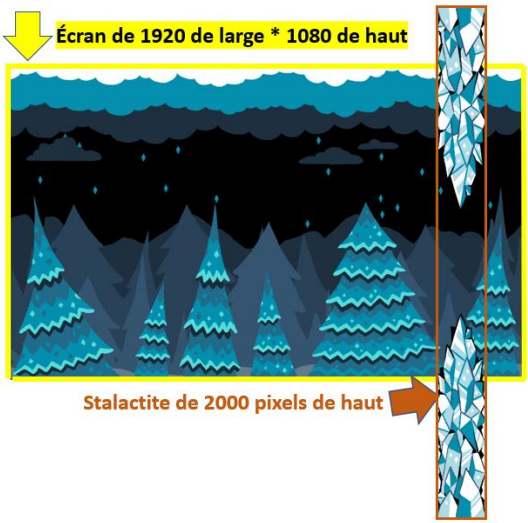
Je vous explique la partie que je viens de rajouter.

Un petit rappel sur l'instruction **For...Next** : C'est une autre forme de boucle mais conditionnelle. **For** *variable=valeur de départ* **To** *variable=valeur de fin* **Next** (*fin de boucle)*
Exemple : For I=1 to 10 : Print I : Next I

Explication de : **Actor I, X=400+(I*320), Y=-780+rnd(I*20), Auto$="offsetX=-5", Image$="ice.png"**
Dans la boucle For…Next qui compte jusqu'à 50, en augmentant la valeur de I de 1 à chaque itération, on crée donc 50 actors avec les noms : Actor 1, Actor 2, Actor 3, … Actor 50.

Remarquez que les actors peuvent donc avoir des noms comme "magic" ou "lucie" et alors on met les deux ", mais aussi des numéros comme dans ce jeu. C'est bien pratique, comme dans cet exemple ou l'on a besoin de manipuler 50 actors.

Chacun de ces 50 actors utilise la même image de stalactite (ice.png) qui comprend la partie haute et la partie basse et qui fait 2000 pixels de haut, soit plus que la hauteur de l'écran qui lui fait 1080 pixels de haut.



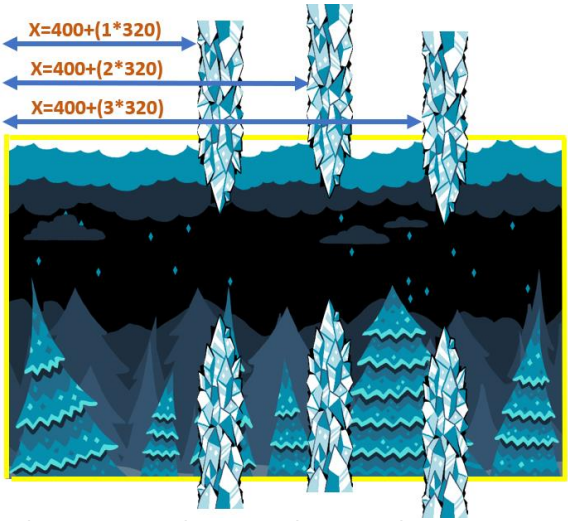
Pour afficher les stalactites à des hauteurs différentes il font donc juste déplacer leur coin haut gauche verticalement, donc en Y. C'est ce que l'on fait avec le paramètre **Y=-780+rnd(I*20)**. On place le Y avec la valeur -780 et on y ajoute une valeur aléatoire qui est le produit de I * 20.

La fonction rnd(x) retourne une valeur aléatoire comprise entre 0 et x. Par exemple rnd(10) retourne une valeur entre 0 et 10.

Ainsi pour l'actor 1 (quand I=**1**) on aura un Y=-780+rnd(**1***20), soit une valeur comprise entre Y=-780+1=-779 et Y=-780+20=-760. Plus I augmente plus l'amplitude de Y sera forte entre deux stalactites. Donc la difficulté du jeu augmentera. (faite le calcul ci-dessus pour I=50)

Pour afficher les stalactites à des positions différentes horizontalement il font donc les positionner également en X. C'est ce que l'on fait avec le paramètre **X=400+(I*320)**.

On place ainsi le X à la valeur 400 plus le produit de I * 30.



Ainsi plus la valeur de I augmente plus la stalactite est créé sur la droite.

Pour finir il suffit de préciser pour chaque Actor que le déplacement est automatique et vers la gauche (donc X diminue, et de 5 pixels) avec le paramètre **Auto$="offsetX=-5"**

Et voila avec ces paramètres qui modifient les X et les Y en une seule ligne d'AOZ on affiche toutes nos stalactites avec de l'aléatoire et une progression de difficulté.

À vous de jouer 😊 :

- Amusez-vous à changer les valeurs des paramètres, c'est le meilleur moyen pour comprendre.
- Comprenez-bien qu'il n'y a pas qu'une méthode pour arriver au résultat, c'est la beauté de la création. Je vous propose ici une façon de faire, mais un autre exercice serait d'y arriver avec un code totalement différent.

Vous êtes prêt, on continue ?
Sinon prenez-votre temps, relisez, modifiez,… Programmer doit être du plaisir.

Bien, on se dit à ce stade que pour finir notre jeu il manque quelque chose d'important : la gestion des collisions. C'est-à-dire si Magic touche les stalactites ou sort de l'écran, il doit, disons… disparaitre, non, non, temporairement.

Un petit rappel sur les étiquettes : n'importe où dans le code on peut placer une marque (ou étiquette) à un endroit, et demander au programme d'y aller. Pour cela l'endroit est noté avec un nom ici START: ou DEAD: Nom qui doit être suivi des deux points : pour le différencier d'une variable ou d'une instruction. Et pour demander au programme d'aller à cette étiquette il faut faire appel à l'instruction GOTO, ici GOTO START ou GOTO DEAD (sans les deux points :)

Voyons le code complété de notre jeu pour faire tout cela et je vous l'explique.

```
START:
Actor "decor", Image$="bg.png", X=0, Y=0, EndX=-1920, LoopMove=True,
Duration=20000

For I= 1 to 50
 Actor I, X=400+(I*320), Y=-780+rnd(I*20), Auto$="offsetX=-5",
Image$="ice.png"
Next I
PY=Screen Height/2 // Positionnons Magic au milieu de l'écran verticalement

do    // Début de la boucle
  If Mouse Key = 0 then PY = PY+7 else PY = PY-15
  Actor "magic",Y=PY, Image$="magicfly.png"
  Wait Vbl

  if Actor Col("magic", image$="ice") = True Then Goto DEAD
  If PY>1000 Then Goto DEAD
```

```
  Locate 0,7 : Pen 6: Print score: score=score+1
loop   //fin de la boucle

DEAD:  // Si Magic entre en collision cette partie du code est exécutée
  Actor "magic", Image$="Magic_Dead-0.png", Y=PY, ENDY=-20,
Auto$="offsetY=-15"
  Actor "gameover", X=660, Y=400, Image$="gameover.png"  //
Affichons l'image de GameOver
  Wait click  // Attendons un appui sur un bouton de la souris (ou sur
l'écran pour les smartphones)
  Del Actor "gameover" // Effacons l'image GameOver

 Goto START // Et on retourne au début du programme.
```

Il y a plein de choses nouvelles et intéressantes à expliquer. Ca va aller, on progresse, c'est bien.

Tout d'abord j'ai ajouté deux étiquettes **START**: et **DEAD:** et on voit les Goto qui correspondent. La dernière ligne est le **Goto START** qui dit donc simplement : retourne au tout début du programme (là ou est le **START:**) on va recommencer à jouer.

Explication de :
if Actor Col("magic", image$="ice") = True Then Goto DEAD
Nous avons déjà vu cela, c'est la fonction qui teste les collisions, ici entre l'actor "magic" (entre " ") et soit un autre actor soit (dans notre cas ici) une image qui s'appelle "ice". Bilan ? Dés que magic touche une stalactite, la fonction retourne que c'est vrai ou True et alors (Then) on va dans le programme à l'étiquette DEAD.

Explication de :
If PY>1000 Then Goto DEAD
On va également à l'étiquette DEAD si magic tombe sur le sol (en bas). Le sol est à Y=1080 pixels (la taille de l'écran), en prenant en compte la hauteur de magic dont le point chaud est en haut à gauche tester >1000 est suffisant.

Note : Remarquez que dans un cas le if est entièrement en minuscules et pas dans l'autre, c'est cool AOZ, on ne vous prend pas la tête avec cela.

Il nous reste à voir le code de DEAD et on a fini, voyez ce n'est pas si compliqué !

Explication de :
Actor "magic", Image$="Magic_Dead-0.png", Y=PY, ENDY=-20, Auto$="offsetY=-15"
Va afficher l'image du fantôme de magic, en partant de PY, soit la dernière position verticale de magic et en finissant l'animation en Y=-20 c'est-à-dire juste au dessus du haut de l'écran. Le fantôme va donc monter et disparaitre en haut de l'écran.

Explication de : **Wait click**
Cette instruction, comme vous vous en doutez attend que l'on clic sur un bouton de la souris;

Explication de : **Del Actor "gameover"**
Cette instruction arrête l'affichage de l'actor (il disparait), donc ici on efface l'image Game over, pour ensuite reprendre le jeu.

À votre tour 😊 :

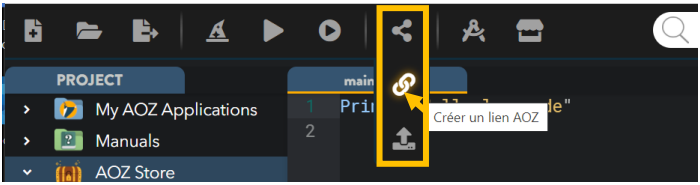
- Modifiez le code, les valeurs… mais étape par étape

15. PUBLIEZ EN UN CLIC

AOZ Studio permet de créer plus facilement, mais également de publier et partager vos programmes.

Plutôt que de prendre un serveur pour héberger votre programme, le monter (installer les logiciels), créer une URL (DNS), … des notions que l'on verra plus tard, AOZ Studio fait tout cela pour vous automatiquement avec le bouton Partager votre Projet aussi appelé PUBLISH qui propose deux fonctionnalités pour partager votre programme :

- Créer un lien AOZ : génère un lien web et son QR code
- Publier votre projet : dans l'AOZ Store



Oui je sais c'est top et cela mérite :

QUELQUES EXPLICATIONS

Comment utiliser le lien et le QR code de mon programme ?
> Lorsque vous allez utiliser Créer un lien AOZ, cela créé une URL (un lien internet) et son QR code uniques de votre programme, utilisables sur Ordinateurs et Smartphones (prenez le QRcode en photo cela activera le lien URL dans le navigateur par défaut).

Où sont stockés mes applications quand j'utilise PUBLISH ?
>Vos programmes sont convertis en HTML, Javascript (le langage des navigateurs internet) et sont stockés dans les serveurs d'AOZ Studio.

Est-ce associé à un compte, un identifiant unique ?
>Oui bientôt, nous sommes entrain de construire l'AOZ Store. Une sorte de magasin digital dans lequel vous pourrez retrouver les programmes que vous avez publiés, ceux que vous voulez partager, voire commercialiser. C'est là que vous pourrez les supprimer par exemple. Pour le moment vous ne pouvez que publier. Veuillez noter que pour le moment la gestion de la sécurité est très sommaire, conservez toujours une copie de vos programmes sources, nous ne pouvons garantir la présence permanente de vos programmes publiés.

Je n'ai pas accès à PUBLISH sans la licence d'AOZ Studio ☹ ?
>Et non. Comme vous l'avez vu vous pouvez utiliser AOZ Studio sans limite, sans option payante et pour tous les appareils, …mais pour publier ou faire une sauvegarde au format .AOZIP, utiliser les serveurs d'AOZ Studio, comme d'avoir le droit de publier vos applications il faut avoir une licence valide. Cela débloquera ces fonctionnalités et vous assurera d'avoir la dernière version avec toutes les nouveautés.

MAINTENANT ON PUBLIE NOTRE OEUVRE !

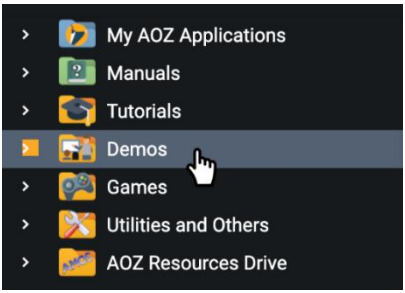
Appuyez sur Créez un lien, attendez un peu et vous aurez un écran comme ci-dessous avec un lien internet (URL) et le QRCode correspondant pour ordinateurs et smarphones !

Écran de fin du Publish pour partager le programme très simplement :
- avec le lien et
- son QR code.

16. LES DÉMOS ET LES JEUX

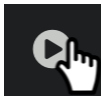
Maintenant, nous allons enfin jouer un peu !

Regardez en haut à gauche de votre écran la liste des répertoires d'AOZ Studio et cliquez sur le dossier Démos ou Games.



Ouvrez un dossier et cliquez sur le **main.aoz** (**main.** ou tout autre nom). Voyez le code de ces programmes, beaucoup d'entre eux ont été développés en une matinée entre la promenade du chien et une pizza.
Note : vous n'avez pas besoin de manger de pizzas pour programmer en AOZ.

Cliquez ensuite sur l'un des deux boutons RUN, comme celui-ci, pour exécuter ce programme dans votre visionneuse AOZ.



AOZ Studio va progressivement vous livrer tous ses secrets, des tours de magie qui vous enchanterons et impressionnerons les autres.

Nous vous laissons découvrir les démos, les jeux. Et bientôt, nous espérons que vous partagerez les vôtres.

Bien, à partir d'ici cela va devenir moins drôle.

Nous allons aborder des chapitres permettant d'écrire des programmes sophistiqués.

Vous pouvez consulter ces chapitres selon vos besoins sans nécessairement avoir à les lire dans l'ordre du guide.

Le mieux est de vous fixer un objectif, mais c'est sans doute déjà fait, par exemple écrire un jeu, ou une application avec l'aide du DESIGNER.

17.ENTRÉES CLAVIER

AOZ dispose d'une gamme complète d'instructions de clavier.

Vous pouvez lire les touches une à la fois, y compris les touches de contrôles (Ctrl, Shift, Alt, Meta), et les touches comme Caps lock, etc. Certains codes sont stockés dans le buffer (tampon d'entrée) du clavier, qui est une zone de mémoire réservée aux touches pressées, mais pas encore traitées. Vous pouvez détecter l'appui simultané de plusieurs touches.

Il existe 4 types d'instructions d'entrées au clavier :

- **Variable** : cette entrée reçoit des données à partir du clavier de n'importe quel type simple, comme une chaîne de caractères, un entier ou un nombre et les alimente directement dans chaque variable d'entrée. Plusieurs variables peuvent être séparées par des virgules (ou de la touche Enter).

- **Character** : ce type d'entrée ne reçoit que des chaînes de caractères à partir du clavier. Un character se réfère à n'importe quelle lettre, chiffre, symbole spécial, etc.

- **Buffer** : ce type permet d'interagir directement avec la mémoire tampon du clavier.

- **State** : ce type d'entrée vérifie l'état actuel d'une touche pour voir si elle est pressée ou non, mais n'a pas à remplir une variable avec le résultat.

Saisies clavier de mots

Input

Nous avons 4 formes pour l'instruction Input :

Input VARNAME { , VARNAME$, VARNAME#, ... }

Description : cette instruction accepte la saisie au clavier pour les variables simples. Ce qui est entré au clavier est inséré dans la variable spécifiée par VARNAME, VARNAME$, etc. L'entrée sera affichée à l'écran au fur et à mesure qu'elle est tapée, avec une simple édition à l'aide de la touche Backspace. Chaque entrée doit être séparée par une virgule, ou la touche **Enter.** Lorsque la touche **Enter** est pressée cela affecte les données saisies dans la/les variable(s) spécifiée(s).

Paramètre(s) : Plusieurs paramètres de n'importe quel type de variable simple : ***(nombre entier, chaîne*** ou ***nombre)***. Chaque variable doit être séparée par une virgule ou une clé **Enter.**

Exemple simple :

```
Do
        Input A$
        Print A$
Loop
```

… ou plus sophistiqué :

```
Do
        Input A,B$,C#
        Print A,B$,C#
Loop
```

Input TEXT$; VARNAME { , VARNAME$, VARNAME#, ... }

Description : une autre forme de l'instruction Input précédemment décrite, qui affiche en plus un texte TEXT$ juste avant la saisie clavier. Sinon, la commande fonctionne idem.

Exemple :

```
Do
        Input "Enter A,B$, and C#: ";A,B$,C#
        Print "Inputs: ";A,B$,C#
        Wait Vbl
Loop
```

À l'exception de l'affichage "Enter A, B$, and C#:" juste avant d'accepter les entrées, cela fonctionne comme la commande d'entrée simple.

Line Input VARNAME { , VARNAME$, VARNAME#, ... }

Description : cette instruction fonctionne de manière identique à la commande Input, sauf qu'elle accepte tous les caractères imprimables, y compris la virgule. La touche **Enter** est utilisée pour séparer chaque entrée. Voir la description de la commande **Input** pour plus de détails.

REMARQUE : en cas d'erreur "Please redo from start" s'affiche, et l'utilisateur doit recommencer sa saisie depuis la première variable.

Line Input TEXT$; VARNAME {, VARNAME$ VARNAME#, ...}

Description : comme pour la commande **Input** standard, **Line Input** affiche un point d'interrogation avant d'accepter les entrées. Sinon cela fonctionne de la même manière que la commande **Line Input**. Voir la description de Line Input pour plus de détails.

Buffer de clavier

Ces commandes vous permettent d'interagir directement avec le buffer ou mémoire tampon du clavier.

***Mettez key K$*** *Put Key n'est pas actuellement implémentée, mais le sera prochainement. Description : cette instruction insère la chaîne spécifiée (K$) dans le tampon d'entrée, comme si elle avait été tapée au clavier. Ceci est souvent utilisé pour pré-remplir les valeurs dans les Input.*

```
Exemple :
Do
  Put Key "YES"
  Input "Do you like AOZ? ";A$
  A$=Upper$(A$)
Loop
```

Clear Key

Description : cette instruction efface le contenu du buffer. Cela peut être appelé avant **Wait key** ou d'autres commandes de clavier pour s'assurer que nous attendons une nouvelle entrée.

Key Speed

Key Speed TIMELAG, DELAYSPEED

Description : cette instruction définie la vitesse à laquelle l'entrée est acceptée. TIMELAG est le temps avant que la répétition commence. DELAYSPEED indique la vitesse à laquelle chaque appui clavier est accepté.

Saisies clavier de touches

N'acceptent généralement qu'un caractère à la fois.

Inkey$

Description : cette fonction examine le buffer pour voir si une touche a été pressée. Si oui, elle met ce caractère dans le résultat de la fonction et la supprime du buffer.

Valeur de retour : Le résultat (X$ ci-dessous) est une chaîne de caractère unique contenant la valeur de la touche pressée à partir du buffer. Si le buffer est vide au moment où **Inkey$** est appelé, alors le résultat est une chaîne vide : "".

Exemple 1 de Inkey$:

```
Centre "Press d or f " : Print
Do
  A$ = Inkey$
  If A$ = "d" Then print "d";
  If A$ = "f" Then print "f";
  Wait Vbl
Loop
```

Exemple 2 de Inkey$:

```
Result$=""
Do
        X$=Inkey$ : If X$ >= "@" And X$ <= "~" Then Print
X$; : Result$=Result$+X$
        If X$=Chr$(8) Then
Result$=Left$(Result$,Len(Result$)-1)
        If X$=Chr$(13) Then Exit
        Wait Vbl ' synchronise l'affichage (optionnel)
        Cls : Print Result$
```

```
Loop
Cls : Print Result$
```

Avec le code ci-dessus vous récupérerez chaque caractère saisis au clavier et vous les cumulerez dans **Result$**. La touche Backspace et les caractères spéciaux sont également manipulés.

Input$(N)

Description : cette fonction renvoie une chaîne de N caractères imprimables saisis par l'utilisateur. Une fois que le nombre spécifié de caractères (**N**) a été atteint, le résultat est retourné, et le traitement se poursuit avec l'instruction suivante. Aucune édition n'est possible.

Valeur de retour : Le résultat de la chaîne (X$) retourné a exactement N caractères de long.

Exemple :

```
Do
        X$=Input$(3)
        Print X$
        Wait Vbl
Loop
```

Le code ci-dessus accepte une chaîne de 3 caractères à partir du clavier et la met dans **X$**.

Lecture de l'état du clavier

Si Déplorable=TRUE, vous devriez le nettoyer avec une lingette ! Plus sérieusement…

Wait Key

Description : cette instruction attend qu'une touche du clavier soit pressée avant d'agir sur la prochaine instruction.

Exemple :

Print "Appuyez sur une touche" : Wait key : Print "Merci"

Dans l'exemple ci-dessus, «Merci» ne sera pas affiché tant qu'une touche n'aura pas été pressée.

Wait Input

Description : cette instruction très pratique attend qu'une touche ou un bouton soit pressé sur clavier, joystick, souris/écran tactile (pas sur gamepad) avant d'agir sur la prochaine instruction.

Exemple :

Wait Input : Print "On continue"

Wait Click

Description : cette instruction attend spécifiquement qu'un bouton soit pressé sur la souris/écran tactile avant d'agir sur la prochaine instruction.

Key State

Description : cette fonction retourne TRUE (vrai) lorsqu'une touche spécifique est pressée.

Paramètre : Le "scan code" de la touche à tester.

Valeur de retour : le résultat booléen est TRUE si la touche spécifiée par le scan code (est pressée.

Exemple :

```
CODE=$42 : REM "c'est le scan code de la lettre B"
Do
 If Key State(CODE) Then Print "touche : ";CODE;" (";Chr$(C
ODE);") était appuyée."
 Wait Vbl
Loop
```

Dans l'exemple ci-dessus, l'affichage ne se produira pas tant que la touche «B» n'aura pas été pressée.

Key Name$

Description : cette fonction renvoie le nom de la touche pressée.

Valeur de retour : le résultat de la chaîne contient une description textuelle (le nom) de la touche pressée.

Exemple :

```
Do
   Locate 1,1 : Print "tapez doucement sur le clavier : ";
   ik$ = Inkey$
   If ik$ <> ""
      Print ik$;"  ";
      k$=Key Name$ : Print k$;
```

```
    End If
    Wait Vbl
Loop
```

Note : cette fonction retourne les noms des touches du clavier QWERTY (pas AZERTY français) lorsque les touches seront pressées.

ScanCode (ScanCode)

Description : cette fonction retourne le scan code de la dernière touche du buffer d'entrée telle que lue par **Inkey$**.

Valeur de retour : Le résultat est un entier avec le code de la dernière touche pressée, ou 0 si le buffer d'entrée est vide.

Exemple :

```
Locate 0,0 : Print Manifest$
Do
        I$=Inkey$
        If I$ <> "" ' Test appuis sur une touche
                SC=ScanCode
                Locate 1,1 : Print I$, Hex$(SC,2) : SC=0
        End If
        Wait Vbl
Loop
```

L'exemple ci-dessus affichera le nom et la valeur de la touche en Hexadécimale.

Key Shift

Description : cette fonction détermine quelles touches modificatrices sont actuellement appuyées. Les touches

modificatrices sont celles qui modifient la valeur et/ou la fonction d'une autre touche pressée en même temps.

Valeur de retour : le résultat est un entier qui dit quelles touches modificatrices sont actuellement pressées. La valeur est comme suit :

Bit	Valeur	Touche
0	$0001	Shift gauche
1	$0002	Shift droit
2	$0004	Ctrl gauche (Caps Lock sur Amiga)
3	$0008	Ctrl droite (les deux Ctrl sur Amiga)
4	$0010	Alt gauche
5	$0020	Alt droit
6	$0040	Meta gauche (Amiga/Cmd/Windows)
7	$0080	Meta droite (Amiga/Cmd/Windows)
8	$0100	Caps Lock (en mode AOZ seulement)

Note pour Amiga :
15 $8008 Ctrl gauche (Amiga slmnt bit 3 pour compatibilité) plus de combinaisons sont possibles avec AOZ qu'avec un ordinateur Amiga. Par exemple, vous pouvez lire les touches de Shift gauche et droite en même temps. En outre, une nouvelle position de bit ($8000) a été ajoutée pour représenter la touche Ctrl gauche. Pour des raisons de compatibilité arrière, en mode Amiga, les touches Ctrl utilisent le bit 3.

REMARQUE MATÉRIELLE : en raison de différences matérielles concernant les claviers, certaines combinaisons de touches pressées peuvent ou non être possibles, alors essayez de faire simple et testez soigneusement votre programme, si possible avec différents claviers, afin d'éviter les problèmes de compatibilité.

ScanShift (ScanShift)

Description : cette fonction détermine quelles touches modificatrices ont été pressées simultanément avec la dernière touche lue dans le buffer (lu par Inkey$). Les touches modificatrices sont celles qui sont destinées à modifier la valeur et/ou la fonction des touches.

Cela fonctionne exactement comme la fonction **Key Shift**, sauf que ScanShift se rapporte à la dernière touche sur laquelle l'utilisateur a appuyé.

Valeur de retour : le résultat est un entier indiquant quelles touches modificatrices ont été pressées en même temps que la dernière touche pressée par l'utilisateur. Cf table plus haut avec **Key Shift**.

Exemple :

```
SS$=""
Do
        I$=Inkey$ ' Prend la touche du buffer clavier
        SS=ScanShift ' Prend l'état des touches modificatrices
        Locate 0,0 : Print Manifest$
        If I$<>""
                Locate 0,1 : Print I$;" "
                Locate 0,2: Print "SS: ";Right$(Bin$(SS,16),16)
                If SS <> 0
                If (SS & 1)=1 Then SS$=SS$ + "Left Shift| "
                If (SS & 2)=2 Then SS$=SS$ + "Right Shift| "
                If (SS & 4)=4 Then If Manifest$="amiga" Then
SS$=SS$+"Caps Lock|" Else SS$=SS$+"Left Ctrl| "
                If (SS & 8)=8 Then SS$=SS$+"Right Ctrl| "
                If (SS & 16)=16 Then SS$=SS$+"Left Alt| "
                If (SS & 32)=32 Then SS$=SS$+"Right Alt| "
                If (SS & 64)=64 Then SS$=SS$+"Left Meta| "
                If (SS & 128)=128 Then SS$=SS$+"Right Meta| "
                If (SS & 256)=256 Then SS$=SS$+"Caps Lock| "
'Seulement pour AOZ
```

```
                If (SS & $8000)=$8000 Then SS$=SS$+"Left Ctrl| "
'Seulement pour Amiga
                End If
                SS$=SS$+Space$(160) 'des espaces pour effacer
        Locate 0,3 : Print SS$
        SS$=""
        End If
        Wait Vbl
Loop
```

L'exemple ci-dessus affichera la dernière touche pressée avec les touches modificatrices qui ont été pressées en même temps.

Bien, tout cela peut paraitre un peu compliqué et pas très utile, pour les maniaques de la souris. Pas faux, pas faux… mais le prochain chapitre leur rendra sans doute le sourire !

Allez un petit truc en passant : pour écrire une instruction sur plusieurs lignes, vous pouvez utiliser le caractère \. Example :

```
Actor \
"Player",\
HotSpot$ ="middle",\
Control$="ArrowRight: offsetX = 6; ArrowLeft: offsetX = -6",\
LeftLimit=40,\
RightLimit=Screen Width + 40
```

18. FORMATAGE DU TEXTE

Ouvrons une application dans le dossier "Utilities and Others".
Dans le dossier appelé Font Viewer cliquez sur le programme : **main.aoz** comme d'habitude.

C'est une visionneuse de polices, vous découvrirez les polices intégrées dans AOZ Studio : elles couvrent la plupart des besoins.

Et si vous en voulez plus, vous pouvez aussi utiliser n'importe quelle police Google avec une simple étiquette (TAG).
Et vous pouvez même créer vos propres polices à l'aide de graphiques.

Voici comment :

1. Dites à AOZ d'intégrer une police dans votre application, en utilisant l'étiquette **#googleFont** pour une police Google (ou **#amigaFont** pour une police Amiga). Et si vous ne savez pas ce qu'est un TAG / une étiquette, il y a un chapitre dédié à la fin de ce guide.

2. La police doit être déclarée dans AOZ. Voici un exemple :

```
#googleFont: "Francois One"
Set Font "Francois One", 80, "bold"
Text Screen Width / 2, Screen Height / 2, "AOZ!", "# Center #"
```

3. Dans votre application, choisissez la police avec l'instruction « Set Font » et ses propriétés :

Set Font "font name", height, weight$, italic$, stretch$

Avec les valeurs possibles suivantes :

- weight$: "normal" "bold" ou "light"
- italic$: "normal"

4. Affichez le texte avec

Text x, y, "Mon texte", tags$

Avec les balises :

- alignement horizontal : #left #center #right
- alignement vertical: #top #middle #alphabetic #hanging #ideographic
- direction : #ltr / #rtl #inherit

ou afficher avec :

Format Text MyBigText$, x, y, width, height, tag$

Avec les balises :

- '#left', '#center', '#right', '#start', '#end' : alignement horizontal
- '#html' : indique un texte formaté en HTML
- '#md' : indique un texte de markdown
- '#nozones' : ignorera les commandes = Zone $ ()
- '#animate' : affichera les liens et les images affichés à l'écran en commençant par un process de détection de souris.

19. STOCKAGE DES IMAGES ET SONS

STOCKAGE DANS UNE BANQUE DE DONNEES

Une banque de données (memory bank) est une zone de mémoire de l'ordinateur, réservée pour stocker des images, des sons et autres fichiers dont votre programme AOZ aura besoin.
C'est différent d'une base de données qui est -généralement- stockée à l'extérieur de l'ordinateur et qui manipule de plus grands volumes d'informations.

AOZ Studio utilise des banques de données. Pour chaque application on trouve ses banques de données dans le dossier « resources » :

- 1.images : Les images utilisés pour les instructions Actor, Sprite ou Bob
- 2.icons : Les images d'icones
- 3.musics : Les sons utilisés pour les instructions de musique
- 5.samples : Les sons utilisés pour les effets sonores

Note, on trouve également dans le répertoire resources :

- Assets: Pour y stocker les images, sons… que vous utiliserez directement avec l'instruction Load Asset (voir plus loin).

Par exemple, si je place le fichier « my_image.png » dans le dossier « 1.images », et que je tape ce code :

```
Paste Bob 100, 100, "my_image"
```

Ou ce code :

```
Actor "image", X=100, Y=100, Image$="my_image"
```

Mon image s'affichera à l'écran aux coordonnées 100,100. Le nom du fichier est alors utilisé comme référence de mon image.

Si je place maintenant le fichier « 1.png » dans le dossier « 1.images », je pourrais utiliser ce code :

```
Paste Bob 100, 100, 1
```

Ou ce code :

```
Actor 1,  X=100, Y=100,  Image=1
```

Le nom de mon fichier est automatiquement transformé en numéro. Ainsi, je peux utiliser ce numéro comme référence de mon image.

Cela fonctionne également pour les sons. Si je place les fichiers « my_sound.wav » et « 25.wav » dans le dossier « 5.samples » et que je tape ce code :

```
Curs Off : Cls 0 : Paper 0
Locate 0,10 : Centre "Press 1 or 2"
Do
  A$ = Inkey$
  If A$ = "1" Then Sam Play "my_sound"
  If A$ = "2" Then Sam Play 25
  Wait Vbl
Loop
```

En appuyant sur la touche 1 ou 2 du clavier le son est joué. Comme vous le voyez, l'instruction Sam Play joue le son donné en référence. Ici, tout comme pour les images, le son est référencé par son nom de fichier (« my_sound ») ou son numéro (25).

- Tout fichier placé dans un des dossiers de banque, sera automatiquement chargé à l'exécution de votre programme, et disponible immédiatement. Notez qu'il augmente donc la taille de votre programme car il est chargé avec lui.

Les banques de données d'AOZ Studio supportent les formats de fichiers les plus courants :

- Images : PNG, GIF, JPG
- Sons/Musiques : WAV, MP3, OGG

CHARGEMENT A LA DEMANDE

Vous venez de comprendre que les banques de données avec les fichiers qui sont dans les répertoires "resources" de votre application sont très pratiques car vous n'avez pas à vous soucier du chargement des fichiers.

AOZ Studio propose également une autre méthode, de chargement à la demande, qui permet également de charger un fichier « à la volé », à un certain moment de votre programme. Dans ce cas l'image, le son ne sont pas préchargés avec votre programme comme avec les banques mémoires et donc la taille du programme en est d'autant réduite.
Pour cela, nous allons parler des "Assets".

Assets

Un « asset », en français une ressource, est un fichier et que vous pouvez charger depuis votre programme. Les fichiers utilisés comme assets doivent être placés dans le dossier « resources/assets », ou dans un dossier spécifié par vous : voir le chapitre : Dossier par défaut.

Vous pouvez organiser ce dossier comme vous le souhaitez, en rangeant les fichiers par catégories (images, audio, …).

« C'est pareil que les banques de données alors ? »
Oui, mais à la différence des banques de données, les Assets ne sont pas chargés automatiquement à l'exécution de votre programme, mais seulement lorsque vous le décidez dans votre programme. L'exemple ci-dessous charge et affiche l'image «image1.png» si il se trouve dans le dossier resources/assets.

```
Cls 0
Load Asset "image1.png", "img1"
Actor 1, Image$="img1"
```

Dans cet exemple **Load Asset** charge l'image « image1.png » depuis le dossier « assets », et lui donne la référence « img1 ». L'image est alors disponible pour le programme.

Lorsque vous appelez l'instruction **Load Asset**, votre programme est mis « en pause » pendant le chargement du fichier et reprend son court lorsque le fichier est chargé. Cela permet de garantir que le fichier sera totalement disponible pour la suite.

Pour les animations (spritesheet), sons ou la musique, vous pouvez également utiliser **Load Asset** pour charger ces fichiers « à la demande » :

```
Load Asset "music.mp3", 1
Load Asset "sfx.wav", 2
Load Asset "sprites/graphics.sprite", "graphics"
Curs Off : Cls 0: Paper 0
Locate 0,10 : Centre "Appuyez sur d ou f"
Do
  A$ = Inkey$
  If A$ = "d" Then Play Audio 1
  If A$ = "f" Then Play Audio 2
  Wait Vbl
Loop
```

Dans cet exemple **Load Asset** charge les fichiers « music.mp3 » et « sfx.wav » avec les noms 1 et 2. Lorsque l'on appuie sur la touche « d » du clavier, le son 1 est joué. Si l'on appuie sur la touche « f », c'est le son 2.

NOTE : Les sons chargés avec **Load Asset** limite l'utilisation des instructions Sam Play. En effet, les sons seront systématiquement joué sur tous les canaux, et à la fréquence prévue par le fichier.

Load Asset permet donc, comme Actor de définir un nom pour les fichiers utilisés, mais également un numéro. A savoir également, deux assets peuvent porter le même nom ou le même numéro, à la condition qu'ils ne soient pas du même type (ex PNG et JPG). Mais si vous nommez deux images (par exemple) du même nom ou du même numéro, la dernière image chargée remplacera la précédente. Logique.

Ainsi, ce code fonctionnera parfaitement :

```
Load Asset "sfx.wav", 1
Load Asset "img1.png", 1
Curs Off : Cls 0 : Paper 0
Locate 0,10 : Centre "appuyez sur la barre d'espace"
Actor 1, Image$="img1"  // Notre image 1 est affichée
Do
  A$ = Inkey$
  If A$ = " " Then Play Audio 1 : // Notre son 1 est joué
  Wait Vbl
Loop
```

Cet exemple charge deux assets : un son et une image. Tous deux portent le numéro 1. Mais AOZ fait la différence entre les deux et n'indique pas d'erreurs.

Formats de fichiers compatibles

Load Asset peut charger plusieurs formats de fichiers, dont certains ne sont pas supportés par les banques de données.

- Modules de musiques
  - .mod : format Pro-Tracker
  - .xm : format FastTracker
- Styles
  - .css
- Javascript

    - .js
- Modèles de données
    - json
- Images
    - .gif
    - .png
    - .jpg
    - .bmp
    - .svg : Format vectoriel
    - .iff / .ilbm : Format Deluxe Paint
- Audio
    - .mp3
    - .wav
    - .ogg
    - .wma
- Video
    - .mp4
    - .mpeg4
    - .avi
    - .wmv
    - .ogv
    - .webm

Chaque format peut être manipulé à l'aide des commandes AOZ dédiées.

« Mais comment AOZ sait que je charge un son, une image ou une vidéo… ? ». La commande **Load Asset** détecte automatiquement le type du fichier, grâce à son extension. Donc il est important de vous assurer que l'extension de votre fichier fait partie de la liste ci-dessus.

Lorsque vous n'avez plus besoin d'un asset, vous pouvez l'effacer avec l'instruction **Del Asset.**

Dans l'exemple ci-dessous **Del Asset** supprime définitivement l'asset de type « audio » portant le numéro 1.

```
Load Asset "sfx.wav", 1
Wait Key
Play Audio 1 : // Notre son 1 est joué
Wait 2
Del Asset "audio", 1 : // Supprime l'asset
```

La commande Load Asset détecte si un fichier utilise déjà la référence donnée comme argument pour ne pas le charger à nouveau, alors assurez-vous de le supprimer si nécessaire.
Example:

```
Load Asset "asset1.png", 50 // Load l'asset1.png file et l'enregistre sous la référence 50.
Load Asset "asset2.png", 50 // Attention : ce load ne va pas fonctionner car la référence 50 est déjà utilisée par une autre image.
```

Si vous souhaitez mettre à jour un élément déjà chargé, vous devez d'abord supprimer l'élément précédent :

```
Load Asset "asset1.png", 50 // Load l'asset1.png file et l'enregistre sous la référence 50.
Del Asset "image", 50 // Efface l'asset 50
Load Asset "asset2.png", 50 // Load l'asset2.png file et l'enregistre sous la référence 50.
```

Dossier par défaut

Le dossier par défaut dans lequel doivent être placés vos fichiers d'assets est le dossier « resources/assets ». C'est depuis ce dossier que le Transpiler d'AOZ chargera les assets. A cet emplacement vous êtes sur que lorsque vous sauverez votre programme en .AOZIP ou le publierez avec PUBLISH les assets seront bien intégrées et que donc ne manqueront pas.

Si vous désirez utiliser des assets, par exemple une image image1.png qui se trouve dans un autre répertoire c'est possible de la façon suivante :

1. utilisez le tag **#useAssetsResources** en le mettant à False. Ainsi, vous pourrez charger un fichier asset depuis un dossier différent de celui par défaut.

2. Précisez le chemin dans le LoadAsset par ex si l'image1.png est dans le répertoire C:/image

Exemple :
#useAssetsResources: False
Load Asset c:/image/"image1.png", "img1"

Bien, bonne chose de faite n'est-ce pas ? Dans le doute faites des essais. Une fois cette partie maitrisée c'est vous le boss.

20. MAGIE AUDIO

Effets sonores

Les instructions audio d'AOZ agissent indépendamment, de sorte qu'ils n'interféreront pas avec votre programmation. Au contraire, l'audio peut améliorer votre travail comme vous le souhaitez, pour agir comme un marqueur, ajouter du réalisme, apaiser, choquer ou injecter des émotions. Commençons par sonner quelques cloches musicales, avec 96 cloches, où Bell. La cloche numéro 1 (Bell 1) joue la note la plus basse jusqu'à Bell 96 pour la note la plus élevée. Exécutez ce programme, puis pressez une touche pour vous faire sonner les cloches :

```
For RING=1 To 96
Bell RING : Wait Key
Next RING
```

Voici d'autres effets à taper et RUN :

```
Boom : Print "It was a dark and stormy night."
Wait 2
Bell 1 : Print "Count Dracula met a bat."
Wait 2
Shoot : Print "Ouch!"
```

Il existe des commandes AOZ pour contrôler le volume, les canaux audio, les motifs d'ondes, le bruit blanc, les échantillons sonores, les modules de musique et bien plus encore, de sorte que vous ne soyez pas limité à sonner les cloches. En fait, vous pouvez imaginer n'importe quel effet sonore ou audio. Regardez plus tard l'instruction **Sam Play** pour produire des sons aussi facilement que possible. Note : il y aura plus à venir dans les futures versions d'AOZ Studio.

Il faut qu'on parle

Synthèse vocale

Vous allez demander à AOZ de parler maintenant. Tapez cette ligne et RUN :

```
Say "It was a dark, and stormy night."
```

Nous pensons que c'est une voix attrayante ? Elle dira tout ce que vous tapez en anglais.

Elle peut aussi parler en Français :

Speech Language "fr"
Say "bonjour comment allez-vous"

Allons maintenant plus loin avec la synthèse vocale :

Say "texte", wait
Demande à AOZ de dire votre texte avec la simple instruction **Say**, suivie du texte. Le paramètre **wait** est une valeur numérique (définie à 0 par défaut) et qui fixe le temps de pause après avoir dit le texte.

Notez que vous avez 2 syntaxes:
Say "Bonjour je suis américaine" ou
Say sentence$= "Bonjour je suis américaine"

Avant de faire dire (**Say**) quelque chose à votre machine, vous pouvez aussi modifier les paramètres :

Set Talk sex, fashion, pitch#, rate
Défini les paramètres de l'instruction Say :

- sex : 0 pour les femmes, ou 1 pour les hommes. Par défaut, ce paramètre est défini à 0.
- mode : non utilisé
- pitch#: Défini la hauteur de la voix entre 0.0 et 2.0 en mode AOZ, (ou 65 à 320 en mode Amiga AMOS).
- rate: Défini la vitesse de la voix entre 0,0 et 10,0 en AOZ, (40 à 400 en mode Amiga AMOS).

Exemple :
Set Talk sex=1, rate=0.5
Say "AOZ Studio is amazing!"

Talk Misc volume#, frequency
Équivalent de l'instruction **Voice Volume**. Le paramètre de fréquence n'est pas utilisé (pour la compatibilité AMOS).

Talk Stop
Arrêtez la dictée du texte écrit.

Set Voice voice
Définissez la voix, parmi celles disponibles pour dire le texte.

Voice pitch#
Définissez la hauteur de la voix, de basse profonde à soprano, pour dire votre texte écrit. Le réglage de la hauteur se situe entre 0.0 et 10.0 en AOZ (de 65 à 320 en AMOS).

Voice Volume Volume#
Définissez la vitesse de diction. La valeur se situe entre 0.0 et 10.0 en AOZ (de 40 à 400 en Amiga AMOS).

Par exemple que l'on peut tester ainsi :

```
Do
 Voice Volume Volume#
 Say "quick"
 Volume#=Volume#+0.2
Loop
```

Reconnaissance vocale

Important à savoir :
-L'enregistrement de l'audio depuis le micro de votre PC, smartphone,… fonctionne uniquement dans un navigateur Web, pas dans la visionneuse AOZ (F2).
Assurez-vous bien que l'option microphone de votre navigateur Web est active et exécutez le programme avec la touche F1 (Exécuté dans un navigateur).

-Afin que le son fonctionne dans un navigateur il faut cliquer dans la fenêtre, c'est une protection des navigateurs.

Lançons la reconnaissance vocale !

Voici un exemple utilisant cette instruction :

```
Speech Recognition Start
Print ">Start to say something"
Do
  TXT$ = Speech Recognition Value$
  If TXT$ <> "" Then Print TXT$
Loop
```

Faites RUN avec la touche F1 et parlez en Anglais

Speech Language
Défini la langue parlée par l'utilisateur. La valeur est un code de pays. Par défaut, il est défini en Anglais « en », « fr » c'est pour Français, comme ceci :

```
Speech Language "fr"
Speech Recognition Start
Print "dis moi bonjour"
Do
 TXT$ = Speech Recognition Value$
 If TXT$ <>"" Then Print TXT$
Loop
```

Faites RUN (touche F1) et parlez Français

Speech Recognition Stop
Arrête la reconnaissance vocale

Speech Recognition Reset
Réinitialise tous les paramètres de la reconnaissance vocale.

Paroles, paroles

Voici d'autres fonction utiles.

=Speech Synthesis Allowed
Retourne vrai si la synthèse vocale est autorisé par le système.

=Speech Recognition Allowed
Retourne Vrai si la reconnaissance vocale est autorisée.

Vous pouvez par exemple tester comme cela :

```
If Speech Synthesis Allowed
  Print "Synthèse vocale autorisée"
Else
  Print "Synthèse vocale NON autorisée"
End If

If Speech Recognition Allowed
  Print "Reconnaissance vocale autorisée"
Else
  Print " Reconnaissance vocale NON autorisée "
End If


Do
  TXT$ = Speech Recognition Value$
  If TXT$ <> "" Then Print TXT$ : End
  Wait Vbl
Loop
=Voice Count
If Voice Count = 0
  Print "No voices found on this system!"
Else
  Print "There are voices available."
End If
```

***Speech Recognition Add Word**, word$ (pas encore utilisé)*

Il y a des exemples dans le dossier Démos. Essayez par ex :

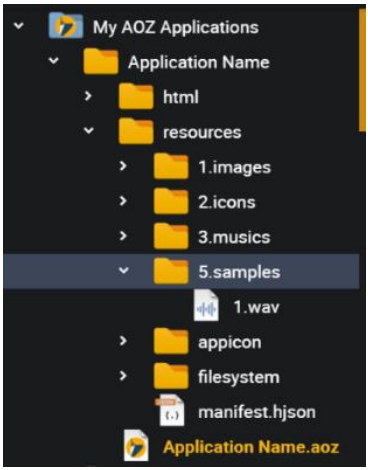
Samples

La plupart des magiciens voudront exploiter des échantillons audio ou samples dans leurs jeux et autres applications. Les samples sont utilisés pour des tas de raisons, pour de l'ambiance atmosphérique, des effets sonores, tout comme dans un film. Avec AOZ vous pouvez les jouer dans n'importe quel ordre, séparément ou simultanément, à n'importe quel volume, et avec des fondus, des croisements et des effets spéciaux.

L'instruction pour jouer (Play) un Sample d'audio est… **Sam Play.** Nous avons hésité avec AltCtrlxx1 mais suite à de longues discussions avec l'équipe nous avons finalement choisi **Sam Play**.

Comme toujours, lorsque vous créez une application, vos fichiers audio seront conservés dans un dossier de sons prêts à l'emploi. Il y a un dossier distinct avec application, alors passons à une vue d'ensemble de ce sujet.

Vous pouvez copier/coller des sons, de la musique, comme des images dans le dossier resources correspondant de votre application en y déplaçant le fichier. Vous pouvez copier un fichier en le faisant glisser dans le dossier pendant que la touche Ctrl est pressée.



Sam Play 1 jouera le son nommé 1 dans le dossier 5.samples ->

Pour les sons et la musique, il est possible de gérer le volume, de jouer en boucle, de connaître la position de lecture actuelle. Voir : Load Asset, Audio Loop On/Off, Play Audio, Stop Audio, Volume Audio).

Une autre méthode consiste à charger le fichier avec l'instruction Load Asset, voici un exemple, sachant que votre fichier doit se trouver dans ce répertoire sounds que vous aurez préalablement créé :

```
Load Asset " resources/sounds/my_audio.wav", "sfx"
Wait Key
Play Audio "sfx"
Pause Audio "sfx"
TM = Time Audio("sfx")
```

ATTENTION : Pour jouer un son le navigateur internet demande une action préalable de l'utilisateur (protection des navigateurs). Ainsi il faut garder l'affichage du SplashScreen d'AOZ Studio ou le remplacer par quelque chose qui de la même manière demande de cliquer préalablement.

Music

Vous avez une gamme complète d'instructions et d'outils pour jouer de la musique avec AOZ Studio, que vous pouvez par exemple créer avec des outils comme Med Soundstudio et Tracker. Si vous ne pouvez pas écrire vos propres chefs-d'œuvre musicaux, ne vous inquiétez pas. AOZ vous permet de prendre les bandes sonores musicales d'un autre compositeur et de les ajouter à vos propres jeux et utilitaires. Il y a des dizaines de milliers de bandes sonores du domaine public écrit avec des systèmes comme SoundTracker, NoiseTracker et toutes les commandes pour les charger et les utiliser sont intégrés à AOZ.

Et il y a dans l'AOZ Drive la belle bibliothèque de samples créés par Miko que nous fournissons libre de droits pour (uniquement) vos applications AOZ.

AOZ prend en charge les formats suivants : wav, ogg, mp3, mod, xm.

21. LA SOURIS

UTILISATION DE LA SOURIS

Les souris, trackball et autres appareils similaires aiment AOZ. Nous en avons déjà discuté dans le chapitre d'introduction sur Actor, en plus de donner la position du pointeur souris et l'état des boutons, nous pouvons également changer la forme du pointeur de souris, limiter la zone de l'affichage dans lequel il est actif et plus encore.

Forme du pointeur souris

Description : Cette instruction **Change Mouse** définit la forme du pointeur de la souris.

Paramètre : Le paramètre entier (N) définit la forme du pointeur comme suit :

N°	Forme pointeur
1	Pointeur
2	Croix
3	Occupé
4	Doigt

Exemple:

```
For M=1 To 4
 Print M : Change Mouse M
 Wait Key ' la forme change sur chaque appuis de touche.
Next M
```

NOTE: Sur l'Amiga, une forme de souris > 3 prendra la forme de la banque Sprite.

Cacher le pointeur souris

Description : Cette instruction **Hide** / **Show** rend le pointeur de souris invisible. Chaque fois que **Hide** est appelé un compteur est incrémenté. Le pointeur de souris restera caché jusqu'à ce que le compteur soit à nouveau égal à 0.

REMARQUE : Bien que **Hide** rende le pointeur invisible, il est toujours actif, et sa position peut être suivie et lue.

Le compteur permet de simplifier la façon de garder une trace de qui a caché la souris, et quand. Si vous avez plusieurs routines cachant ou montrant le pointeur de souris cela peut être lourd à gérer. Le compteur suit cela automatiquement.

Exemple :

```
hides=0 'simule le compteur interne pour comprendre le fonctionnement. Appuyez sur les touches H ou S.
Curs Off 'cache le curseur
Do
 I$=Upper$ (Inkey$)
 If Mouse Key=1 Or I$="H" Then Hide : Inc hides : Repeat Until Mouse Key=0
 If Mouse Key=2 Or I$="S" Then Show : Dec hides : Repeat Until Mouse Key=0
 Locate 0,0 : Print Using "-##";hides
 Wait Vbl
Loop
```

Montrer le curseur souris

Description *:* Cette instruction **Show** rend le pointeur de souris visible. Chaque fois que **Show** est appelé cela décrémente le compteur Hide. Le pointeur de souris devient visible lorsque le nombre de Show est égale au nombre de Hide. (autrement dit lorsque le compteur Hide devient 0)

Cachez tout

Description *:* Cette instruction **Hide On** rend le pointeur de souris invisible tout le temps. Il ignore le compteur interne utilisé par les instructions **Hide** et **Show**.

REMARQUE : Bien que **Hide On** rende le pointeur invisible tout le temps, il est toujours actif, et sa position peut être suivie et lue.

Montrez tout

Description *:* Cette instruction **Show On** rend le pointeur de souris visible tout le temps. Il ignore le compteur interne utilisé par les instructions **Hide** et **Show**.

Exemple :

```
hides=0 'simule le compteur interne pour comprendre le fonctionnement. Appuye
z sur H ou la barre d'espace.
Curs Off 'cache le curseur
Do
   I$=Upper$ (Inkey$)
   If Mouse Key=1 Or I$="H" Then Hide : Inc hides : Repeat Until Mouse Key=0
   If Mouse Key=2 Or I$=" " Then Show On : Repeat Until Mouse Key=0 'Force à êtr
e toujours visible
   Locate 0,0 : Print Using "-##";hides
   Wait Vbl
Loop
```

Limit Mouse X1,Y2 To X2,Y2

Description *:* Cette instruction limite le mouvement du pointeur de souris à une zone rectangulaire spécifiée de l'écran actuel.

Paramètres : Les paramètres (entiers) représentent les coins supérieur gauche et inférieur droit de la zone rectangulaire définie. Le pointeur de souris ne sera pas affiché en dehors de cette zone définie.
X1 et Y1 sont les coordonnées X et Y du coin supérieur gauche.
X2 et Y2 sont les coordonnées X et Y du coin inférieur droit.

Exemple :

```
Screen Open 0,800,600,32,Lowres
Limit Mouse 0,0 To 400,300
Do
  Wait Vbl
Loop
```

L'exemple ci-dessus limitera le mouvement de la souris à la zone spécifiée.

REMARQUE : Lorsque **Limit Mouse** est appelée sans paramètre, elle restaure la zone normale de la souris, lui permettant de se déplacer autour de l'écran entier. Exemple :

= Mouse Click

Description *:* Cette fonction renvoie l'information sur les boutons de souris cliqués (un seul clic). Le résultat est comme suit :

Bit	And	Description
0	$01	Bouton gauche
1	$02	Bouton droit
2	$04	Bouton du milieu (s'il existe)

Valeur de retour : Le résultat est un entier qui indique chaque position de bit représentant l'un des boutons de la souris comme décrit ci-dessus.

Exemple :

```
Do
  Btns$=""
  CLK = Mouse Click
  Locate 0,I :
  If (CLK & 1) = 1 Then Btns$ = Btns$+"Left": I=I+1
  If (CLK & 2) = 2 Then Btns$ = Btns$+"Right": I=I+1
  If (CLK & 4) = 4 Then Btns$ = Btns$+"Middle": I=I+1
  Print Bin$(CLK,8);" ";Btns$
  Wait Vbl
Loop
```

REMARQUE : Une fois qu'un bouton a été appuyé, l'état «cliqué » ne s'activera plus tant que le bouton de la souris n'aura pas été relâché, puis cliqué à nouveau.

= Mouse Key (Bouton)

Description *:* Cette fonction renvoie l'état actuel des boutons de souris. Ce résultat est comme suit :

Bit	And	Description
0	$01	Bouton gauche
1	$02	Bouton droit
2	$04	Bouton du milieu (s'il existe)

Valeur de retour : Le résultat est un entier qui indique chaque position de bit représentant l'un des boutons de la souris comme décrit ci-dessus.

= Mouse Wheel

Description: Cette fonction renvoie l'état actuel de la roue de la souris. L'état est donné comme suit :

Résultat	État de la roue de la souris
0	Neutre
-1	Roue déplacée vers le haut
1	Roue déplacée vers le bas

Valeur de retour : Le résultat est un entier qui indique si la roue de la souris a été déplacée vers le haut ou vers le bas, ou si elle reste au neutre.

Exemple:

```
PrevMW=0
Do
        MW = Mouse Wheel
        If MW <> PrevMW Then Print Using "-#";MW
        PrevMW = MW
Loop
```

L'exemple ci-dessus vous permet de voir les changements dans l'état de la roue de la souris avec la fonction Mouse Wheel.

= Mouse Zone

Description *:* Cette fonction renvoie le numéro de la zone actuellement sous le pointeur de souris.

REMARQUE : Une zone est simplement un espace rectangulaire défini de l'écran, réservée à la détection de la position de collision ou de souris. (Voir aussi *HZone*)

Valeur de retour: Un entier dont le numéro spécifie la zone sous les coordonnées spécifiées. S'il n'y a pas de zone sous ces coordonnées, 0 sera retourné.

Exemple :

```
Palette 0,$FFFFFF,$FF0000,$00FF00,$0000FF,$FFFF00,$00FFFF,
$FF00FF
Cls 0
Reserve Zone 5
MakeZone[1,20,0,100,100]
MakeZone[2,101,0,200,100]
MakeZone[3,20,101,200,200]
MakeZone[4,201,0,799,599]
MakeZone[5,20,201,200,599]
```

```
Do
        Locate 20,5 : Print Using "-###";Mouse Zone
        Wait Vbl
Loop

Procedure MakeZone[zn,x1,y1,x2,y2]
        Ink zn+1 : Box x1,y1 To x2,y2
        Set Zone zn,x1,y1 TO x2,y2
End Procedure
```

Cet exemple introduit de nouveaux concepts, si cela semble très compliqué pas de panique, passez à la suite.

= Mouse Screen

Description : Cette fonction renvoie le numéro d'écran sous le pointeur de la souris. *(Voir aussi Scln(X,Y)*

Valeur de retour : Un entier qui est le numéro d'écran sous le pointeur de la souris. S'il n'y a pas d'écran sous ces coordonnées, un nombre négatif sera retourné.

Exemple *:*

```
Global scrWidth,scrHeight
scrWidth=360 : scrHeight=220
#displayWidth: 1080
#displayHeight: 660

ScreenSample[1,0,0]
ScreenSample[2,scrWidth,scrHeight]
ScreenSample[3,scrWidth*2,scrHeight*2]
ScreenSample[4,0,scrHeight*2]
ScreenSample[5,scrWidth*2,0]

Do
        Locate 1,1 : Print Using "##"; Mouse Screen
```

```
        Locate 1,2 : Print Using "-####"; X Mouse;" , "; : Print Using "-
####";Y Mouse;" "
        Wait Vbl
Loop

Procedure ScreenSample[n,xPos,yPos]
 Screen Open n,scrWidth,scrHeight,32,Lowres
 Screen Display n,xPos,yPos
Palette 0, $FFFFFF, $FF0000, $00FF00, $0000FF, $FFFF00, $00FFFF,
$FF00FF
 Curs off : Ink n+1 : Bar 0,0 To scrWidth-1,scrHeight-1
End Procedure
```

Cet exemple introduit de nouveaux concepts, si cela semble très compliqué pas de panique, passez à la suite.

= Scln (X,Y)

Description : Cette fonction retourne le numéro d'écran aux coordonnées matérielles spécifiées. Généralement utilisé avec **X Mouse** et **Y Mouse** pour déterminer quand le pointeur de souris est entré dans un écran particulier.

Paramètres *: Les* paramètres entiers **X** et **Y** indiquent les coordonnées matérielles que nous cherchons.

Valeur de retour : est un entier qui est le numéro d'écran sous les coordonnées spécifiées. S'il n'y a pas d'écran sous ces coordonnées, un nombre négatif sera retourné.

Exemple:

```
Screen Open 0,800,600,32,Lowres
#displayWidth: 800
#displayHeight: 600

SetPalette : Screen Display 0,0,0 : ClearIt
Ink 4 : BigX

S=1
Screen Open S,800,600,32,Lowres
SetPalette : Screen Display S,0,0 : ClearIt
Ink 2 : BigX

S=2
Screen Open S,800,600,32,Lowres
SetPalette : Screen Display S,1920-800,0 : ClearIt
Ink 3 : BigX

S=3
Screen Open S,1920-1600,1080-600-1,32,Lowres
SetPalette : Screen Display S,800,601 : ClearIt
Ink 5 : BigX

Screen 1 : Pen 1 ' Screen number display to screen 1
Do
  Locate 1,1 : Print Using "-#";Scln(X Mouse,Y Mouse)
  Wait Vbl
Loop

Procedure SetPalette
  ' Défini la palette pour cet écran.
  Palette 0,$FFFFFF,$FF0000,$00FF00,$0000FF,$FFFF00,$00FFFF,$FF00FF
End Procedure

Procedure ClearIt
  Pen 1 : Paper 0 : Cls 0
End Procedure

Procedure BigX
  ' Dessine boites et grosses croix sur la totalité de l'écran
  Box 0,0 To Screen Width-1,Screen Height-1
  Draw 0,0 To Screen Width-1,Screen Height-1
  Draw Screen Width-1,0 To 0,Screen Height-1
End Procedure
```

Essayez de déplacer la souris sur les différentes zones d'écran. Vous verrez le numéro d'écran apparaître et un nombre négatif lorsqu'il n'y a pas d'écran sous le pointeur de la souris.

= X Hard(X_SCREEN)

Description *:* Cette fonction convertit une coordonnée d'écran horizontale en coordonnée "matérielle" ou dit autrement physique sur l'écran.

REMARQUE : Les coordonnées matérielles sont relatives au coin supérieur gauche de la zone d'affichage par rapport à l'écran actuel. C'est comme les zones overscan d'un moniteur vidéo.
Paramètre *:* L'entier X_SCREEN sera la coordonnée de l'écran à convertir.
Valeur de retour *:* Le résultat sera un entier avec la coordonnée matérielle horizontale par rapport à la zone d'affichage de l'écran actuel.

Exemple:

```
// Défini la taille de l'écran
#displayWidth: 1920
#displayHeight: 1080
Palette 0,$FF0000
Ink 1 : Pen 1 : Paper 0 : Cls 0
Box 0,0 To 1919,1079 ' Affiche les bords (en rouge)
Locate 1,0 : Print "Zone d'affichage"
// Ouvre un écran plus petit.
Screen Open 1,800,600
Screen Display 1, 100,70,, ' Bouge l'écran 0 un peu en bas à droite
Palette 0,$FFFFFF
Ink 1 : Pen 1 : Paper 0 : Cls 0
Box 0,0 To 799,599 ' Affiche les bords de l'écran 1 (en blanc)'
Locate 1,0 : Print "Screen 1"
Locate 1,2 : Print X Hard(0),Y Hard(0)
' Au dessus cela affiche les coordonnées hardware du haut gauche
```

= Y Hard (Y_SCREEN)

Description *:* Cette fonction convertit une coordonnée d'écran verticale en une coordonnée "matérielle" ou dit autrement physique sur l'écran. Cf X Hard(X_SCREEN)

X Souris =

Description *:* Cette variable système peut être utilisée pour définir la position du pointeur de souris avec des coordonnées matérielles.

REMARQUE : Votre navigateur web peut ou non être en mesure de contrôler la position de la souris.

Exemple:

```
#displayWidth: 320
#displayHeight: 200
Screen Open 0,320,200,32,Lowres
X Mouse = X Hard(160) ' Milieu de l'écran horizontalement
Y Mouse = Y Hard(100) ' Milieu de l'écran verticalement
Wait Key
End
```

Ce petit code, si le navigateur est compatible, ouvre un écran de 320 x 200 et positionne la souris au milieu de l'écran.

= X Mouse

Description: Cette fonction renvoie la position horizontale de la souris pour l'écran actuel.

Valeur de retour : Un entier qui est la position horizontale de la souris à l'écran. Le résultat est exprimé à l'aide de coordonnées matérielles.

REMARQUE : Les coordonnées matérielles prennent en compte l'ensemble de la zone d'affichage, y compris la partie (overscan) autour de l'écran.

Exemple *:*

```
Screen Open 1,1280,800,32,Lowres
Screen Display 1,200,50
Locate 1,0 : Print "Coordonnées écran       "
Locate 1,2 : Print "Coordonnées matérielle "
Do
  Locate 22,0 : Print Using "X: -####";X Screen(X Mouse);
  Print Using " Y: -####";Y Screen(Y Mouse)

  Locate 22,2 : Print Using "X: -####";X Mouse;
  Print Using " Y: -####";Y Mouse
  Wait Vbl
Loop
```

= Y Mouse

Description: Cette fonction renvoie la position verticale de la souris pour l'écran actuel. Voir X Mouse.

= X Screen(X_HARD)

Description: Cette fonction convertit une coordonnée matérielle horizontale en une coordonnée d'écran.

REMARQUE : Les coordonnées de l'écran sont relatives au coin supérieur gauche de l'écran actuel.

Paramètres *:* L'entier **X_HARD** est la coordonnée matérielle à convertir. (Les coordonnées matérielles sont relatives à l'ensemble de la zone d'affichage, y compris la partie overscan.)

Valeur de retour : Le résultat est un entier qui est la coordonnée d'écran horizontale, correspondant de sa coordonnée matériel.

Exemple:

```
Ink 1 : Pen 1
Box 1,1 To Screen Width-2,Screen Height-2
Do
  Locate 2,2 : Print Using "X: -####";X Screen(X Mouse);" "
  Locate 2,4 : Print Using "Y: -####";Y Screen(Y Mouse);" "
  Wait Vbl
Loop
```

= Écran Y (Y_HARD)

Description: Cette **fonction** convertit une coordonnée matérielle verticale en une coordonnée d'écran. Cf X Screen(X_HARD)

22. CREER UNE INTERFACE

Lorsque vous allez créer une application vous aurez rapidement besoin de mettre des boutons cliquables, des barres de progression, des boutons radio, réglettes, etc. Ce que nous appelons des Composants.

AOZ Studio vous aidera beaucoup pour cela :
1°) avec la famille des instructions UI et
2°) avec l'outil DESIGNER

Nous allons parler des deux dans ce chapitre.

Chapitre qui n'est pas définitif, car le jeu d'instructions UI évolue, tout comme le DESIGNER.

Les instructions UI

Sont un ensemble d'instructions et de fonctions qui permettent de créer et gérer des composants d'interface utilisateur. Parfait à utiliser si vous souhaitez créer une interface pour votre application.

Le DESIGNER est un outil qui utilise ces instructions UI, donc si vous avez besoin d'exemples, vous pouvez voir le code correspondant généré par le DESIGNER.

Comme avec l'instruction Actor, l'ordre des paramètres dans l'instruction est libre, il suffit qu'ils soient séparés par une virgule , . Seuls les paramètres que vous utilisez peuvent être définis, les autres prendront alors les valeurs par défaut.

Voici la liste des instructions de l'interface utilisateur :

UI Confirm title$, content$, confirmButton$, closeButton$, onConfirm$

Affiche un composant d'interface popup "Confirmer".

Parameters:

- title$: The text for the title of the popup box. If not set, the default text is "Confirm"
- content$: The text for the content of the popup box., If not set, the default content is "Are you sure?"
- confirmButton$: The text to be displayed on the confirm button. If not set, the default text is "Confirm"
- closeButton$: The text to be displayed on the close button. If not set the default text is "Close"
- onConfirm$: The name of an AOZ Procedure to be called if the confirm button is pressed.

UI Progress id$, x, y, value, width, height, class$

Affiche une barre de progression.

Parameters:

- id$: A unique identifier for the component.
- x: The horizontal position on the screen. If not set, 10 by default.
- y: The vertical position on the screen. If not set, 10 by default.

- value: A value of the progress bar between 0 and 100, If not set, 0 by default.
- width: The width of the component in pixels. If not set, 400 by default.
- height: The height of the component in pixels. If not set, 20 by default.
- class$: The CSS (Cascading Style Sheet) class to be used to style the component. A default style is used if not set.

UI Radio id$, x, y, items$, fontSize, value$, onChange$, fontName$, class$, padding

Affiche des boutons radio.

Parameters:

- id$: A unique identifier for the component.
- x: The horizontal position on the screen. If not set, 10 by default.
- y: The vertical position on the screen. If not set, 10 by default.
- items$: A list of value:item pairs to use for the radio buttons. If not set, no radio buttons will be displayed
- fontSize: The size of the font in pixels. If not set, 20 by default.

- value$: The value of the selected radio button. If not set, non are selected.
- onChange$: The name of an AOZ procedure to be called when the selected radio is changed.
- fontName$: The name of a font to be used for the textual part of the radio buttons.
- class$: The CSS (Cascading Style Sheet) class to be used to style the items. A default style is used if not set.
- padding: The padding in pixels between the content and the text block dimensions. If not set, 5 by default.

**UI TextBlock id$, x, y, width, height, content$, fontSize, padding, class$, fontName**$

Affiche un bloc de texte.

Parameters:

- id$: A unique identifier for the component.
- x: The horizontal position on the screen. If not set, 10 by default.
- y: The vertical position on the screen. If not set, 10 by default.
- width: The width of the component in pixels. If not set, 400 by default.
- height: The height of the component in pixels. If not set, the height will expand to fill the content$

- content$: The text to be displayed. If not set, "TextBlock Content" by default.
- fontSize: The size of the font in pixels. If not set, 20 by default.
- padding: The padding in pixels between the content and the text block dimensions. If not set, 5 by default.
- class$: The CSS (Cascading Style Sheet) class to be used to style the component. A default style is used if not set.
- fontName$: The name of a font to be used for the content$

UI ColorPicker id$, x, y, width, height, value$, onChange$, class$

Affiche un composant de choix de couleurs.

Parameters:

- id$: A unique identifier for the component.
- x: The horizontal position on the screen. If not set, 10 by default.
- y: The vertical position on the screen. If not set, 10 by default.
- width: The width of the component in pixels. If not set, 40 by default.

- height: The height of the component in pixels. If not set, 40 by default.
- value$: The hexadecimal color value for the selected color. If not set, #ffffff by default.
- onChange$: The name of an AOZ procedure to be called when the selected color is changed.
- class$: The CSS (Cascading Style Sheet) class to be used to style the component. A default style is used if not set.

UI Slider id$, x, y, width, min, max, step, value, onChange$, class$

Affiche une réglette (slider)

Parameters:

- id$: A unique identifier for the component.
- x: The horizontal position on the screen. If not set, 10 by default.
- y: The vertical position on the screen. If not set, 10 by default.
- width: The width of the component in pixels. If not set, 300 by default.
- min: The minimum value of the slider. If not set, 0 by default.

- max: The maximum value of the slider. If not set, 100 by default.
- step: The value for each step of the slider. If not set, 1 by default.
- value: The value of the slider. If not set, 0 by default.
- onChange$: The name of an AOZ procedure to be called when the slider value changes.
- class$: The CSS (Cascading Style Sheet) class to be used to style the component. A default style is used if not set.

UI Select id$, x, y, width, items$, fontSize, padding, class$, value$, onChange$, fontName$

Affiche un menu déroulant à choix multiple.

Parameters:

- id$: A unique identifier for the component.
- x: The horizontal position on the screen. If not set, 10 by default.
- y: The vertical position on the screen. If not set, 10 by default.
- width: The width of the component in pixels. If not set, 300 by default.

- items$: A list of value:item pairs to use for list of options in the dropdown list. If not set, no options will be available.
- fontSize: The size of the font in pixels. If not set, 20 by default.
- padding: The padding in pixels between the content and the text block dimensions. If not set, 5 by default.
- class$: The CSS (Cascading Style Sheet) class to be used to style the component. A default style is used if not set.
- value$: The value of the selected item. If not set, the first item is selected.
- onChange$: The name of an AOZ procedure to be called when the selected item is changed.
- fontName$: The name of a font to be used for the text.

UI CheckBox id$, x, y, width, height, value$, class$, onChange$

Affiche des cases à cocher.

Parameters:

- id$: A unique identifier for the component.
- x: The horizontal position on the screen. If not set, 10 by default.

- y: The vertical position on the screen. If not set, 10 by default.
- width: The width of the component in pixels. If not set, 30 by default.
- height: The height of the component in pixels. If not set, 30 by default.
- value$: The value of the check box. If not set, false by default.
- class$: The CSS (Cascading Style Sheet) class to be used to style the component. No style is used if not set.
- onChange$: The name of an AOZ procedure to be called when the selected item is changed.

UI TextBox id$, value$, placeHolder$, class$, x, y, width, fontSize, padding, type$, fontName$, min, max, onChange$

Affiche une zone de texte.

Parameters:

- id$: A unique identifier for the component.
- value$: The text inside the text box. Empty by default.
- placeHolder$: The place holder text. Empty by default.
- class$: The CSS (Cascading Style Sheet) class to be used to style the component. No style is used if not set.

- x: The horizontal position on the screen. If not set, 10 by default.
- y: The vertical position on the screen. If not set, 10 by default.
- width: The width of the component in pixels. If not set, 300 by default.
- fontSize: The size of the font in pixels. If not set, 20 by default.
- padding: The padding in pixels between the content and the text block dimensions. If not set, 5 by default.
- type$: The expected type of text to be entered. If not set, "text" by default.
- fontName$: The name of a font to be used for the text.
- min: Sets the minimum value when type$="number"
- max: Sets the maximum value when type$="number"
- onChange$: The name of an AOZ procedure to be called when the selected item is changed.

UI Button id$, content$, class$, x, y, width, height, fontSize, padding, onClick$, iconClass$, fontName$, tooltip$, tooltipPlacement$

Affiche un bouton cliquable ou non

Parameters:

- id$: A unique identifier for the component.

- content$: The text to display on the button. If not set, "button" by default.
- class$: The CSS (Cascading Style Sheet) class to be used to style the component. No style is used if not set.
- x: The horizontal position on the screen. If not set, 10 by default.
- y: The vertical position on the screen. If not set, 10 by default.
- width: The width of the component in pixels. If not set, the width will auto grow to fit the content.
- height: The height of the component in pixels. If not set, the height will auto grow to fit the content.
- fontSize: The size of the font in pixels. If not set, 20 by default.
- padding: The padding in pixels between the content and the text block dimensions. If not set, 5 by default.
- onClick$: The name of an AOZ procedure to be called when the button is pressed.
- iconClass$: The name of a CSS (Cascading Style Sheet) class used to add an icon to the button.
- fontName$: The name of a font to be used for the content$ text.
- tooltip$: The text to display as a tooltip popup.
- tooltipPlacement$: The position of the tooltip popup. If not set, "auto" by default.

UI TextArea id$, value$, placeHolder$, class$, x, y, width, rows, fontSize, padding, fontName$

Affiche un écran de texte.

Parameters:

- id$: A unique identifier for the component.
- value$: The text to display in the component. Empty by default.
- placeHolder$: The place holder text. Empty by default.
- class$: The CSS (Cascading Style Sheet) class to be used to style the component. No style is used if not set.
- x: The horizontal position on the screen. If not set, 10 by default.
- y: The vertical position on the screen. If not set, 10 by default.
- width: The width of the component in pixels. If not set, 300 by default.
- rows: The number of rows for the component. If not set, 2 by default.
- fontSize: The size of the font in pixels. If not set, 20 by default.
- padding: The padding in pixels between the content and the text block dimensions. If not set, 5 by default.
- fontName$: The name of a font to be used for the content$ text.

UI Popup id$, content$, placement$, delay, class$

Affiche un popup.

Parameters:

- id$: A unique identifier for the component.
- content$: The text to be displayed in the popup. Empty by default.
- placement$: The screen location for the popup. If not set, "top-center" by default.
- delay: The number of milliseconds before the popup is hidden from the screen. If not set, 5 by default.
- class$: The CSS (Cascading Style Sheet) class to be used to style the component. If not set, a default style is used.

UI Cls

Efface tous les composants de la page (plus visibles).

UI Delete id$

Détruit tous les composants de la page (n'existent plus).

Parameters:

- id$: The unique identifier for the component

UI Show id$

Affiche le composant.

Parameters:

- id$: The unique identifier for the component

UI Hide id$

Cache le composant.

Parameters:

- id$: The unique identifier for the component

UI Value id$, value$

Met à jour le composant avec une nouvelle valeur.

Parameters:

- id$: The unique identifier for the component you want to set its value.
- value$: The value used to update the component.

UI Value$(id$)

Fonction qui renvoi la valeur d'un composant

Parameters:

- id$: The unique identifier for the component you want to return its value.

Value returned: string: The value of the UI component.

UI Property$ id$, propertyName$

Fonction qui renvoi la valeur d'une propriété d'un composant

Parameters:

- id$: The unique identifier of an existing UI component
- propertyName$: The name of a UI Component property

UI X id$

Fonction qui renvoi la valeur de X du composant.

Parameters:

- id$: The unique identifier of an existing UI component

UI Y id$

Fonction qui renvoi la valeur de Y du composant.

Parameters:

- id$: The unique identifier of an existing UI component

UI Width id$

Fonction qui renvoi la valeur de Width du composant.

Parameters:

- id$: The unique identifier of an existing UI component

UI Height id$

Fonction qui renvoi la valeur de Height du composant.

Parameters:

- id$: The unique identifier of an existing UI component

UI FontSize id$

Fonction qui renvoi la valeur de la FontSize du composant.

Parameters:

- id$: The unique identifier of an existing UI component

UI FontName$ id$

Fonction qui renvoi le 'fontName$' du composant

Parameters:

- id$: The unique identifier of an existing UI component

UI Padding id$

Fonction qui renvoi la valeur du Padding (espacement de police) du composant.

Parameters:

- id$: The unique identifier of an existing UI component

UI Content$ id$

Fonction qui renvoi la valeur de 'content$' du composant.

Parameters:

- id$: The unique identifier of an existing UI component

UI Min id$

Fonction qui renvoi la valeur Min du composant.

Parameters:

- id$: The unique identifier of an existing UI component

UI Max id$

Fonction qui renvoi la valeur Max du composant.

Parameters:

- id$: The unique identifier of an existing UI component

UI Step id$

Fonction qui renvoi la valeur de Step du composant.

Parameters:

- id$: The unique identifier of an existing UI component

UI Rows id$

Fonction qui renvoi la valeur Row du composant.

Parameters:

- id$: The unique identifier of an existing UI component

UI Placeholder$ id$

Fonction qui renvoi la valeur 'placeHolder' du composant.

Parameters:

- id$: The unique identifier of an existing UI component

UI Tooltip$ id$

Fonction qui renvoi la valeur 'tooltip' du composant.

Parameters:

- id$: The unique identifier of an existing UI component

UI Tooltip$ id$

Fonction qui renvoi la valeur 'tooltip$' du composant.

Parameters:

- id$: The unique identifier of an existing UI component

UI IconClass$ id$

Fonction qui renvoi la valeur 'iconClass$' du composant.

Parameters:

- id$: The unique identifier of an existing UI component

UI Class$ id$

Fonction qui renvoi la valeur 'Class$' du composant.

Parameters:

- id$: The unique identifier of an existing UI component

UI Type$ id$

Fonction qui renvoi la valeur 'Type$' du composant.

Parameters:

- id$: The unique identifier of an existing UI component

UI OnChange$ id$

Fonction qui renvoi la valeur 'onChange$' du composant.

Parameters:

- id$: The unique identifier of an existing UI component

UI OnClick$ id$

Fonction qui renvoi la valeur 'onClick$' du composant.

Parameters:

- id$: The unique identifier of an existing UI component

UI Items$ id$

Fonction qui renvoi la valeur 'Item$' du composant.

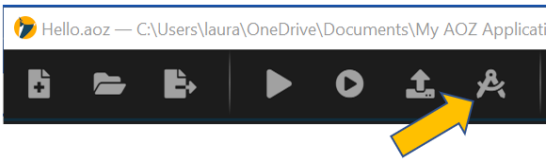
Parameters:

- id$: The unique identifier of an existing UI component

L'AOZ Designer

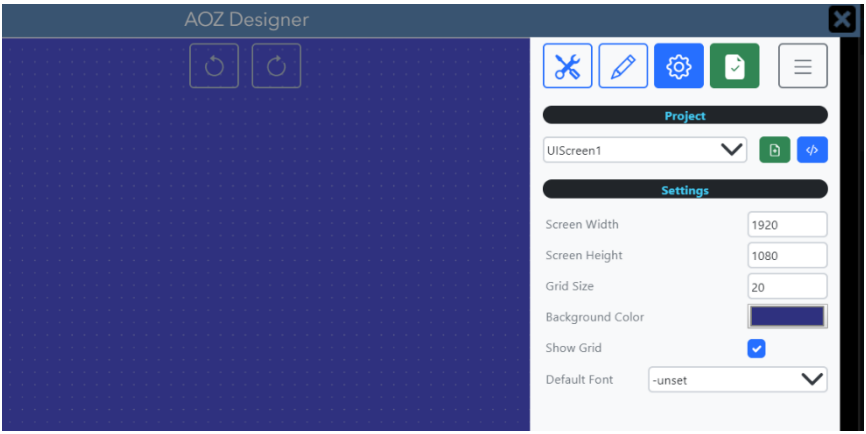
Suivez-moi pas à pas… :

1°) Lancez le DESIGNER en cliquant sur ce bouton.



2°) Découvrez le DESIGNER. Une fois que vous le maîtrisez et que vous souhaitez créer votre interface :

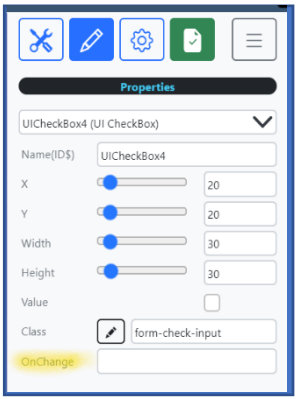
3°) Cliquez sur le bouton "Settings" :



Vous pouvez y créer différentes pages sous la barre noire "Project", y choisir la résolution, la couleur d'arrière-plan, la police par défaut, etc. sous la barre noire "Settings".

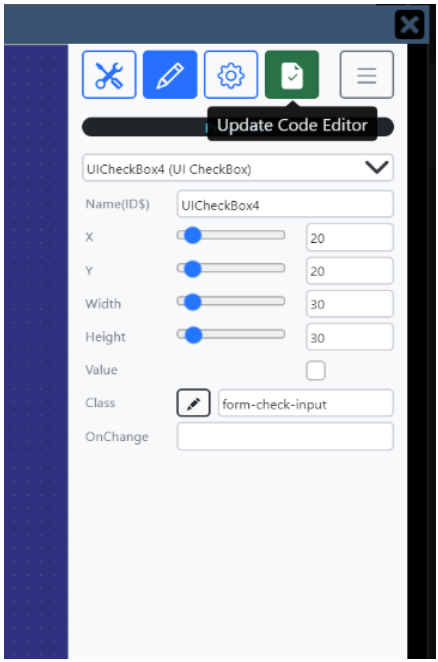
4°) Puis cliquez sur le 1er bouton : "Components", pour créer le design de votre (vos) page(s) grâce aux différents composants, modifiez les paramètres...

Remarque : il y a des paramètres spécifiques en bas de la page Propriétés : "OneCLick" et "OnChange", ici vous donnerez le nom de la procédure qui sera appelée dans votre programme AOZ une fois l'événement survenu.
Exemple: Une case à cocher (CheckBox) est cliquée.



5°) Finalement cliquez sur le bouton Update Code Editor :

Cela générera automatiquement le code AOZ correspondant dans la fenêtre source de votre programme. Code que vous pouvez transpiler avec les boutons RUN.



Remarque : vous pouvez modifier le code généré par le DESIGNER dans votre programme et plus tard revenir dans le DESIGNER, il reconnaîtra son code permettant de le modifier à tout moment depuis le DESIGNER ou depuis l'éditeur AOZ.

23. L'AOZ DEBUGGER

Qu'est-ce qu'un débogueur (ou debugger en Anglais) ?

C'est un outil intéressant qui permet d'aider le programmeur à trouver et comprendre les erreurs dans le code.

Le débogueur d'AOZ Studio est très polyvalent et permet de :

- mettre des points d'arrêts et de parcourir le code
- s'introduire dans le code et surveiller une expression
- inspecter et modifier les variables et les tableaux
- afficher des informations dans une fenêtre de journal
- inspecter tous les sprites, bobs et Actors de votre application
- localiser les éléments graphiques avec précision grâce à des règles et une grille
- exécuter l'application étape par étape, au ralenti ou à pleine vitesse
- regarder l'évolution d'une expression en temps réel

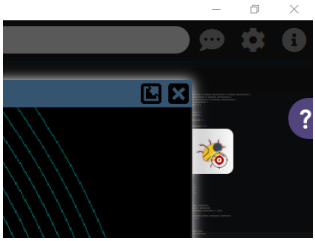
Note avancée : Le débogueur AOZ est intégré par le transpiler dans le code Javascript de votre application. Il ne dépend pas de l'IDE d'AOZ et fonctionne parfaitement dans le navigateur.

Comment afficher le debugger?

Vous pouvez afficher le debugger AOZ de deux manières :

Dans la visionneuse AOZ

(AOZ Viewer) en cliquant sur le bouton sur le côté droit :



Un clic sur ce bouton ouvre le debugger qui n'interféra avec votre application : il s'affiche par-dessus dans un écran transparent qui n'interagit en aucune façon avec l'application, "il regarde ce qui se passe". Tous les mouvements des touches, du joystick et de la souris, à l'exception de la touche « Escape » sont transmis à l'application.

Avec le debugger la fenêtre « Variables » affiche en temps réel toutes les valeurs de toutes les variables mises à jour de votre programme. (Note : certaines valeurs peuvent changer si rapidement qu'elles peuvent être impossibles à lire).

Lorsque vous passez la souris sur le nom d'un tableau, le debugger affiche aussi le contenu du tableau mis à jour en temps réel.

Vous pouvez entrer dans le code à tout moment en appuyant sur **F8** ou en cliquant sur le bouton "**Pause**".

Pour masquer le debugger tout en le gardant en mémoire, appuyez sur "**Échap**". Le débogueur restera à l'arrière prêt à apparaître lorsque vous appuyez une autre fois sur "**Échap**".

Veuillez noter que dans les futures versions le debugger, lorsqu'il est masqué, pourra faire des choses intéressantes comme évaluer la vitesse des différentes routines (ce que les professionnels appellent "profiler") ou enregistrer les valeurs de toutes les variables et objets graphiques afin que vous puissiez rejouer et prendre le relais juste avant le bug.

Pour masquer le debugger et le supprimer complètement de la mémoire, cliquez une deuxième fois sur le bouton « Debugger ».

Lorsqu'il est exécuté depuis l'AOZ Viewer, le debugger vous permet de taper des commandes dans la fenêtre de la console, comme si vous étiez en mode direct. Utilisez cette fonctionnalité très intéressante pour modifier la position des objets, changer les valeurs des variables ou manipuler des données, puis relancez l'application pour voir l'effet.

Depuis votre navigateur

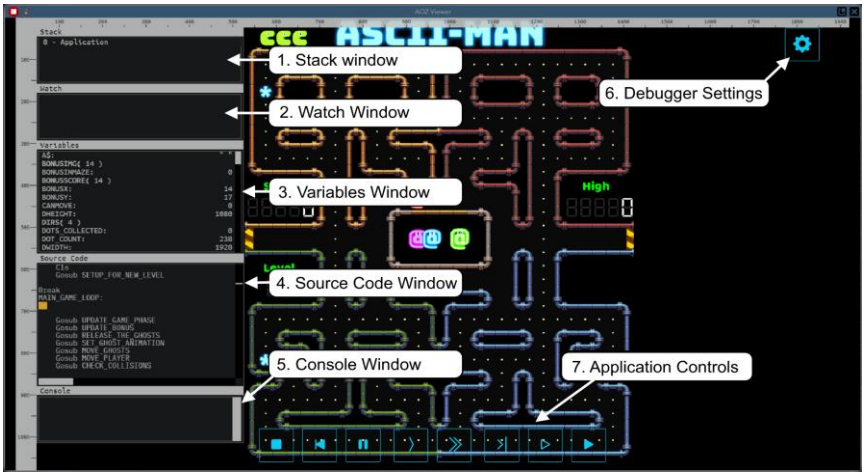
Le debugger complet s'affiche en utilisant les instructions suivantes dans votre code:

- **Debug**
- **Break**
- **Break If**

La version réduite du debugger n'affiche elle que les sous fenêtres Log et Watch avec les instructions :

- **Log**
- **Watch**

Dans le navigateur la fenêtre du debugger n'est pas interactive, elle permet seulement d'afficher les informations.



La fenêtre du debugger comprend les sous fenêtres suivantes (nous utilisons des noms Anglais comme pour les instructions):

2.1. Stack Window

Cette sous fenêtre affiche la pile d'appels actuelle de l'application. La pile augmente lorsque Gosub est utilisé ou qu'une procédure est appelée et diminue lorsque Return est exécuté ou que la procédure se termine.

2.2. Watch Window

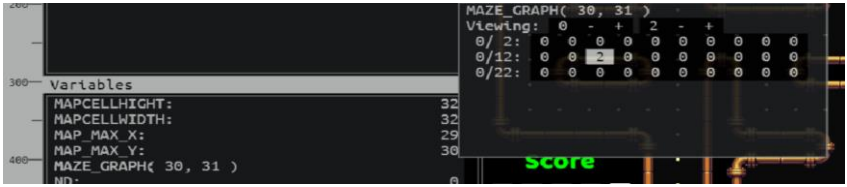
affiche les expressions que vous avez demandé à évaluer avec la commande Watch ou Break If.

Lorsque le débogueur est en cours d'exécution, l'expression est évaluée après chaque instruction (ce qui peut affecter un peu la vitesse de l'application, mais ce n'est pas important dans ce mode) et le résultat est affiché. **Break If** mettra le debugger en mode source-trace si l'expression associée est évaluée comme vraie (True).

2.3. Variables Window

affiche toutes les variables actuellement définies dans l'application. Les variables globales seront situées en haut de la liste tandis que les variables locales des procédures seront à la fin. Comme pour watch, la valeur de toutes les variables est mise à jour après chaque instruction.
Vous pouvez modifier la valeur des variables en cliquant dessus.

Les tableaux sont affichés dans une fenêtre contextuelle spécifique, vous permettant d'inspecter toutes les valeurs.



· Cliquez sur **+** et **-** pour mettre à jour la position de la vue de chaque dimension dans la popup.
· Cliquez sur la dimension elle-même pour la modifier manuellement.
· Cliquez sur l'une des valeurs de variable pour la modifier.

2.4. Source Code window

Affiche le code source de l'application.
L'instruction suivante à exécuter est surlignée en rouge, vous pouvez déplacer le curseur et inspecter le code source en cliquant sur la fenêtre. Lorsque le curseur est activé et situé après l'instruction en cours, une pression sur la touche **ENTER** lancera l'application jusqu'à la ligne où se trouve le curseur. Cette fonctionnalité vous permet de parcourir rapidement votre code.

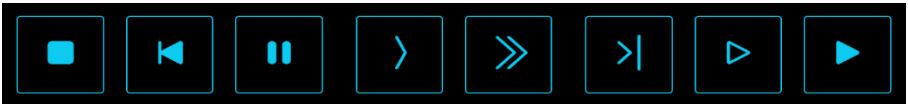
2.5. Console window

affiche chaque ligne qui lui a été envoyée avec l'instruction Log. Par défaut, le nombre de lignes mémorisées est de 100, mais cette valeur peut être définie dans l'instruction Debug On avec le paramètre consoleLines.

2.6. Le bouton des paramètres du debugger

ouvre la fenêtre de configuration du débogueur, où vous pouvez définir certains de ses paramètres comme les couleurs et le positionnement de la fenêtre.

2.7. Les boutons de contrôle des applications



Les boutons de contrôle de l'application offrent un contrôle total. De gauche à droite :

- **Arrêter** : quitte l'application et revient à l'éditeur lorsqu'elle a été exécutée sous l'IDE AOZ.
- **Redémarrer** : relance l'application. Toutes les variables seront réinitialisées, tout comme l'affichage.
- **Pause** (F8) : mettra le programme en pause à l'emplacement actuel du pointeur de programme. Si vous avez une boucle principale régulée par une instruction Wait Vbl, il est probable que la prochaine instruction sera celle après l'instruction Wait Vbl.
- **Pas à pas** (F9) : exécute l'instruction suivante et s'arrête. Si l'instruction suivante est un Gosub ou un appel de procédure ou de méthode, le débogueur entrera dans ce code.
- **Step Over** (F10) : exécute l'instruction suivante dans son ensemble, même s'il s'agit d'un Gosub ou d'un appel de

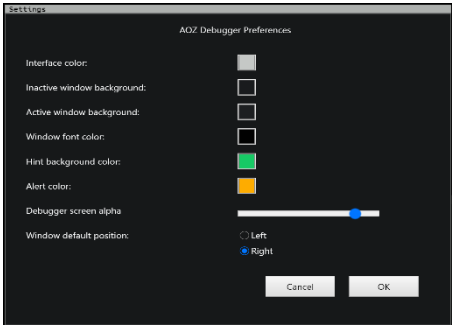
procédure. Le débogueur s'arrêtera à la prochaine instruction après l'appel. Si l'instruction suivante n'est pas un appel de routine, alors F10 se comporte comme F9.

• **Avance d'une image** (barre d'espace) : exécute l'application pour une frame complète. En d'autres termes «Avance d'une boucle». La prochaine fois que le débogueur apparaîtra, tous les images auront bougé et toutes les valeurs auront été mises à jour.

• **Exécuter au ralenti** : lance l'application au ralenti. Pendant l'exécution, vous pouvez appuyer sur Pause (**F8**) pour entrer dans le code à l'emplacement actuel du pointeur de programme.

• **Exécuter à pleine vitesse** : lance l'application à pleine vitesse. Vous pouvez masquer le débogueur en appuyant sur la touche **ÉCHAP** : cela masquera l'affichage du débogueur mais il surveillera toujours l'application et s'arrêtera si il rencontre un point d'arrêt.

2.8. Preferences Window



La fenêtre Préférences permet de choisir les différentes couleurs de son interface. Pour changer une couleur, il suffit de cliquer sur son bouton et de choisir la couleur dans le sélecteur de couleurs. La valeur alpha définit la transparence, vous pouvez appuyer sur les touches **NUMPAD +** and **NUMPAD –** pour changer ces valeurs.

Si la Position par défaut de la fenêtre est définie sur la gauche, toutes les fenêtres seront alignées à gauche de l'écran du

débogueur. Vous pouvez changer la position en appuyant sur **NUMPAD ENTER**.

RACCOURCIS CLAVIER DU DEBUGGER

Lorsque le débogueur est affiché, vous pouvez utiliser les touches suivantes :

- **NUMPAD + / NUMPAD -** : modifiez rapidement la valeur alpha de l'écran du débogueur, le rendant plus ou moins transparent.

- **PAVÉ NUMÉRIQUE** :

    - comportement par défaut : basculer la position des fenêtres du débogueur alternativement à gauche ou à droite de l'affichage.

    - lorsque le focus est dans la fenêtre source : lancez l'application jusqu'à la ligne où se trouve le curseur (uniquement si cet emplacement est après l'instruction suivante).

- **F8** : Mettre l'application en Pause

- **F9** : passez à l'instruction suivante

- **F10** : Passer au-dessus de l'instruction suivante

- **R** : active/désactive les règles

- **G** : active/désactive la grille

- **B** : active/désactive l'affichage des informations sur les Bob

- **S** : active/désactive l'affichage des informations sur les Sprites

- **W** : active/désactive la fenêtre de surveillance

- **S** : active/désactive la fenêtre Stack

- **C** : active/désactive la fenêtre de la Console

- **V** : active/désactive la fenêtre Variables

- **ESPACE** : exécute une trame de l'application

- **ESCAPE** : masque le débogueur et lance l'application à pleine vitesse ou affiche les informations du débogueur sans interférer avec l'application (permettant de voir les variables changer)

Instructions de débogage

Debug, visible, console, variables, watch, source, bobs, sprites, grid, gridWidth, gridHeight, rulers, screens, collisions, alpha#

L'instruction **Debug** lance le débogueur sans affecter le fonctionnement de votre application. Si le paramètre **visible** est défini à True, le débogueur apparaîtra et affichera les informations du moment. Il continuera à afficher les informations mises à jour pendant l'exécution du programme. Pour entrer dans le code, appuyez sur **F8** ou cliquez sur le bouton Pause.

Les paramètres de **Debug** :

• **visible** (booléen) True pour afficher le débogueur, False pour le garder caché

• **console** (booléen) True pour afficher la fenêtre de la console (par défaut) ou false pour la garder cachée

• **watch** (booléen) True pour afficher la fenêtre de surveillance (par défaut) ou false pour la garder cachée

• **source** (booléen) True pour afficher la fenêtre du code source (par défaut) ou false pour la garder cachée

• **bobs** (booléen) True pour afficher des informations pour tous les bobs ou false pour uniquement le bob sous la souris (par défaut)

• **sprites** (booléen) True pour afficher des informations pour tous les sprites ou false pour uniquement le sprite sous la souris (par défaut)

• **rulers** (booléen) True pour afficher les règles ou false pour les masquer

• **collisions** (booléen) True pour afficher les informations de collision ou false pour les garder cachées

• **grid** (boolean) True pour afficher la grille ou false pour la garder cachée

• **gridWidth** (entier) Pas horizontal de la grille (par défaut = 32)

• **gridHeight** (entier) Pas vertical de la grille (par défaut = 32)

• **alpha#** (float) valeur de la transparence alpha de l'écran du débogueur (par défaut = 0,95)

Break

L'instruction **Break** sans paramètre lancera le débogueur avec le programme en pause sur l'instruction suivant l'instruction Break. Vous êtes alors libre d'entrer dans le code avec F9 et F10 ou de relancer l'application avec F8.

Break If

L'instruction **Break If** ajoute une nouvelle expression à évaluer dans la fenêtre Watch, cela peut être n'importe quelle expression dont le résultat est un booléen (vrai ou faux). Lorsque l'expression devient vraie, le débogueur est lancé et mis en pause sur l'instruction suivant l'instruction Break If.

L'expression est évaluée en permanence après chaque instruction de votre application. Cela a le grand avantage de positionner le débogueur immédiatement sur la ligne qui est après l'expression vraie.

Exemple : Disons que votre programme modifie la variable X en mettant sa valeur à 1000, mais vous ne pouvez pas localiser exactement où dans votre code. Pour le savoir, insérez simplement l'instruction au démarrage de votre application et lancez-la… Elle s'arrêtera à l'emplacement exact où la valeur de X est définie à 1000.

```
Break If X = 1000
```

Paramètres : expression (booléen)

Watch expression

Pour ajouter une expression à la fenêtre de surveillance qui est évaluée après chaque instruction, le résultat est affiché dans la fenêtre. Un grand nombre d'expressions à surveiller peut réduire la vitesse de votre application.

Paramètres:
expression (nombre) : une expression numérique à évaluer
expression$ (string): une expression de chaîne à évaluer

Log text$

L'instruction Log génère du texte dans la fenêtre de la console. Si le debugger ne s'affiche pas lorsque Log est rencontré, une version réduite du debugger s'affiche avec uniquement la fenêtre de la console. Le débogueur complet apparaîtra si vous utilisez **Break** ou appuyez sur **F8**.

Paramètres : text$ (string) : le texte à afficher dans la fenêtre de log.

NOTE IMPORTANTE.

Pour s'exécuter, le debugger ajoute des informations supplémentaires automatiquement au code Javascript lors de la transpilation, cela augmente donc la taille du code. Votre application sera également un peu plus lente.

C'est la raison pour laquelle **il est recommandé de retirer les instructions de débogage** telles que « Log », « Break », « Break If » ou « Debug » **avant de publier / mettre en ligne**. Par protection AOZ Studio signalera une erreur si vous essayez de publier une application avec un tel code.

24. INCLUDE

Include permet d'insérer un ou plusieurs programmes dans un programme, dit autrement d'insérer un ou des codes sources AOZ dans le programme AOZ actuel. Il s'agit d'une instruction très puissante qui permet donc de séparer toutes sortes de programmes, procédures, routines, données dans des fichiers externes. Par conséquent, cela contribue à rendre votre code principal plus lisible et facile à débugger.

La syntaxe est simple :
Include « Chemin du code source à inclure ».

Vous indiquez simplement le chemin vers le fichier à inclure comme paramètre. Vous pouvez également, si le fichier est situé quelque part dans le lecteur AOZ, uniquement spécifier le nom du fichier. AOZ effectuera une recherche dans le lecteur AOZ et utilisera le fichier s'il est trouvé.

Il est parfaitement possible d'inclure plusieurs fichiers et d'inclure des fichiers dans un fichier qui est lui-même inclus (en cascade), AOZ s'assure qu'aucun code n'est dupliqué.

Voici un exemple d'inclusion de code de "AOZ Drive/includes" :

```
Include "fbdemo"  // Include le code pour  facebook...
Do                // Boucle principale du programme
 ...
 Loop
VIEW_INFOS        // Appel le code dans fbdemo qui est
maintenant "included"
```

Dans les fichiers inclus, vous pouvez placer des étiquettes que vous pouvez appeler à partir de n'importe quel code avec les instructions Goto ou Gosub. Mais attention à l'ordre dans lequel vous placez vos inclusions.
Conseil : utilisez plutôt les fichiers inclus pour les procédures de vos programmes, classés par catégories comme par ex : - user.aozinc - title.aozinc - menu.aozinc - game.aozinc

25. LES TAGS D'AOZ

Le transpiler AOZ (qui convertit votre code AOZ en Javascript et HTML) offre un système complet de balises (ou TAGS) pour contrôler tous les aspects de la transpilation de votre application. C'est beaucoup de pouvoirs entre vos mains.
Au fait, si vous avez décidé de commencer la lecture de ce guide par ce chapitre… c'est sans doute une mauvaise idée. À moins que vous soyez un développeur chevronné, rompus aux processus de compilation…

C'est quoi un TAG ?

Un TAG ou balise en Français est une chaîne qui commence avec le caractère « hashtag », (# comme sur Facebook ou Twitter).
Les TAGs sont souvent suivis d'un paramètre. Si le TAG a besoin d'un paramètre, alors vous devez le séparer du TAG lui-même avec un deux points « : ».
Ce paramètre peut être :

- **un booléen :** True ou False (pour Vrai ou Faux). Une erreur de syntaxe sera générée si la valeur est différente de ces deux- là.
- **une chaîne :** la chaîne doit être enfermée dans des guillemets.
- **un nombre :** entier ou point flottant
- **une variable système** précédemment déclarée avec l'étiquette #define. Vous pouvez y mettre un booléen, une chaîne ou un nombre. Vous pourrez bientôt également la définir dans la ligne de commande du transpiler (prochaines versions d'AOZ), le Manifest de l'application ou dans le panneau de configuration de l'application.

Remarque : le fichier Manifest.json se trouve dans le dossier "ressources" qui se trouve dans le dossier de votre application. Il décrit le cadre de son fonctionnement : la résolution, la police de caractères, le son, etc…Une lecture très intéressante.

Exemples de Tags AOZ :
#splashScreen:False //n'affiche plus l'écran du logo AOZ
#forceCompile
#displayWidth:3000
#include "./utilities/ma_super_procedure.aoz"
#ifdef LINUX_VERSION
  ...ici du code juste pour Linux
#endif

Comment utiliser les Tags

Il vous suffit d'insérer le Tag dans votre code source à l'endroit approprié.
La plupart des Tags ont un effet « global » et sont détectés pendant la phase de prétraitement du transpiler et servent généralement à donner des instructions au transpiler
D'autres balises n'ont qu'un effet sur la fonction, la méthode ou la procédure dans laquelle elles sont incluses.

Et d'autres Tags auront un effet immédiat, comme le Tag **« #include »**, qui permet de couper votre programme en morceaux que vous pourrez réutiliser. Il charge et insère le code mentionné dans le chemin de l'étiquette du Include à la position d'insertion avant de transpiler, voici un exemple d'Include :

Disons que ce petit programme est enregistré dans "Mes documents/AOZUtils/procedure.aoz » :

Procédure TAG_DEMO
Print "Ce fichier sera inclus !"
End Proc

Ensuite dans une autre application AOZ, vous pourrez faire :
#include « Mes documents/AOZ Utils/procedure.aoz »
TAG_DEMO
End

Liste des Tags disponibles

Tags de transpilation	
Définir le manifeste	
#manifest	pour définir le nom du manifeste de l'application
#export	pour définir le langage de sortie de la transpilation (html uniquement… pour l'instant)
#saveTo	pour indiquer où enregistrer l'application transpilée
#platform	pour indiquer la machine à émuler (aoz, amiga ou atari)
#tvStandard	pour une émulation d'écran en PAL ou NTSP pour l'amiga
Choisir les recompilations utiles	
#cleanExtensions	pour forcer la recompilation de toutes les extensions
#cleanModules	pour forcer la recompilation de tous les modules
#clean	pour forcer la recompilation des tous les fichiers (images, polices utilisées…)
#forceTranspile	pour forcer la recompilation d'un module ou d'une extension spécifique
#excludeFromBuild	pour exclure un module ou une extension spécifique
#includeSource	pour inclure en commentaire le code source dans l'application transpilée
Choisir le niveau de tolérance du code	
#caseSensitive	pour tenir compte (ou pas) de l'usage des Majuscules / minuscules dans le code
#syntax	???
#basicRemarks	pour accepter le format des commentaires en Basic
#noWarning	Masquer (ou pas) certains avertissements lors de la transpilation
Traiter les problèmes	
#log	pour enregistrer un journal de la transpilation
#logTo	pour indiquer où enregistrer ce journal
Tout contrôler (ou presque)	
#developerMode	pour des fonctions pratiques lors du développement des extensions et modules (notamment)
#useSource	pour transpiler un code qui se substitue au code de l'application, en conservant les ressources du projet
#define	pour définir une variable de transpilation, et permettre des transpilations conditionnelles

#let	pour attribuer une valeur à une variable de transpilation
#if, #ifdef, #else, #endif	pour définir des conditions avec des variables de transpilation
#useLocalTags	pour utiliser (ou pas) les tags locaux dans les modules et extensions (à reformuler)
#endian	pour définir l'ordre des octets dans les fichiers et la mémoire
Tags des applications	
Mentionner AOZ… et utiliser les sons	
#splashScreen	pour afficher l'écran d'éclaboussure AOZ au lancement de votre application
#useSound	pour signaler que votre application utilise le son… et forcer une interaction sur le SplashScreen (les navigateurs demandent maintenant de cliquer avant de produire le son demandé)
Dimensionner la zone d'affichage	
#displayWidth	pour définir la largeur maximale de l'écran
#displayHeight	pour définir la hauteur maximale de l'écran
#forceFullScreen	pour afficher l'application en plein écran
#keepProportions	pour conserver les proportions initiales lors d'un passage en plein écran
Donner un titre, choisir des polices et un clavier	
#appTitle	pour définir le titre de la fenêtre d'exécution de votre application
#googleFont	pour intégrer une police Google Font dans votre application
#amigaFont	pour intégrer une police Amiga dans votre application
#keymap	pour définir la keymap à utiliser
Signaler la fin de l'application, les erreurs et les crash éventuels	
#displayEndAlert	pour afficher une alerte signalant la fin de l'application
#displayErrorAlert	pour afficher une alerte signalant une erreur rencontrée par l'application
#sendCrashReport	pour envoyer à AOZ Studio un rapport en cas de crash de l'application
Tout contrôler (ou presque)	
#insertIntoHead	pour inclure votre code dans le <head> du fichier index.html généré
#tabWidth	pour définir la largeur d'une tabulation
#fps	pour afficher un indicateur FPS en haut de l'écran (à expliciter)

Tags de Transpilation

Ce paragraphe regroupe les balises utilisées par le transpiler lui-même pour transpiler votre application (ça ne fait pas le café).

cleanExtensions
type : tag
description : supprime le dossier objet de toutes les extensions avant de transpiler l'application, forçant une re-transpilation complète.
param: True ou False pour activer ou désactiver la fonctionnalité.
Eg: **#cleanExtensions: True**

Vous devrez peut-être utiliser ce Tag si vous avez copié une nouvelle extension dans votre dossier d'extension.
En outre, gardez à l'esprit (au moins pour les premières versions d'AOZ) que ce Tag peut résoudre certains problèmes compliqués de désynchronisation entre le transpiler utilisé pour pré-transpiler certaines extensions et la version actuelle de votre transpiler. Si cela se produit, faites une transpilation de votre application avec ce Tag défini à True, toutes les extensions seront alors re-transpilées (cela peut prendre un certain temps) et tout sera prêt à être utilisé.

cleanModules
description : supprime le dossier objet de tous les modules du langage AOZ avant de transpiler l'application, forçant une re-transpilation complète.
Eg: **#cleanModules: True**

Les modules contiennent la syntaxe de base de la langue. Ils sont dans AOZ le code source visible dans le dossier aoz/language/v1_0. Tous les modules sont pré-transpilés une fois pour toutes à chaque nouvelle version (c'est pourquoi la première fois pour une nouvelle version d'AOZ cela prendra plus de temps) pour ensuite accélérer la vitesse de transpilation.

Ce principe fonctionne très bien tant que toutes les sources ont été synchronisées avec la même version du transpiler. si ce n'est pas le cas, certains problèmes peuvent se produire, des erreurs de syntaxe, d'inadéquation, des plantages du transpiler ou du runtime (qui exécute le code).
Le transpiler d'AOZ Studio est modulaire : en changeant le chemin vers les dossiers qui contiennent la définition de la syntaxe, vous modifiez la syntaxe (plus à venir ! :)). Normalement, et pas avant d'être un expert en AOZ, vous ne devriez pas avoir à toucher ce code d'AOZ.
Vous devrez probablement utiliser ce Tag si vous avez copié manuellement un nouveau module dans votre dossier de module.

En outre, gardez à l'esprit (au moins pour les premières versions d'AOZ) que ce Tag peut résoudre certains problèmes de désynchronisation entre le transpiler utilisé pour pré-transpiler des modules et la version actuelle de votre transpiler.
Si cela se produit, faire une transpilation de votre application avec ce Tag défini à True : tous les modules seront re-transpilés (cela peut prendre un certain temps).

clean
description : supprime le dossier HTML de l'application avant de la transpiler, forçant une re-transpilation complète de tous les fichiers.
param: True ou False pour activer ou désactiver la fonctionnalité.
Lorsque vous travaillez sur une application, pendant le développement, vous ajoutez souvent des fichiers que vous supprimez plus tard (comme des images, des polices, etc.) ... Par défaut, et pour éviter de transpiler les fichiers qui ont déjà été transpilés, le Transpiler AOZ n'efface pas le dossier de destination, celui dans lequel votre application est construit.

Après un certain temps, ce dossier peut contenir de vieux fichiers et des choses que vous n'utilisez plus. Vous pouvez supprimer toutes ces choses inutiles avec ce Tag !
Notez que la transpilation prendra un peu plus de temps car le transpiler devra chercher et re-transpiler chaque nouveau fichier.

caseSensitive
description : le transpiler peut être, ou non, sensible à la casse (c'est à dire prendre en compte les différences Majuscules/minuscules) dans votre code.
param:True ou False pour activer ou désactiver la fonctionnalité.
Le comportement par défaut du transpileur AOZ est d'être insensible à la casse. C'est plus facile pour vos premiers programmes, cela vous permet d'aller plus vite car vous pouvez taper les noms de vos variables, procédures, méthodes ou objets, indifféremment avec de lettres majuscules ou minuscules. Par la suite, vous pourra passer au mode sensible à la casse, car c'est la façon dont tous les outils professionnels fonctionnent. On ne verrait pas du code C être insensible à la casse.

Avec le mode sensible à la casse, vous devez savoir quelques choses :

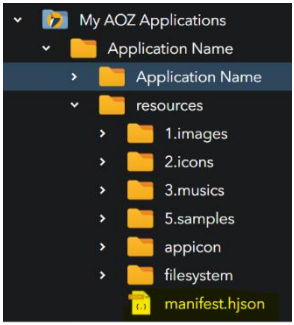
- AOZ rejettera les noms d'instructions mal écrits : Toutes les instructions d'AOZ commencent par une lettre majuscule au début de chaque mot. Exemple «Print», «True», «Screen Open» sera accepté, mais pas «screen Open» ou «false».
- La sensibilité agit également pour les variables, par exemple si vous définissez une variable comme « MyVariable = 2 », et plus tard faites «Print myvariable », votre application affichera « 0 » au lieu de « 2 ». Cela peut être générer des bogues (nom en Français pour bugs) avec des variables non initialisées. C'est la raison pour laquelle il vaut mieux supprimer tous les avertissements « Variable non déclarée » lors de la transpilation car cela peut générer un bug à un moment ou un autre, ou même un plantage de votre application.

syntax
param:nameOfSyntax$:string: « loose » ou « strict »
description : définit les variables du mode de transpilation, telles que la sensibilité à la casse et le respect de la syntaxe du Basic.
AMOS Basic, grand-père d'AOZ, était bien connu pour avoir une syntaxe souple. En outre, la langue était casse-insensible, ce qui n'est pas aujourd'hui recommandé.
Afin d'être compatible et facile à utiliser, AOZ est par défaut dans le mode "souple". Cela signifie que vous pouvez utiliser des lettres majuscules ou minuscules dans les noms de vos variables et procédures.
Par exemple : en mode «souple», le programme ci-après fonctionne :

```
score = 10 : Print SCORE
Display_Score[ score ]
Procedure DISPLAY_SCORE[ SCORE ]
        Print SCORE
End Proc
```

endian
description: définit l'ordre des octets dans les fichiers et la mémoire
param:type#:string: Soit «little» (pour pc) soit «big» (pour Amiga et Atari).
Il y a longtemps, très longtemps, certains informaticiens préféraient stocker les valeurs numériques binaires les plus faibles d'abord, puis les plus élevées. Cela a rendu les formats d'applications et de fichiers incompatibles entre les ordinateurs de différentes marques. Si votre application va enregistrer des données qui seront utilisés sur l'Amiga ou Atari, définissez ce Tag à «big»... Sinon "little" est définitivement le bon choix.

manifest
param:string: le nom du manifeste à utiliser
description: indique le manifeste à utiliser pour transpiler votre application.



Nous avons déjà parlé plus haut de ce Tag important. Le manifeste est un fichier de configuration qui se trouve dans le répertoire resources de votre application et dans lequel toutes les informations sur la transpilation de votre application sont définies. Par exemple la taille de l'écran avec ses largeur et longueur d'affichage.

Les données sont stockées sous la forme d'un objet HJSON. HJSON est une version lisible de JSON, avec les mêmes informations, mais présenté d'une manière très lisible avec des commentaires.
Les valeurs les plus importantes sont paramétrables dans l'assistant de configuration du projet (quand vous faites NEW).

La version V1.0 d'AOZ prend en charge les Tags de Manifest suivantes :

- « aoz » : l'application sera compilée pour les machines modernes en utilisant la version la plus récente de l'ensemble d'instructions.
- « amos » : votre application est conçue pour l'ordinateur Amiga. Le renderer d'AOZ (qui s'occupe de l'affichage) émule alors l'affichage d'un téléviseur CRT, y compris les zones mortes en haut, à gauche, à droite et en bas. Dans AMOS, ces zones étaient appelées «coordonnées matérielle» et n'étaient généralement pas visibles. En cas de besoin, reportez-vous au manuel original d'AMOS, inclus dans le répertoire «manuals».

- « stos » : (pas encore disponible) A utiliser pour exécuter des applications de STOS sur l'Atari ST... Il émulera l'affichage de la machine.

platform
description : indique la machine émulée par le renderer.
param:nameOfThePlatform$:string: « aoz », « amiga » ou « atari »
Ce Tag est utilisé par le renderer. En mode « normal » (comprendre « aoz »), le renderer utilise une machine moderne permettant des graphiques haute résolution et des multitudes de sprites et bobs. Il n'y a pas de «Coordonnées matérielles», les sprites, bobs sont affichés à leurs coordonnées exactes.

Note : Si ce Tag est défini sur « amiga », le renderer émule l'affichage d'un téléviseur CRT, avec un nombre limité de pixels et de zones vierges sur la gauche, le haut, la droite et le bas.
Veuillez noter que pour le moment, le "copper" de l'Amiga n'est pas émulé et que toute application utilisant ses commandes n'auront tout simplement aucun effet sur l'écran.

Si ce Tag est sur « atari », l'affichage sera également émulé avec un résultat maximum de 640x400 en 2 couleurs.

useLocalTags
description : laisser fonctionner les Tags locaux dans les extensions ou les modules.
content : les Tags locaux sont uniquement valides dans des fichiers ou extensions.

basicRemarks
description: autoriser l'utilisation, ou non, du caractère ' (apostrophe) comme une remarque dans votre code
param:onOff:boolean
La syntaxe originale du langage Basic vous permet de définir des remarques/commentaires avec le caractère apostrophe, qu'AOZ

ignore dans la transpilation comme tous les modes de remarques possibles :

```
Print "Hello AOZ"
'       C'est une remarque avec une apostrophe
Rem  Ca une remarque avec l'instruction Rem
//      Une remarque de style langage C
/*
  Et ca c'est une longue remarque de style C,
  qui peut inclure n'importe quel nombre de lignes entre ces
   /*   .... */
*/
```

Pour les commentaires nous vous recommandons d'utiliser **//** et **/* */** , plutôt que **'** et **Rem** qui sont anciens.

noWarning
description : supprime un avertissement spécifique du transpiler dans la vérification de la syntaxe.
param:warningId:string: la chaîne d'identification de l'avertissement.
comment:
L'utilisation la plus fréquente de cette instruction est de supprimer l'avertissement : Variable non réservée.
Bien que nous vous suggérions fortement de ne pas supprimer cet avertissement, si vous codez une nouvelle application pour définir toutes les variables avant de les utiliser, ce Tag peut s'avérer utile lors de l'exécution des applications STOS et AMOS qui utilisaient l'ancienne syntaxe des remarques.

Liste des identificateurs d'avertissement :
- font_not_found
- garbage_found_in_folder
- font_not_supported
- file_at_root_of_filesystem
- screen_not_multiple_of_font_size

- missing_folder
- missing_resources_folder
- creating_directory
- cannot_set_permissions
- illegal_bank_element_filename
- file_to_include
- copying_file_to_filesystem
- variable_not_declared
- duplicate_error_message
- instruction_not_implemented
- should_wait
- out_of_unique_identifiers

Exemple :
#noWarning: « font_not_found »
#noWarning: « variable+not_declared »

tvStandard
description : indique qu'il faut émuler un écran NTSC pour Amiga. (L'émulation Amiga est par défaut en mode PAL.)
param:Active ou désactive cette fonctionnalité

export
description: définit le langage de sortie de la transpilation.
param:exportType$:string: le nom de la plate-forme de destination.
Ce Tag ne supporte actuellement que « html ». Dans une version future, il vous permettra également d'exporter directement vers Facebook (avec la valeur « htmlFb »), des exécutables pour Windows macOS et Linux (« windowsExe », « macOS », « linuxExe ») et plus tard pour des moteurs 3D spécifiques comme Unreal ou Unity.

saveTo
description : indique où enregistrer l'application transpilée.

param:path$:string: chemin vers un dossier où l'application transpilée est enregistrée.
Par défaut, AOZ enregistre le dossier HTML avec l'application transpilée au même niveau que le code source .aoz.

Le fichier de log est un journal dans lequel le transpiler enregistre ce qui se passe, ainsi en cas de problème, son examen permet d'en identifier la cause.

log
description : demande au transpiler d'enregistrer un journal à chaque transpilation.
param: True pour enregistrer le log (journal) de transpilation, false = pas de log, c'est la valeur par défaut.
Ces journaux (logs) peuvent être utiles en cas de problèmes complexes avec le transpiler. Il ne « compile » pas, mais pourquoi ? Il devrait ! Si cela vous arrive, activez la fonction journal pour inspecter les fichiers de logs et sans doute trouver le problème.

logTo
description : change le répertoire utilisé par le transpileur pour enregistrer le log.
param:path$:string: Chemin vers un dossier où les journaux du transpileur sont enregistrés.
Par défaut, AOZ n'enregistre aucun journal. Vous pouvez activer cette fonctionnalité avec l'étiquette « #log:true » : AAOZ enregistrera ses journaux de transpilation dans le dossier aoz/logs. Ce Tag vous permet de définir votre propre dossier pour enregistrer les journaux.

forceTranspile
description : force la transpilation de ce module ou extension.
Ce Tag est le contraire de #excludeFromBuild, elle force la transpilation du module ou de l'extension chaque fois que vous transpilez votre application.

Les développeurs apprécieront la sécurité mentale de savoir que le code de votre nouvelle extension est toujours compilé, spécialement en face de gros problèmes de débogage où le doute vient (une extension non recompilée pourrait être l'origine de ce bug)

developerMode
description: transformer AOZ en développeur mode.
param: True ou False pour activer ou désactiver la fonctionnalité.
AOZ contient un mode développeur qui peut être utile lorsque vous écrivez une extension ou que vous souhaitez des fonctionnalités plus profondes.
Il ne fait pas grand-chose pour le moment (sic), mais de nouvelles options seront ajoutées à coup sûr dans les prochains mois.
Pour l'instant (v 1.0 d'AOZ), être en mode développeur apporte les avantages suivants :

- lors de l'exécution de votre application, AOZ ne va *pas* intercepter les touches système du navigateur, vous serez ainsi en mesure d'ouvrir le debugger Chrome avec la touche de fonction F12 par exemple.
- AOZ scanne toutes les extensions et modules des dossiers à la recherche du code qui a changé depuis la dernière compilation. Si vous ne développez pas d'extension et n'avez pas modifié le contenu de l'intérieur du dossier système aoz, vous n'en avez pas besoin. Mais si vous développez une nouvelle extension, un simple RUN de votre application de test AOZ fera compiler cette extension. Veuillez noter que cette option rendra également le processus de transpilation (un peu) plus lent car AOZ doit scanner les 50 sous-dossiers dans les répertoires d'extensions et de modules. Comme le nombre d'extensions va augmenter à l'avenir, et atteindre un jour la taille de node.js (rêvons !), il pourrait devenir très pratique alors de s'assurer que tout est compilé avec la dernière version.

Vous pouvez activer le mode développeur et le désactiver dans le panneau « Paramètres » de l'AOZ Studio.

includeSource
description: copie le code source de l'application comme remarques dans le code transpilé
param: True ou False pour activer ou désactiver la fonctionnalité.
Cette option est True par défaut, et oblige le transpileur à inclure dans le code qu'il produit la ligne d' AOZ transpilée.
Un grand soin a été pris pour que le code source de votre application transpilée soit lisible, nous espérons que cela vous conviendra.
Le code de votre application transpilée se trouve dans le code source my_application/html/run/application.js après une transpilation réussie.

useSource
description : remplace le script d'entrée par la chaîne donnée et procède à la compilation.
param: code$:string: Le code à remplacer par.
Ce Tag est destiné aux développeurs qui utilisent le transpiler AOZ comme un module node.js (à venir plus tard).
Avec ce Tag, vous pouvez appeler la fonction « transpile » du module AOZ remplaçant le code de l'application mentionnée dans le chemin de la commande « transpile » par un autre code.
Ce nouveau code remplacera simplement le code d'origine dans le projet, le projet gardant son dossier de ressources.
Nous l'utilisons en interne par exemple lorsque vous ouvrez la fenêtre Mode Direct dans l'IDE AOZ : l'application actuelle est transpilée avec un code source vide, ce qui rend toutes les ressources d'image et de son disponibles en mode direct pour expérimenter.

define
description : définit une variable à utiliser pendant la transpilation.
param: Le nom de la variable à créer, sans guillemets.
AOZ permet la transpilation conditionnelle comme en C++ ou C#. Vous définissez les variables de transpilation avec la balise #define, puis utilisez #if, #else ou #endif, ou #ifdef et #endif pour définir les zones de votre application qui seront transpilées.
Cette option, que l'on retrouve dans les compilateurs professionnels C ou C#, est disponible en AOZ. Cet exemple montre l'utilisation de ce Tag :

```
#splashScreen:false
#define MACVERSION
#ifdef MACVERSION
        Print "Cette application a été transpilée pour macOS"
#else
        Print "Cette application a été transpilée pour une autre plateforme."
#endif
```

Si vous remplacez l'étiquette par :
#define WINDOWS
… vous allez changer le code transpilé. Si vous inspectez le code de l'application transpilée, vous verrez que seul le code choisi a été transpilé. Ce système vous permet également de supprimer physiquement le code, ce qui peut être important pour des applications de démonstration jusqu'à ce que l'utilisateur achète une licence.

AOZ contient un préprocesseur, appelé « Pass Zero » (Salut Pass Zéro). Il scanne le code source en tenant compte des variables transpiler et saute la partie du code qui n'est pas active à ce moment.

Cette option très puissante vous permet d'effectuer plusieurs versions de la même application, avec seulement des modifications mineures d'une plate-forme à l'autre, ou différentes versions telles qu'une version gratuite et commerciale avec la même application.

let
description : attribue une valeur à une variable système
param: nom: le nom de la variable.
param: valeur: any:La valeur à mettre dans la variable, peut être une chaîne, un nombre ou une valeur de type boolean.
Attention : notez que ce Tag n'a pas besoin du caractère : après le nom de l'étiquette, mais plutôt du caractère égal, « = ».

Utilisez ce Tag pour la transpilation conditionnelle de parties spécifiques du code source.

```
#let VERSION = "0.99"
#if version <> "1.00"
        Print "Hmmm, ca peut crasher !"
#else
        Print " Oui ! Ca ne va pas crasher, je promet (croise
les doigts :)"
#endif
```

if
description : compare la valeur d'une variable système avec une autre variable système
param: variable:nom: le nom de la variable à tester.
param:expression:any: l'expression à comparer à
La variable système peut porter des valeurs, et #let tag vous permet de les définir. Notez que #let diffère de #define en ce sens qu'une valeur réelle est affectée à la variable du système, alors qu'avec #define la variable est créée avec une valeur de zéro.

L'expression peut inclure :
- nombres
- chaîne
- valeurs boolean
- autre variable système contenant des valeurs

Notes qu'il n'y a pas de « type » associé à des variables système, et que vous n'avez pas besoin d'ajouter un caractère $ après le nom d'une variable système de chaîne ou un # après un chiffre en virgule flottante.

```
#let VERSION = "0.99"
#if version <> "1.00"
  Print "Hmmm, ca pourrait crasher !"
#else
  Print "YES ! Ca ne devrait pas crasher :)"
#endif
```

ifdef
description : vérifiez si une variable de transpilation est définie ou non.
param: variable: nom: Le nom de la variable à tester.

Utilisez ce Tag pour la transpilation conditionnelle de parties spécifiques du code source.

```
#define:IWILLNOTCOMPILE
#ifdef IWILLNOTCOMPILE
  A = "bonjour" // Cette ligne générera une erreur de syntaxe
#endif
```

else
description : indique le code à transpiler si l'état d'un Tag #if ou #ifdef n'est PAS vérifié.

Utilisez ce Tag pour la transpilation conditionnelle de parties spécifiques du code source.

```
#define IWILLNOTCOMPILE
#ifdef IWILLNOTCOMPILE
  A = "bonjour"// Cette ligne générera une erreur de syntaxe
#else
  Print "Tout va bien !"
#endif
```

Endif

description: ferme une section de test commencée avec #if ou #ifdef
Utilisez ce Tag pour la transpilation conditionnelle de parties spécifiques du code source.

excludeFromBuild
description : exclu l'extension ou le module du build (du code) général (l'utiliser pour le développement).
Ce Tag a été créée pour les développeurs de modules ou d'extensions pour AOZ. Une fois que le code de l'extension sur lequel vous travaillez est ouvert, il peut empêcher la chaîne de compilation des autres modules en générant des erreurs.
Utilisez ce Tag pour supprimer le module ou l'extension de la liste de transpilation, et vous pouvez utiliser AOZ jusqu'à ce que vous reveniez à ce code.

Tags des applications

SplashScreen
description : active ou désactive l'écran d'éclaboussures (c'est la traduction de SplashScreen !), l'écran avec le logo AOZ Studio qui s'affiche au lancement de votre application AOZ.
param:onOff:boolean:Activer ou désactiver l'écran splash.

Ex : **#SplashScreen: False** //n'affiche pas l'écran AOZ Studio

Note : L'utilisation de ce Tag n'est possible qu'avec la version achetée sous licence AOZ Studio. La version gratuite ne permet pas d'enlever ces éclaboussures 😊.
Oui, on assume, c'est fait pour vous inciter à acheter une licence !
Par défaut, lorsque vous exécutez votre application AOZ, un écran d'éclaboussures de logos, dont d'AOZ Studio s'affiche.
Si vous utilisez la version gratuite d'AOZ, cet écran ou ces publicités sont toujours affichés (cf. contrat de licence). Pire la durée de ces éclaboussures augmentera progressivement.
Si vous avez une version/licence payante valide, vous pourrez désactiver cet affichage avec ce tag.

Pour les applications qui utilisent des sons, un écran splashscreen a toutefois une utilité : il permet à votre superbe bande son d'être entendue dans votre navigateur Web. En effet, les navigateurs modernes comprennent une protection pour empêcher toute page web de jouer des sons à l'ouverture d'une page : les sons ne sont audibles qu'après une première interaction de l'utilisateur (un clic, un toucher d'écran, une frappe au clavier…).
Afficher un splashscreen, quelque chose sur lequel il faut cliquer est une bonne façon de provoquer une telle interaction… et ainsi permettre ensuite au navigateur de jouer votre musique.
Je résume : Si vous supprimez l'écran d'éclaboussures, vous devrez attendre que l'utilisateur clique quelque part avant de jouer le premier son.

displayWidth
description: définit la largeur maximale de l'écran en pixels.
param:width:integer: la largeur de l'affichage en pixels.
Ce tag est équivalent à modifier la valeur display.width dans le manifeste de votre application.
Eg: **#displayWidth:640**

displayHeight
description : définit la hauteur maximale de l'écran en pixels.
param: height:integer: la hauteur de l'écran en pixels.
Ce tag équivaut à modifier la valeur display.height dans le manifeste de votre application.
Eg: **#displayHeight:4800**

forceFullScreen
description: force l'application en plein écran
param: Vrai ou faux pour activer ou désactiver la fonctionnalité.
Ce tag force votre application en plein écran, en tenant compte d'exécution :

- dans un navigateur, votre application s'affichera en pleine fenêtre d'abord, et il faudra que l'utilisateur interagisse pour que la fenêtre passe en plein écran ;
- dans le viewer autonome d'AOZ, votre application s'affichera immédiatement en plein écran.

Eg: **#forceFullScreen:true**

keepProportions
description : en mode pleine page ou plein écran, indiquer si le rapport de dimension de l'application doit être préservé.
param: True ou False pour activer ou désactiver la fonctionnalité.
Si vous demandez à conserver les proportions initiales, des barres vides seront ajoutées à gauche-droite ou en haut-bas de l'application.
Dans le cas contraire, votre espace de travail sera redimensionné

pour occuper toute la surface de l'écran… au risque d'une d'un étirement horizontal ou vertical de vos graphiques.
Eg: **#keepPropotions:true**

fps
description : affiche un indicateur fps en haut de l'écran de l'application.
param: True ou False pour activer ou désactiver la fonctionnalité. Vous pouvez définir la largeur et la hauteur de l'indicateur FPS en modifiant le manifeste utilisé par votre application.

appTitle
param:string:Le nouveau titre à définir
description: définit le titre de la fenêtre d'exécution de l'application
Cette instruction n'aura aucun effet si l'application est en cours d'utilisation dans AOZ Studio.

googleFont
description: intègre une police Google dans votre application AOZ.
param:fontName$:string: le nom de la police à intégrer.
comment:
AOZ Studio prend en charge Google Fonts et intègre la ou les polices souhaitées dans votre application… même en mode non connecté.

Les polices sont de simples fichiers téléchargés à partir de Google, et sont enregistrés dans le dossier aoz/polices/google de votre installation d'AOZ.
Pour ajouter une nouvelle police Google, utilisez l'accessoire « Add Font » ou copiez les fichiers manuellement. Vous trouverez un modèle dans aoz / modèles / google police.
Attention : certaines polices Google particulières nécessitent de nombreux fichiers pour inclure toutes les tailles et formes.

Ajouter une telle police peut augmenter considérablement la taille de votre application.

amigaFont
description: intègre une police Amiga dans votre application AOZ.
param:fontName:string:Le nom de la police à ajouter.
comment:
Les amateurs d'Amiga seront heureux de savoir qu'AOZ supporte les polices Amiga en noir et blanc. Vous serez heureux revoir ces polices sur un écran moderne, certaines d'entre elles étaient vraiment super.
Le système fonctionne de la même manière que pour #googleFont : vous indiquez le nom de la police et les fichiers qui la définissent seront copiés dans votre application.
Pour ajouter une nouvelle police Amiga, utilisez l'accessoire « Add Font » ou copiez les fichiers manuellement. Vous trouverez un modèle dans aoz / modèles / google police.
La police Amiga fonctionnera quelle que soit la configuration du transpiler (aoz ou amiga ou atari), mais elle peut être déformée sur des écrans en haute résolution.
Contrairement aux polices modernes qui s'adaptent aux différentes résolutions d'écran (ce sont des polices vectorielles) les polices Amiga sont définies par des points (on parle de police bitmap) et fournie avec un fichier par taille de police… Et si AOZ ne peut pas trouver une taille de police adaptée à la résolution de votre magnifique écran, il prendra la police la plus proche et agrandira / déformera les lettres. Désolé, on adore les polices Amiga, mais on ne peut pas mieux faire !

keymap
description : définit la keymap à utiliser avec l'application.
param:keymap$:string:"aoz », « amiga » ou « atari »
Une valeur de « aoz » utilisera le clavier Javascript par défaut. Une valeur de « amiga » ou « atari » utilisera la keymap propres à ces machines.

tabWidth
description: définit la largeur d'une tabulation
param:numberOfSpaces:integer: le nombre d'espaces à utiliser pour chaque tabulation. Par défaut est de 4.

displayEndAlert
description : affiche une alerte à la fin de votre application en cas d'erreur.
param: Allumez ou éteignez la fonction.
AOZ affiche une alerte indiquant la fin de votre application. Si vous inactivez cette option avec ce Tag, elle cessera immédiatement (le programme s'arrêtera dans le navigateur).
Les applications enregistrées reviendront au système avec un code d'erreur de 0 (aucune erreur).

displayErrorAlert
description : affiche une alerte à la fin de votre application en cas d'erreur.
param:onOff: Allumez ou éteignez la fonction.
Lorsqu'une erreur se produit dans votre application, AOZ affiche par défaut une boîte d'alerte avant d'arrêter de fumer. Si vous désactivez cette option avec ce Tag, elle cessera immédiatement après la panne (le programme s'arrêtera dans le navigateur).
Les applications enregistrent car les commandes autonomes signaleront le nombre d'erreurs en conséquence.

sendCrashReport
description : envoie automatiquement un rapport après un crash de l'application ou du transpiler à AOZ Studio, avec des informations sur le crash.
param: True ou False pour activer ou désactiver la fonctionnalité.
Les informations transmises sont totalement anonymes et ne contiennent que des données techniques sur AOZ, votre application et votre machine.

Elles nous aident à améliorer AOZ et à supprimer autant de bugs que possible. Mais c'est à vous d'accepter ce partage d'informations. Merci d'avance ;)

useSounds
description : force l'indicateur « son » et affiche « Cliquez pour continuer » sur l'écran d'éclaboussure.
param: True ou False pour activer ou désactiver la fonctionnalité.
Pour les applications qui utilisent des sons, l'écran splash a un rôle très important cependant: il permet aux sons d'être entendus dans votre navigateur. Navigateur moderne comprennent une protection pour empêcher toute page de jouer des sons forts (imaginez une page popup ouverture avec une vidéo criant à vous d'acheter un produit).
Avant tout son est joué sur n'importe quel navigateur (étant Chrome, Firefox, Safari ... ou leur équivalent mobile), il doit y avoir une véritable interaction utilisateur, un simple clic ou toucher. C'est la raison pour laquelle si vous utilisez des sons dans votre application, l'écran d'éclaboussures par défaut demande une telle confirmation. Dans ce cas, l'écran splash a un rôle réel.
Si vous supprimez l'écran d'éclaboussure, vous devrez attendre que l'utilisateur clique quelque part avant de jouer le premier son, sinon toute l'API sonore sera désactivée.
Cf le TAG:splashScreen

insertIntoHead
description : insère une chaîne dans la section <head> du fichier index.html généré par le transpiler.
param:CSS$:string: une chaîne contenant du code à insérer dans le HTML <head> </head> section de l'index.html fichier qui vous lance application.

Pour les développeurs avancés seulement, ou les personnes qui veulent faire des extensions, ce tag vous donne la liberté d'inclure votre code dans index.html.

26. VITESSE ET WAIT VBL

Lors du développement d'AOZ Studio, nous avons été confrontés à un problème : le Basic est un langage qui autorise les boucles, mais pas le JavaScript.

Cela signifie qu'en JavaScript, il est impossible pour une application de « tourner » plus longtemps qu'une durée raisonnable ; durée qui dépend de la machine, de sa charge, du navigateur. Si une application JavaScript prend trop de temps, le navigateur semble « planter » alors qu'étant mono-tâche il est juste occupé.

Afin d'implémenter des boucles dans AOZ, nous avons utilisé une astuce : l'application AOZ est exécutée jusqu'à chaque branche : comme celles des instructions Goto, Gosub, les appels de procédure, ou les simples If / Else / End If… bref, tout ce qui conduit à une modification du pointeur d'exécution du programme.

À chaque fois AOZ prend la décision de rafraîchir ou non l'affichage et de "redonner la main" au système afin que le navigateur dispose de suffisamment de temps machine. Avant de "rendre la main", AOZ stocke la position du pointeur de programme dans votre application et définit un "rendez-vous" avec le système au prochain rafraîchissement de l'écran, généralement au 1/60e de seconde suivant.

1/60e de seconde plus tard, AOZ rentre dans l'application à la position qu'il a quitté et continue, assurant un affichage cohérent et une vitesse régulière...

Le "compilation.speed" dans le manifeste affecte le rafraîchissement ou non de l'écran et la façon de rendre la main au système. Il peut prendre trois valeurs.

● « **fast** » : c'est le réglage par défaut.

En mode rapide, AOZ compte sur vous, le programmeur, pour redonner la main au système. Pour cela utilisez l'instruction "Wait Vbl" à la fin de la boucle principale de votre application. Si vous regardez les tutoriels et exemples fournis avec AOZ, vous verrez qu'ils ont tous un Wait Vbl quelque part si il y a des affichages.

En mode « fast », l'affichage est mis à jour quand l'instruction Wait Vbl est rencontré. Cela signifie que l'effet de toutes les instructions graphiques Bob, Sprite, Actor, ne sera visible à l'écran que lorsque la main sera "rendue", mais cela n'a aucun effet sur l'application elle-même ou le taux de rafraîchissement.

Remarque : vous n'avez besoin de Wait Vbl que dans la boucle principale (la boucle la plus basse de votre application). L'instruction Wait Vbl prend 1/60e de seconde à s'exécuter, elle ralentit donc le programme ; dés lors l'insérer dans des boucles qui manipulent des données ou n'affichent rien n'a aucun intérêt et ralentirait le programme.

Attention, certaines instructions graphiques peuvent prendre du temps à s'exécuter, une petite boucle avec des milliers d'ellipses prendra beaucoup de temps : pendant cette boucle, le navigateur peut avoir l'air d'avoir planté et vous devrez peut-être utiliser le paramètre « fair » à la place.

● « **fair** » : pour une application graphique plus rapide.

Si votre application a une boucle principale qui dessine ou imprime beaucoup de données à l'écran, et en même temps effectue des animations, les opérations graphiques peuvent prendre plus d'1/60ème de seconde, le navigateur va ralentir et l'affichage peut devenir saccadé.

Le paramètre « fair » garantit que la main est rendue plus souvent au système en multipliant le nombre de tests dans les branches et en vérifiant que l'application n'a pas mis plus de 16 millisecondes à s'exécuter.

Si ce paramètre ne fonctionne pas pour votre application, vous pouvez choisir le mode : « safe ».

● « **safe** » : devrait fonctionner pour tout.

Ce paramètre garantit que l'application ne prend pas plus de 12 millisecondes et vérifie le retour toutes les 1000 branches. En gros, toutes les 1000 fins de boucles, AOZ demande l'heure au système et vérifie que l'application n'a pas dépassé ce temps. Cette procédure garantit que le navigateur reste réactif quel que soit le code.

Note : Vous pouvez trouver dans certaines applications plus anciennes, des versions du manifeste qui utilisent un mode «graphic», ceci n'est plus supporté.

Comment trouver les bons réglages ?

Lorsque vous créez une nouvelle application, le mode par défaut est « fast ». Ce devrait être le mode de choix car c'est le plus rapide et le plus cohérent. Vous devez juste vous assurer que votre boucle principale est régulée par une instruction Wait Vbl à la fin.

Si cela ne suffit pas, remplacez « fast » par « fair » et relancez votre application. Si cela ne suffit toujours pas, remplacez « équitable » par « safe » et réessayez.

Si avec ce dernier paramétrage vous ne parvenez toujours pas à obtenir une vitesse et un affichage cohérents pour votre application, n'hésitez pas à nous contacter sur Discord nous vous aiderons à tirer le meilleur parti de votre code en trouvant où stabiliser la vitesse.

27. COMMUNIQUER AVEC UN SERVEUR

Préambule : Nous pensons qu'AOZ Studio doit vous simplifier tous les aspects de la programmation. La création et la gestion des bases de données est un domaine sur lequel nous allons travailler pour vous proposer également de nouvelles solutions.
En attendant si vous avez besoin d'utiliser des bases de données voila deux manières de faire classiques :

Les programmes ont souvent besoin d'échanger des informations à travers le réseau en se connectant à des serveurs qui peuvent se trouver n'importe où dans le Monde.
AOZ Studio permet d'établir très facilement ce type de communication et d'échanger des données entre plusieurs machines.

Garçon !
Imaginons que notre programme soit un client, installé à la terrasse d'un café, et qui passe sa commande auprès du serveur. Ce dernier répond à la demande (requête) de son client et lui sert ce qu'il a demandé (son café-croissant).
Pour votre application, c'est le même principe. Votre programme AOZ joue le rôle du client, et passe sa commande auprès d'un serveur à distance.
La requête de notre programme AOZ est alors traitée par le serveur et celui-ci retourne une réponse.

Voici un exemple :

```
// Adresse du serveur sur laquelle la requête va être effectuée
URL$="http://aoz-studio.online/tutos/get_post/get.php"

START:
Input "Que puis-je vous servir ? (café, chocolat ou croissant ):"; REQ$
 //Passons la commande au serveur
 //pSuccess est la procédure appelée en cas de réponse de l'URL
 //pError est la procedure appelée en cas d'échec de la requête
```

```
Call Service "get", URL$, "request=" + REQ$, "pSuccess", "pError"
Print "Autre chose ?"
Wait key

Goto START

Procedure pSuccess[ status$, response$ ]
        Print "pSuccess"
        Print status$
        Print response$
End Proc
Procedure pError[ status$, response$ ]
        Print "Error!"
        Print status$
        Print response$
End Proc
```

Passez votre commande en saisissant ce que vous souhaitez commander parmi un café, un chocolat ou un croissant.

Votre commande est stockée dans la variable REQ$, et transmise au serveur grâce à l'instruction AOZ **Call Service**.
Call Service appelle l'adresse du service (contenu dans la variable URL$) et transmet le contenu de la variable REQ$ en la passant dans le paramètre « request ».
Lorsque le serveur réceptionne la commande, il la traite et appelle la procédure AOZ « pSuccess » et lui donne sa réponse.

Call Service "get",... pour lire
Call Service "post",... pour écrire

Si une erreur de traitement survient, c'est la procédure AOZ « pError » qui est appelée.

Call Service propose plusieurs paramètres :

Call Service <méthode>, <adresse>, <valeurs à transmettre>, <procédure AOZ de réponse>, <procédure AOZ de l'erreur>

- <méthode> : AOZ gère 2 méthodes de transmission des données au serveur :
    - « get » : les valeurs sont transmises en claires et peuvent être facilement lisibles par un tier. C'est la méthode à utiliser pour effectuer des demandes qui ne transmettent pas de données sensibles.
    - « post » : les valeurs sont encodées et invisibles par un tiers. C'est la méthode à utiliser pour transmettre des données sensibles (identification, email, compte bancaire…)
- <adresse> : Adresse du serveur qui pourra traiter votre demande
- <valeurs à transmettre> : Liste des valeurs à transmettre au serveur pour le traitement de votre demande. Cette liste est constituée de « nom variable=valeur variable » séparés par le caractère « & ».

  Par exemple :
  « userid=magician&password=1234567 »
- <procédure AOZ de réponse> et <procédure AOZ de l'erreur> : Nom des procédures AOZ à appeler en cas de réponse ou d'erreur du serveur.
  Elles peuvent récupérer 2 valeurs :
    - status$: Contient le code de l'état de la requête :
        - « 200 » : réponse
        - « 403 » : permission refusée
        - « 404 » : serveur introuvable
        - « 500 » : erreur lors de la demande
    - response$: Contient la réponse du serveur

Bien dans ses chaussettes !
Le second moyen de communication possible avec AOZ Studio est nommée « socket ». Il s'agit d'un service qui permet à plusieurs personnes de communiquer ensemble, comme, par exemple, pour un Chat ou un jeu multijoueurs.

Le serveur reçoit les messages d'un ou plusieurs utilisateurs, et les transmet aux autres utilisateurs connectés. C'est un échange quasi instantané. Contrairement au système de requêtes dont on a parlé avant, qui appelle le serveur uniquement quand c'est nécessaire, ici le serveur écoute constamment l'arrive de nouveaux messages.

AOZ Studio propose plusieurs commandes pour ouvrir une communication vers un serveur de « socket » et transmettre/recevoir des informations.

Ouvrez, depuis votre navigateur, le site suivant :
https://www.piesocket.com/websocket-tester
et cliquez sur le bouton Connect.

```
Rem Ouvre la connexion au serveur de Socket sur le canal 1
WebSocket Open 1,
"wss://demo.websocket.me/v3/channel_1?api_key=oCdCMcMPQpbvNj
UIzqtvF1d2X2okWpDQj4AwARJuAgtjhzKxVEjQU6IdCjwm&notify_self"

Rem Attend la connexion au serveur
Repeat
   Wait Vbl
Until WebSocket Is Connected(1)

Print "La communication a été établie."
Print "Appuyez sur une touche pour envoyer HELLO au serveur."
Do
   If WebSocket Is Message(1)
      Rem Affiche le message reçu
      Pen 5 : Print "Reçu: " + WebSocket Message$(1)
   End If
   A$=Inkey$
   If A$ = " " : REM Note il y a un espace entre les guillements
```

```
      Rem Envoi "HELLO" au serveur
      Pen 4 : Print "Envoi: HELLO"
      WebSocket Send Message 1, "HELLO"
    End If
    If A$="e" Then Exit
    Wait Vbl
Loop
Rem Déconnexion du serveur de Socket
WebSocket Close 1
```

Exécutez ce programme. Lorsque la connexion est établie avec le serveur, et que vous appuyez sur la barre ESPACE, un message HELLO est envoyé. Si vous regardez le site sur votre navigateur, vous remarquerez que le message HELLO est apparu dans le champ des messages.

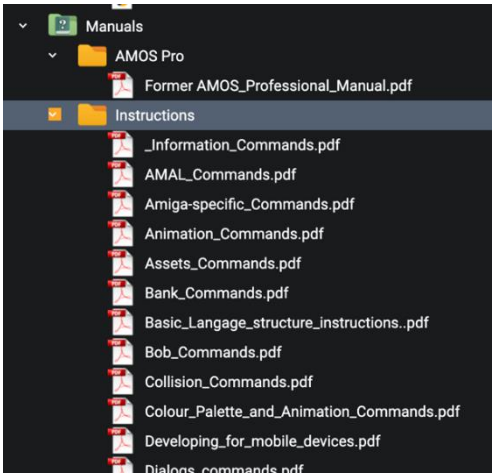
Depuis ce site, envoyez un message au même serveur. Si vous regardez votre programme, le message apparait sur votre écran. La connexion fonctionne parfaitement et les données sont bien transmises.

28. POUR LES UTILISATEURS D'AMOS

Ce chapitre s'adresse à nos anciens amis qui connaissent AMOS, Easy AMOS et AMOS Professional, qui sont les ancêtres d'AOZ et avec lesquels AOZ Studio est compatible.

Compatibilité

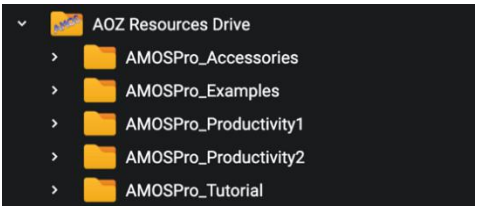
Tout d'abord, nous sommes ravis de vous dire que la plupart du code et des banques de vos anciens programmes peuvent être directement chargés dans AOZ. Si tout va bien, tout s'importe automatiquement quand vous chargez un programme AMOS.



Tout le manuel AMOS Professional vous attend dans le répertoire Manuals, ainsi que toutes les instructions, commandes et modules de A à Z. Et vous trouverez une grande gamme d'instructions flambant neuves, qui vous permettront de développer votre travail pour une utilisation sur les téléphones mobiles, ou de manipuler le code HTML pour vos sites Web.

Nous avons même inclus le jeu amiga classique appelé CyBall et l'avons importé dans AOZ pour vous, ainsi qu'un tutoriel complet, comment cela a été fait sur Youtube par Phill Bell Comment j'ai porté mon jeu AMOS à AOZ Studio Start to Finish

Rappelez-vous, il y a un dossier dédié AMOS que vous pouvez explorer à votre guise, avec des masses de tutoriels, d'exemples et d'accessoires que vous pouvez agiter avec votre baguette magique.



Il y a plus de similitudes qu'il n'y a des différences entre votre ancien AMOS et votre nouvelle langue AOZ. La principale différence est que AOZ fonctionne beaucoup plus vite, avec plus de fonctions et instructions et n'est plus limité à un ordinateur.

FICHIERS AUDIO

Le moteur tracker utilisé prend en charge non seulement les fichiers Amiga SoundTracker, mais aussi les nouveaux formats supportant jusqu'à 32 canaux, l'audio XM et plus encore.

BOBS ET SPRITES

Maintenant il est temps de vous présenter les instructions de renommée mondiale **Bob** et **Sprite**, créé pour AMOS. Si vous vous souvenez AMOS est le grand-père d'AOZ créé dans les années 1980. Avec AMOS on manipulait les objets graphiques avec des Bobs et des Sprites. Avec AOZ, on utilise plutôt Actor, mais pour assurer la compatibilité avec les programmes AMOS, AOZ supporte toujours ces instructions que vous pourrez trouver dans les programmes des pionniers, nombre de démos et de jeux qui les utilisent et tant de programmeurs qui les aiment encore.

Il existe deux types d'objets mobiles, appelés « Bobs » et « Sprites ». Occupons-nous d'abord de Bobs.

Un Bob peut être déplacé à l'écran et il n'y a pas de limite au nombre de Bobs que vous pouvez utiliser, en dehors de la quantité de mémoire disponible bien sûr. En d'autres termes, vous pouvez avoir des milliers d'entre eux exécutant des pirouettes à l'écran en même temps. Vous pouvez contrôler un Bob, le suivre et même lui donner des caractéristiques particulières s'il entre en collision avec autre chose ou entre dans une zone d'écran spéciale. Vous pouvez animer un Bob pour faire des personnages de dessins animés en mouvement, en fait tout ce dont votre imagination peut rêver.
Chez papy AMOS un Bob était stocké dans l'écran, il pouvait manger beaucoup de mémoire, il y avait donc un type alternatif d'objet appelé Sprite.

Un Sprite est également un objet graphique que vous pouviez manipuler totalement, et qui était complètement indépendant de l'écran. Dans le monde des ordinateurs des années 80 les Sprites étaient manipulés directement par le « matériel », et l'instruction Bob manipulée par le logiciel. Aujourd'hui, cette différence

n’existe plus et les limites mémoires sont quasi inexistantes. Mais on nomme toujours Sprite un objet graphique en mouvement.

Comme on peut s’y attendre, AOZ offre des tas de tours de magie pour les mouvements intelligents. Le chapitre suivant poursuivra sans vergogne, avec des exemples utilisant l'instruction Bobs.

À L’INFINI ET AU-DELÀ
Avec toutes ces merveilles pour créer du mouvement et de l’animation : Actor, Sprite et Bob, vous aurez probablement envie de les placer dans un univers approprié. Bonne nouvelle, vos univers peuvent être bidimensionnels, tridimensionnels et avec des effets comme le défilement parallaxe (ou scrolling) avec des couches de graphismes qui se déplacent à des vitesses différentes pour créer un effet de perspective.

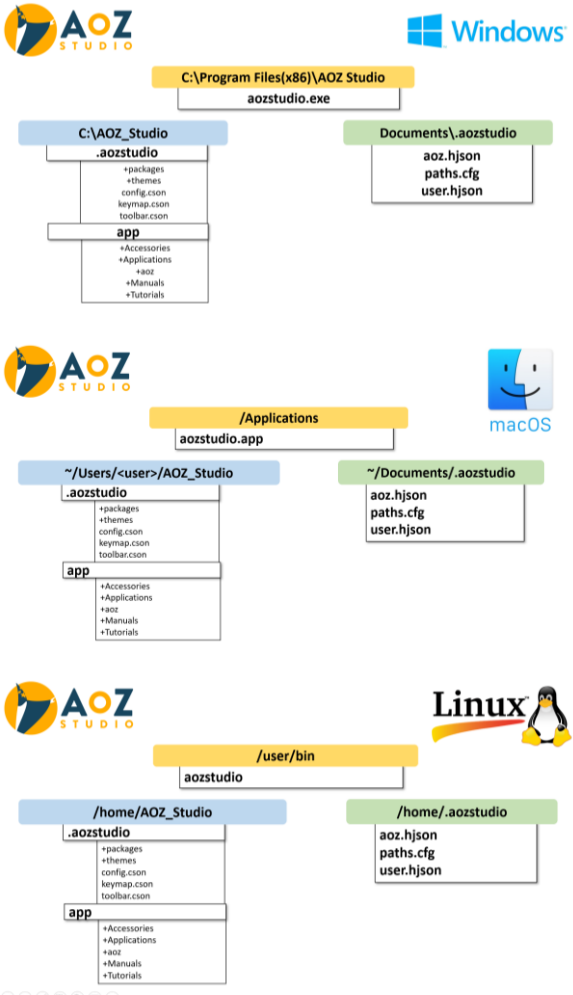
AMAL

AMAL est un petit langage au sein d'AMOS qui supportait également la souris Amiga. Maintenant, il est préférable d'utiliser l'instruction Actor. Discuter d'AMAL dépasse le cadre de ce chapitre, veuillez consulter les chapitres concernant AMAL dans l'ancien manuel AMOS Amiga pour plus d'informations.

29. INSTALLATION D'AOZ STUDIO

AOZ Studio est compatible avec Windows, Mac OS et la plupart des versions de linux. Nous prévoyons d'étendre sa disponibilité, par exemple aux Raspberry.

Voici où se trouvent les fichiers d'AOZ Studio :



30. COMMANDES CLAVIER

Voici une liste des commandes de raccourcis clavier :

MAC	PC / LINUX	LES RACCOURCIS CLAVIER LES + UTILES
		Spécifique à AOZ Studio
F1		Execute le programme dans le navigateur
F2		Execute le programme dans l'AOZ Viewer
F4		Ligne suivante ayant une erreur
Shift F4		Ligne précédente ayant une erreur
F5		Aide sur l'instruction sous le curseur
F6		Ouvre le manuel sur l'instruction sous le curseur
Ctrl Shift C		Messages d'erreur
Ctrl C		Stop le programme AOZ
		Spécifique au navigateur
⌘ Ctrl F	**F11**	Navigateur en plein écran On/Off
F11		Desktop On/Off (Mac slmnt)
F12		Widgets On/Off (Mac slmnt)
⌘ Opt I	**Ctrl Shift I**	Outils developpeur du navigateur
⌘ H		Cache AOZ Studio On/Off (Mac slmnt)
⌘ Opt H		Cache les autres On/Off (Mac slmnt)
⌘ Q	**Ctrl Q**	Quitte AOZ Studio
Ctrl Shift S	**Ctrl Shift S**	Save As...
⌘ Shift T		Ouvre le dernier item (Mac slmnt)
⌘ K ⌘ W	**Ctrl K Ctrl W**	Ferme le panneau
		Cherche et remplace
⌘ F	**Ctrl F**	Cherche
⌘ G	**Ctrl G**	Cherche le suivant
⌘ Opt F		Remplace
		Curseur
⌘ X	**Ctrl X**	Coupe
⌘ C	**Ctrl C**	Copie
⌘ V	**Ctrl V**	Colle
⌘ Z	**Ctrl Z**	Annule
⌘ Y	**Ctrl Y**	Remet
⌘]	**Ctrl]**	Indente 1 ligne
⌘ [	**Ctrl [**	Désindent 1 ligne
⌘ A	**Ctrl A**	Selectionne tout
⌘ D	**Ctrl D**	Selectionne le mot
⌘ L	**Ctrl L**	Selectionne le mot
Ctrl K	**Ctrl K**	Coupe du curseur à la fin de la ligne
⌘ Shift Up	**Ctrl Shift Up**	Sélectionne du curseur au début du programme
⌘ Shift Down	**Ctrl Shift Down**	Sélectionne du curseur à la fin du programme
Option Shift Right	**Alt Shift Right**	Sélectionne du curseur à la fin du mot suivant
Option Shift Left	**Alt Shift Left**	Sélectionne du curseur au début du mot suivant
Shift End		Sélectionne du curseur à la fin de la ligne
Shift Up		Sélectionne du curseur à la ligne au dessus
Shift Down		Sélectionne du curseur à la ligne en dessous
Shift Right		Sélectionne 1 caractére de plus à droite
Shift Left		Sélectionne 1 caractére de plus à gauche
Shift Home		Sélectionne du curseur au 1er car. non vide de la ligne
Ctrl ⌘ Up	**Ctrl Up**	Bouge une ligne au dessus
Ctrl ⌘ Down	**Ctrl Down**	Bouge une ligne en dessous
⌘ Shift D	**Ctrl Shift D**	Duplique la ligne
Ctrl Shift K	**Ctrl Shift K**	Efface la ligne
Opt Delete	**Ctrl Delete**	Efface du curseur à la fin du mot
Opt Backspace	**Ctrl Backspace**	Efface du curseur au début du mot
⌘ J	**Ctrl J**	Joindre les lignes
⌘ K ⌘ U	**Ctrl K Ctrl U**	Majuscule
⌘ K ⌘ L	**Ctrl K Ctrl L**	Minuscule
		Curseur
⌘ Up	**Ctrl Up**	Curseur à la 1ére ligne
⌘ Down	**Ctrl Down**	Curseur à la derniére ligne
Home		Curseur au début
End		Curseur à la fin
		Regroupement de lignes de code
⌘ Opt [	**Ctrl Alt [**	Plie
⌘ Opt Shift [	**Ctrl Alt Shift [**	Plie tout
⌘ Opt]	**Ctrl Alt]**	Déplie
⌘ Opt Shift]	**Ctrl Alt Shift]**	Déplie tout
		Divers
Ctrl G		Va à la ligne (demandée)
⌘ Ctrl Space		Emoji / symboles spéciaux
⌘ {	**Ctrl PgUp**	Fichier précédent
⌘ }	**Ctrl PgDn**	Fichier suivant

31. MERCI

Merci à notre fantastique communauté d'utilisateurs. Rien n'aurait été possible sans vous au cours de ce voyage de près de 3 ans à concevoir AOZ Studio.

S'il vous plaît soyez patient. Nous travaillons dur pour améliorer ce gros machin, la route est encore longue et sinueuse. Nous espérons vraiment que vous apprécierez ce que nous avons fait jusqu'à présent et que vous imaginerez avec nous ce que nous allons faire ensuite.

Partagez vos désirs les plus fous sur la chaine Discord, https://discord.com/invite/JmFyFRA En plus de magiciens bienveillants vous nous y trouverez pour vous aider…

Découvrez la chaîne Youtube d'AOZ Studio

Jusqu'au jour où c'est vous qui aiderez les autres ; vous aurez alors officiellement le statut de Magicien. Mais si vous êtes là c'est que vous avez déjà la baguette, bravo !