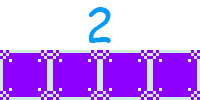
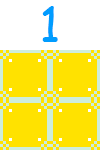
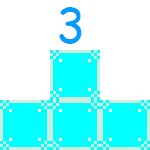
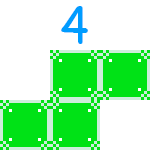
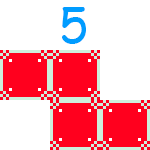
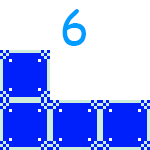
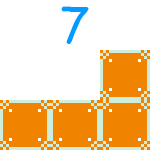
**Projet ISN : Programmation d’un jeu en Python, avec PyGame TetrISN**

Nom : SAUVAL Autres membres du groupe :  
Prénom : Danny LEDUEY Alexandre Classe : TS3 MICHELO Kévin

**I – Qu’est-ce que Tetris ?**

Tetris est un jeu simple, basé sur 7 pièces composées de 4 carrés. Les différentes sont :





(Les pièces ont été numérotées pour simplifier le repérage, notamment dans la programmation)

De manière générale, le but du jeu est de former des lignes avec des pièces qui descendent à un intervalle de temps t. Le temps t dépend du niveau. Lorsqu’une ou plusieurs lignes sont complètes, elles disparaissent et les cubes présents à l’écran descendent du nombre de lignes supprimées. Le jeu se termine lorsqu'il n'est plus possible de faire descendre des formes sur le tableau.

Dans notre TetrISN, le comptage des points se fait de la manière suivante :

* Si on appuie sur la flèche « haut » pour directement faire descendre la pièce tout en bas, on ajoute au score 5 points.
* Lorsqu’une pièce à été posé, on ajoute 20 points.
* Lorsqu’une ou plusieurs lignes est retirée, on ajoute au score 60 points, ainsi qu’un bonus égal au nombre de lignes supprimées d’un coup, multiplié par 10. Par exemple, si je supprime deux lignes : 60+1\*10 = 70 points.

**II – Comment programmer notre TetrISN ?**

* **Langage de programmation python…. :**

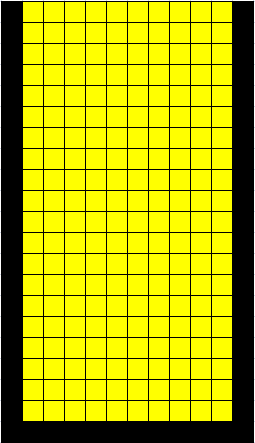
Le Python est un animal qui peut s'avérer dangereux mais aussi le nom d'un langage de programmation très puissant qui est de plus en plus utilisé pour développer des applications rapidement qui soient tout de même efficaces ! Il est très utilisé dans le monde scientifique pour sa syntaxe aérée et une vitesse d'exécution correcte.

* **…..accompagné de la librairie pygame.**

L'histoire de Pygame commence en l'an 2000, lorsque **Pete Shinners** découvre Python et SDL.

SDL, pour Simple Directmedia Library, est une bibliothèque libre multi-plateformes permettant la gestion du multimédia dans la programmation. C’est à dire la gestion de l’affichage vidéo 2D, la gestion de l’audio et la gestion des évènements (clavier, souris, manettes….). Lorsque Pete a découvert Python et SDL, il se dit qu'il serait peut-être bon de combiner les deux. Il voulait ainsi proposer aux codeurs Python une bibliothèque multimédia aussi riche que celle du C, tout en conservant ce qui fait l'âme de Python, la simplicité.

**III – Ma partie :**

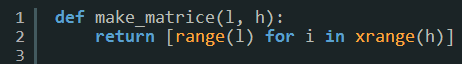
Ayant réparti le travail entre nous, j’ai eu à réaliser dans la partie programmation la zone de jeu, ainsi qu’une musique. J’ai également fait un peu de graphismes, en prenant un fond d’écran fait par Alex, pour écrire du texte dessus (fond d’écran pause, et écran après avoir perdu).

Pour cela, je me suis renseigné davantage sur le jeu Tetris, notamment en en testant quelques uns.

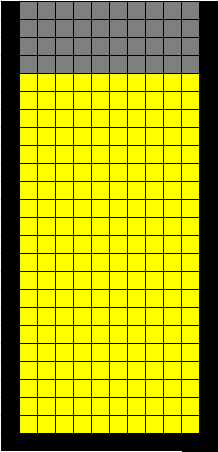
1. **La zone de jeu Tetris :**

La zone de jeu de Tetris fait 10x20. Cela signifie qu’il y a 20 lignes de 10 carrés alignés. Pour programmer cela, j’ai eu l’idée d’utiliser à un tableau ou une matrice à deux dimensions, qui contiendrait 20 lignes et 10 colonnes. Cependant, il ne fallait pas oublier les bords, qui ajoute de chaque côté un cube, et un autre en bas. On arrive donc à 20 colonnes et 12 lignes.

Mais pour le nombre de lignes, ça n’est pas tout. On va devoir faire apparaitre des pièces qui partiront de haut pour aller en bas. Mais on ne doit pas faire apparaitre la totalité de la pièce. On doit la faire apparaitre de manière progressive. Sachant que la hauteur maximale d’une pièce est celle de la barre (2), on doit ajouter 4 lignes. Ce qui donne un tableau de 12x25. Pour créer simplement un tableau à deux dimensions, j’ai trouvé sur un sujet de « Open Classroom » une fonction qui permet de créer une matrice en donnant en paramètre à cette fonction la largeur et la hauteur.

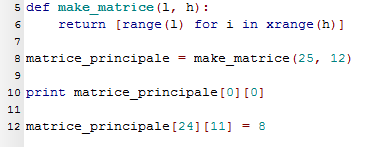


Je n’ai pas compris comment fonctionne cette fonction, mais je n’ai pas pu trouver plus simple et cette méthode fonctionne très bien, alors inutile de s’en priver…..

Le lien vers le sujet est : [mettre-un-tableau-a-deux-dimensions-a-zero](http://fr.openclassrooms.com/forum/sujet/mettre-un-tableau-a-deux-dimensions-a-zero-77265#.U4HnNfl_uSo)

On a donc enfin un tableau 25\*12 :

Pour l’utiliser c’est très simple.



Il suffit d’entrer comme première valeur la ligne, et comme seconde la colonne. On peut de cette manière lire ou écrire dans le case (x,y) du tableau.

1. **Avec quoi remplir le tableau ?**

Pour se repérer facilement, j’ai appelé le tableau « matrice\_principale ».

Pour un traitement simple, j’ai choisis de remplir le tableau avec des nombres entiers. Pour cela, lorsqu’on a dans le tableau un :

* + 0 : Il n’y a rien.
  + 1 : Cube du Cube.
  + 2 : Cube de la Barre.
  + 3 : Cube du T.
  + 4 : Cube du S.
  + 5 : Cube du Z.
  + 6 : Cube du J.
  + 7 : Cube du L.
  + 8 : Cube d’un bord.

Au départ du jeu, on aura donc les colonnes 0 et 11 qui seront à 8, ainsi que la ligne 24. Sachant que le comptage commence à 0. Alexandre s’est occupé d’ajouter ces 8 dans le tableau, à l’aide de boucles « for ». Pour traiter le « comment remplir le tableau ? » il faut d’abord passer au 3.

1. **Comment créer les pièces du jeu ? :**

Alexandre s’est occupé de dessiner les petits cubes à la base de chaque pièce. Les couleurs, propres à chaque pièce, sont celles de la page 1.

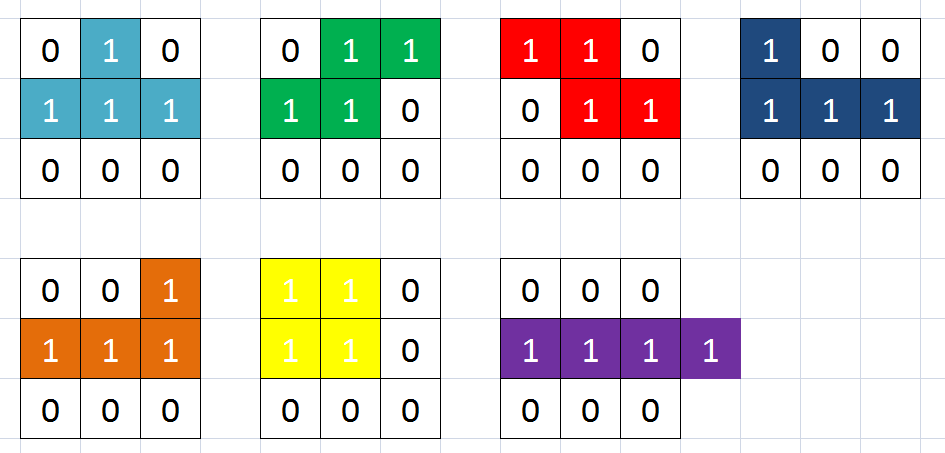
J’ai donc du trouver un moyen de programmer les pièces. Pour cela, j’ai commencé par réfléchir comment j’allais créer les formes. J’ai décidé d’utiliser un tableau d’une « profondeur » de 3.

G:\ISN\Python\Projet ISN\Danny\Dossier\ligneforme.PNG

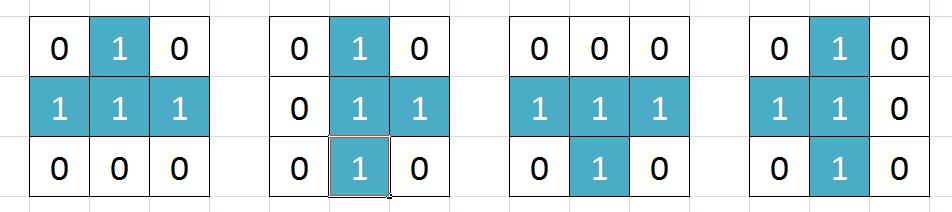
Avec toutes ces parenthèses, ça semble compliqué, mais c’est simple.

Chaque pièce à plusieurs formes, 4 étant le nombre maximal de formes différentes. Pour simplifier la programmation, j’ai décidé de faire comme si toutes les pièces avaient toutes 4 formes différentes. Ca n’est pas vrai, mais cela permet d’économiser des lignes de calculs.

Pour créer ce tableau, on a besoin de définir 4 formes. Chaque forme doit contenir x lignes et y colonnes. Pour simplifier la programmation, j’ai de nouveau décidé de généraliser un cas pour toutes les pièces. Le cas général a été de créer dans chaque forme une matrice 3x3. Le souci est que pour la barre, on a une hauteur ou largeur de 4, ce qui dépasse la matrice. Il faudra donc traiter ce cas à part.



De cette manière, toutes les formes vont pouvoir être gérées de la même manière et les rotations seront simples à gérer.



Sachant que la présence d’un cube est signalée par un « 1 », pour les différents tests de collisions ou encore pour modifier la matrice\_principale, il suffira de tester si le cube (x,y) est à 1 ou non. Donc la présence de « 0 » n’est pas gênante.

Pour en revenir à la ligne de déclaration des formes :

G:\ISN\Python\Projet ISN\Danny\Dossier\ligneforme.PNG

L’utilisation est la suivante :

Le tableau est de la forme suivante :

objet\_cube\_forme = (F1,F2,F3,F4)

F1 est la première, F2 la seconde, jusqu’à la quatrième.

Pour toutes les pièces autres que la barre, chaque F est composé de 3 lignes et 3 colonnes.

Fx = (X1, X2, X3) 🡪 Le nombre de X défini le nombre de lignes.

Xx = (Y1, Y2, Y3) 🡪 Le nombre de Y défini le nombre de colonnes.

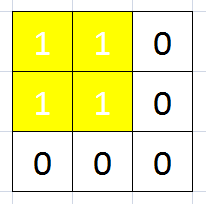
Ainsi :

Une ligne, Deux colonnes

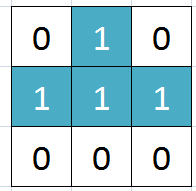
Une forme

G:\ISN\Python\Projet ISN\Danny\Dossier\ligneforme.PNG

En positionnant chaque ligne à sa place, on obtient bien un cube :



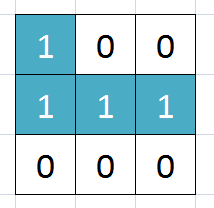
1,1 1,1

Sachant que lorsqu’ils sont « tout à droite », ou « tout en bas », les zéros sont inutiles, le temps que le placement est correct. Par exemple, pour la pièce, 3, le T :

0,1 1,1,1

Donne bien :

Alors que si on enlève le zéro de gauche, on aura en première colonne un 1, donc :

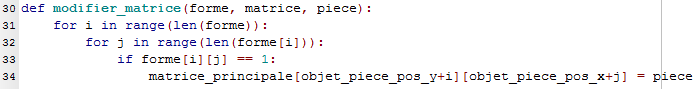


1. **Comment remplir la matrice ? :**

On remplit la matrice avec le numéro de la pièce, stocké dans la variable « piece\_actuelle ».

Pour remplir la matrice, c’est très simple, on crée une « double boucle for ». C'est-à-dire que l’on va créer une première boucle for qui va permettre de parcourir les lignes de la forme, puis à l’intérieur une autre boucle for qui va parcourir les colonnes de chaque ligne.

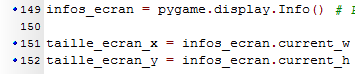
A chaque fois, on testera : « si la valeur [i][j] de la forme vaut 1, alors la valeur [objet\_piece\_pos\_y+i][objet\_piece\_pos\_x+j] vaut 1 ».



1. **Comment gérer la descente des pièces jusqu’en bas, puis passer à la pièce suivante en laissant la pièce précédente en bas ?**

Tout d’abord, il faut connaître la hauteur que nous offre l’écran. Pourquoi ? Et bien parce que c’est cette hauteur qui va définir la taille que doivent avoir les cubes pour que les 21 (il y a 25 lignes mais les 4 premières sont cachées) à afficher en hauteur rentrent sur l’écran, et c’est aussi cette valeur qui va définir l’écart entre chaque coin (en haut à gauche) des cubes.

Pour cela, il faut récupérer les informations relatives à la résolution offerte par l’écran.



On utilise donc une fonction de pygame, qui nous permet de récupérer dans les variables de notre choix, (ici taille\_ecran\_) la hauteur (.current\_h) et la largeur (.current\_w) de l’écran.

Pour connaître la hauteur de chaque cube, il suffit donc de diviser la hauteur par 21. Ce qui donne la taille que doit faire un cube. J’ai donc utilisé cette variable pour redimensionner les cubes (a) et ensuite pour gérer l‘affichage des pièces (b).

1. Redimensionnement des cubes :

G:\ISN\Python\Projet ISN\Danny\Dossier\smotth.PNG

Cette fonction permet de redimensionner un objet de surface, selon deux paramètres, la hauteur et la largeur que l’on a déterminé précédemment. Le soucis est que ces deux paramètres doivent être des entier, et le calcul précédent (taille\_ecran\_y/21) ne retourne pas forcément un entier. Il a donc fallu remplacer le simple « / » par un double « // » pour ne garder que la partie entière de la division. La fenêtre du jeu ne remplira donc pas entièrement la hauteur de l’écran, mais ça n’est pas très grave.

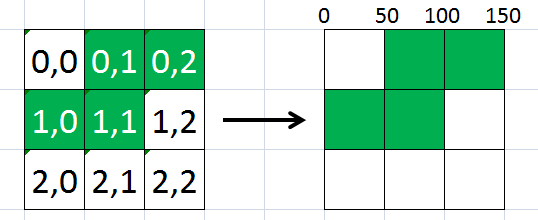
1. Affichage des pièces :

Pour l’affichage des pièces. Il suffit de coller chaque cube à une position égale :

* en x : « numéro de la colonne » fois « la taille d’un cube ».
* en y « le numéro de la ligne » fois « la taille d’un cube ».

Il suffit simplement d’utiliser une boucle for avec comme paramètre la ligne. Dans cette boucle for, on met une autre boucle for pour les colonnes. C’est une sorte de « double boucle for ».

Prenons un exemple avec une matrice 3x3.



On multiplie le nombre de la colonne par la taille d’un cube. Ici j’ai pris 50. Ici, la première valeur de la matrice est 0 comme il ne faut rien afficher, donc on ne fait rien. Ensuite, on a un 1, il faut afficher quelque chose, que l’on affiche à (0\*0, 1\*50). Et ainsi de suite.

1. **Gérer les déplacements :**

Les déplacements possibles pour chaque pièce sont : « gauche », « droite », « bas ». Il faut donc gérer les collisions pour ces trois déplacements.

Pour cela, j’ai simplement créé des fonctions :

* bouger\_possible\_gauche()
* bouger\_possible\_bas()
* bouger\_possible\_droite()

Ces fonctions prennent deux paramètres :

* La forme actuelle de la pièce
* La matrice

Pour expliquer leur fonctionnement, pour l’exemple du cas où l’on veut aller à gauche :

La fonction « bouger\_possible\_gauche() » considère tout d’abord que le déplacement est possible, avec une variable « bouger = 1 ». Ensuite, la fonction va à l’aide d’une « double boucle for » (comme tout à l’heure), tester si la valeur [i][j] de la forme est égale à 1, et si c’est le cas, elle vérifie si la valeur dans la matrice à la position [objet\_piece\_pos\_y+i][objet\_piece\_pos\_x-1+j] est égale à 1. Si c’est le cas, bouger = 0. La fonction renvoie la variable « bouger », et en fonction de si on a un 1 ou un 0, on peut effectuer un non le déplacement.

Remarque : Pour vérifier la valeur de la matrice, les coordonnées utilisés ont été : [objet\_piece\_pos\_y+i][objet\_piece\_pos\_x-1+j]. Pour aller à gauche, on retire forcément 1 à la valeur de la colonne. De même, pour aller à droite on ajoute 1, et pour aller en bas, on ajoute un à la valeur de la ligne.

Ces fonctions vont donc être utilisées pour :

* Permettre d’ailler à droite ou à gauche avec les flèches
* Permettre d’aller en bas ave la flèche « bas »
* Permettre de descendre une pièce avec le temps.

La flèche « haut » du clavier permet de descendre la pièce tout en bas. Pour cela, il m’a simplement fallu créer une boucle :

Tant que bouger\_possible\_droite() est vrai :

Position de l’objet += 1

1. **Gérer les rotations :**

La gestion des rotations s’est fait de la même manière que pour gérer les déplacements. Sauf que cette fois si, on n’ajoute/ne retire pas 1 à la valeur de la ligne ou de la colonne. Pour les rotations, on envoie simplement à la fonction la forme suivant la forme actuelle, et on vérifie s’il y a des conflits. S’il n’y en a pas, on n’effectue pas la rotation, en revanche s’il y a un conflit, on effectue trois autres tests :

* Un pour savoir si en se déplaçant à droite, il y a un conflit.
* Si le précédent est faux, on test à gauche.
* Si le précédent est faux, on test en se déplaçant vers le haut s’il y a un conflit, et s’il n’y en a pas, on bloque la rotation.

Après le résultat de ces tests, on effectue un déplacement vers la droite, ou un vers la gauche, ou un vers le haut, ou bien on ne fait rien du tout.

1. **Création et intégration d’une musique à notre jeu :**

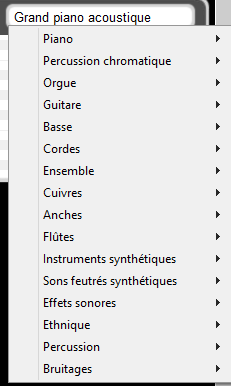
Pour créer la musique, je voulais au départ la jouer moi-même au piano, mais par rapport à l’enregistrement ça aurait été compliqué. Je me suis donc renseigné sur la MAO, Musique Assisté par Ordinateur, en allant sur Wikipedia. La MAO est donc la création de musique sur un ordinateur. Je savais déjà cela, et en allant sur wikipedia, je cherchais surtout de bons logiciels LIBRES qui me permettraient de faire de la MAO. Sachant que j’avais déjà sur mon ordinateur Guitar Pro 6, mais étant un logiciel payant je ne l’ai pas utilisé.

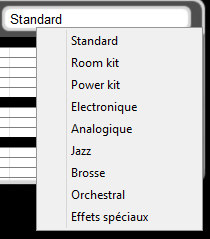
Dans la partie regroupant divers logiciel de MAO j’ai choisi le premier de la liste.

[MAO : Logiciels Libres](http://fr.wikipedia.org/wiki/Musique_assist%C3%A9e_par_ordinateur%23Logiciels_libres)

Ou du moins le deuxième comme le premier est incompatible avec Windows.

Le logiciel que j’ai choisi s’appelle « Aria Maestosa ». Ce logiciel est plutôt un séquenceur MIDI, qui permet de créer, éditer et jouer ces fichiers.

L’avantage de ce logiciel est qu’il permet de créer plusieurs pistes musicales de manière simple et intuitive. De plus il gère de nombreux sons.

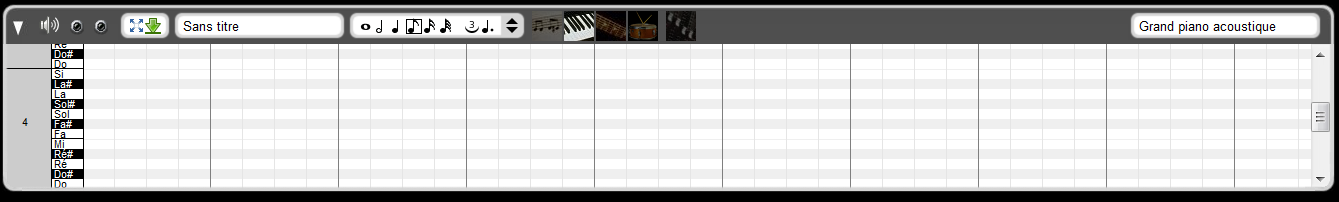


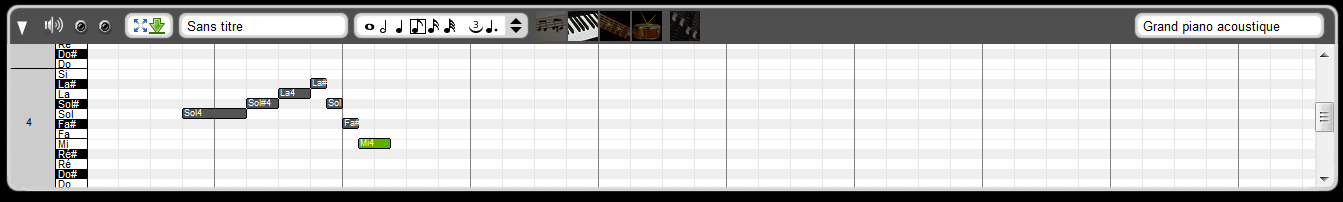
Instruments divers

Batterie

Pour créer ma musique, j’ai utilisé notamment des sons de piano, basse, batterie, strings, et vocals.

Pour la création de la musique, il suffit de créer chaque note avec la souris :





J’ai donc recréé la musique Tetris en reprenant les notes que j’ai trouvées sur internet. Quand à la batterie, j’ai improvisé. Pour les autres sons, j’ai simplement copié-collé les notes de la mélodie principale, en changeant simplement le son. Pour ce qui est de la ligne de basse, n’étant pas bassiste du tout, j’ai téléchargé un fichier midi sur internet sur lequel j’ai repris la ligne de basse (que j’ai un peu modifié).

<http://www.midiarchive.co.uk/download/Games/Tetris/Tetris%20-%20Song%20A.mid>

Pour ce qui est des droits d’auteur liés à la musique, c’est assez délicat de trouver une réponse parfaitement claire. Il n’y a aucun droit sur le nom du jeu « Tetris », que nous n’avons de toute façon pas utilisé. En revanche il y en a sur la musique. Mais, dans le post de « Silencement » sur open classroom dans le sujet : [Question sur les droits d'auteur](http://fr.openclassrooms.com/forum/sujet/question-sur-les-droits-d-auteurs-73052#.U4Xce_l_uSo), il y a un article de Wikipédia traitant de ce sujet. Au bout de 70 ans après la mort de l’auteur, une œuvre tombe dans le domaine public. Tetris date de 1989, ce qui ne fait pas 70 ans. Sauf qu’Alexei Pajitnov, « créateur » de Tetris, a utilisé pour Tetris une musique russe « Korobeïniki (Коробе́йники) » qui date de 1861. Ce qui fait bien plus de 70 ans.

Mais il faut savoir que la réponse n’est pas claire, donc si l’on voulait diffuser au « grand public » notre super jeu, il vaudrait mieux changer la musique. Surtout que si Tetris voulait nous attaquer, je ne pense pas qu’on fasse le poids.

Pour intégrer la musique au jeu, j’ai eu beaucoup de mal. D’après la documentation de Pygame, on ne peut utiliser que des fichiers OGG. Parfois les MP3 fonctionnent, mais j’ai testé et ça n’a pas été le cas.

L’ennui est que Aria Maestosa est un séquenceur midi, et ne permet de gérer que des fichiers midi et non OGG. J’ai donc du convertir le fichier midi en OGG. Le problème a été la conversion. J’ai essayé de nombreux logiciels permettant cette conversion, mais à chaque fois, que ce soit de midi vers MP3 ou de midi vers OGG, les pistes n’étaient plus synchronisées, le volume de chaque piste n’était pas constant, ce qui donnait un résultat horrible. Le seul moyen que j’ai trouvé a été d’utiliser le logiciel libre Audacity, et de régler le logiciel sur écoute. De cette manière, Audacity prélève le signal envoyé aux haut-parleurs, et permet ensuite d’enregistré le son obtenu en OGG.

