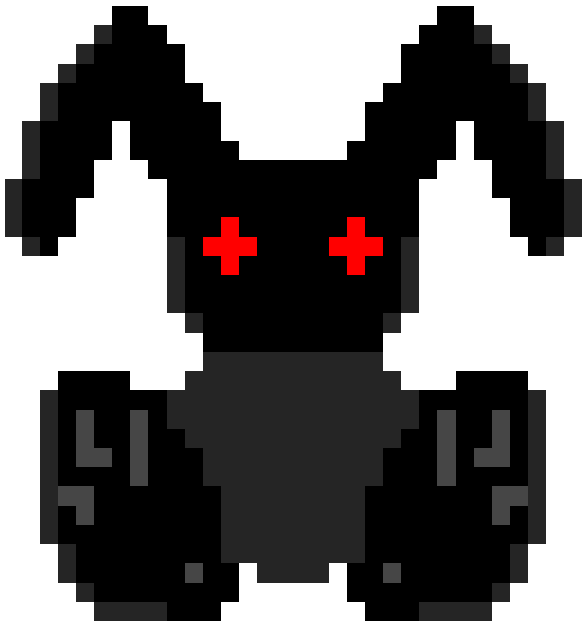

LAPINS NOIRS

FRITURE

Affichage aléatoire

- La Caverne aux Lapins Noirs -

Ce document est strictement personnel et ne doit en aucun cas être diffusé.

INDEX

01 – Avant-propos

Votre travail doit être rendu via le dossier **~/colle/friture/** dans votre espace personnel.

Si vous faites erreur et que le dossier que vous utilisez pour votre rendu est différent, vous ne serez pas évalué faute d'avoir pu trouver votre travail.

Ce travail est a effectuer en binômes aléatoires. Vous pouvez communiquer avec votre binôme mais pas avec les autres équipes. Votre équipe doit être l'auteur de votre travail. Utiliser le code d'une autre équipe, c'est **tricher**. Et tricher annule **toutes** les médailles que vous avez reçu sur l'activité. La vérification de la triche est réalisée de la même manière que la correction : de manière **automatique**. Prenez garde si vous pensez pouvoir passer au travers.

Votre rendu doit respecter **strictement** l'ensemble des règles suivantes :

- Il ne doit contenir **aucun** fichier objet. (*.o)
- Il ne doit contenir **aucun** fichier tampon. (*~, #*#)
- Il ne doit pas contenir votre production finale (programme ou bibliothèque)

La présence d'un fichier interdit mettra immédiatement fin à votre évaluation.

Votre programme doit respecter les Tables de la Norme dans leur intégralité. Vous êtes invité à les observer depuis **l'Infosphère**. Elles sont disponibles comme ressource de cette activité.

02 – Fonctions autorisées

La bibliothèque logicielle venant avec le C est vaste et disponible. La LibLapin, que vous utilisez dans vos projets multimédia, est également vaste… Cependant nous avons fait le choix de vous interdire leurs utilisation intégrales, afin de vous amener progressivement à reprogrammer vous même ses fonctionnalités les plus utiles.

L'utilisation d'une fonction interdite est assimilée à de la triche. La triche provoque l'arrêt de l'évaluation et la perte des médailles.

Vous n'avez le droit d'utiliser aucune fonction issue de la LibC ou de la LibLapin à l'exception de celles que nous vous autoriserons explicitement.

Pouvoir utiliser une fonction ne signifie pas nécessairement que celle-ci soit utile à votre cas.

Pour cette activité, issu de la LibC, vous n'avez le droit qu'à la liste suivante :

- open
- close
- read
- srand
- cos
- atan2
- time
- write
- alloca
- atexit
- rand
- sin
- sqrt

Remarquez bien l'absence de **malloc** et de **free**. En effet ils sont **interdits** ! Issu de la LibLapin, vous avez le droit aux fonctions suivantes :

- bunny_start
- bunny_stop
- **bunny_malloc**
- **bunny_free**
- bunny_new_pixelarray
- bunny_delete_clipable
- bunny_blit
- bunny_display
- bunny_set_*_function
- bunny_set_*_response
- bunny_loop
- bunny_create_effect
- bunny_compute_effect
- bunny_sound_play
- bunny_sound_stop
- bunny_delete_sound
- bunny_open_configuration
- bunny_delete_configuration
- bunny_configuration_getf
- bunny_configuration_setf
- bunny_configuration_*
- bunny_set_memory_check
- bunny_release
- bunny_usleep

La suite sur la page d'après.

Pour utiliser **bunny_malloc**, vous pouvez soit programme directement avec, soit en utilisant le modèle de projet qui vous a été transmis, mettre 1 dans la variable de Makefile **BMALLOC**, qui transformera **malloc** en **bunny_malloc**.

Vous appellerez **bunny_set_memory_check** au début de votre fonction **main** de sorte à provoquer une vérification de vos allocations à la fermeture du programme.

bunny_malloc, par défaut, limitera votre consommation de RAM à 20Mo.

Pour information :
Une image en 1920*1080 fait environ 8Mo.

Une musique en 44kHz de 1 minute en stéréo fait environ 10Mo.

Vous devrez donc disposer d'une discipline de fer avec vos allocations... et probablement trouver des compromis.

L'utilisation de **bunny_malloc** parfois **cachera** des erreurs dans votre programme, et parfois en **révélera** : son principe d'allocation était différent de **malloc**, il sera parfois plus « fort » ou plus « faible ». Ne vous mentez pas à vous en disant « l'utilisation de **bunny_malloc** fait planter mon programme », ce n'est pas **bunny_malloc**, c'est *vous*.

N'hésitez pas à l'activer, à le désactiver (Et lorsqu'il est désactivé, à utiliser valgrind). Et n'oubliez pas que désormais, votre demande de RAM a de véritables chances d'échouer. Car exploiter aussi peu de RAM, cela va très vite…

Dans votre rendu, il ne devra pas y avoir la moindre trace de malloc.

03 – Méthode de construction

Il va vous être demandé de rendre des programmes. Il vous sera toujours demandé de fournir un **dossier** pour l'exercice le requérant. Un **Makefile** vous sera également demandé. Le **nom du programme** de sortie vous sera précisé à chaque fois. Un Makefile incorrect, un mauvais nom de programme, et votre correction n'aura pas lieu…

Des fonctions peuvent être demandés explicitement, mais dans le cadre de cette activité, cela ne signifie rien d'autre qu'une obligation de la fournir pleinement fonctionnelle : elle sera simplement testée séparément du reste, bien qu'il vous soit demandé de la fournir en même temps que le reste.

Votre compilation devra toujours comporter les options **-W**, **-Wall** et **-Werror**.

Dans le cadre de la programmation multimédia, le système de correction établira toujours la variable d'environnement **BMALLOC** à 1. Si vous utilisez le modèle de projet, cela provoquera l'utilisation de **bunny_malloc** dans votre bibliothèque personnelle comme dans votre projet rendu.

L'objectif de cette organisation est de permettre une correction sans pour autant vous empêcher de profiter de votre travail.

04 – noise

Mettre dans le dossier noise/

Programmez la fonction suivante :

```
void std_noise(t_bunny_pixelarray *picture);
```

Cette fonction, pour l'instant, remplit l'image envoyée en paramètre de pixel aléatoires. Vous commencez par le pixel situé en haut à gauche, c'est à dire le même que celui située exactement au début de **picture→pixels**.

Pour générer une valeur aléatoire, utilisez la fonction **rand**. Le remplissage se fera en plaçant le résultat de cette opération dans chaque pixel : **rand() | BLACK**.

Dans votre main, vous devez utiliser **srand(time(NULL))** afin de générer des nombre aléatoires toujours différent : la fonction **srand** initialise le générateur d'aléatoire à partir du paramètre. La fonction **time** renvoi le nombre de seconde écoulée depuis le 1er janvier 1970. En conséquence, à chaque seconde passée, les nombres aléatoires générés par **rand** changeront.

Vous fournirez en plus de cette fonction **std_noise** (situé bien évidemment dans **src/noise.c**), une fonction main située dans **src/main.c**. Votre fichier **include/noise.h** permettra d'assurer une compilation sans mise en gardes.

05 – noise_clip

Mettre dans le dossier noise/

Modifiez la fonction précédente de manière à prendre en compte les attributs situés dans le **t_bunny_clipable** de **t_bunny_pixelarray** limitant la portée de certaines fonctions. Les attributs en questions sont **clip_x_position**, **clip_y_position**, **clip_width** et **clip_height**.

Ces quatre attributs indiquent le coin supérieur gauche du rectangle, la largeur et la hauteur du rectangle qui seront impacté dans l'image par la fonction **std_noise**.

06 – noise_gray

Mettre dans le dossier : noise_gray/

Programmez la fonction suivante :

```
void std_noise_gray(t_bunny_pixelarray *picture);
```

Cette fonction fonctionne de la même manière que std_noise à l'exception du fait qu'elle ne produit que du bruit gris. Le gris est une couleur qui a la particularité d'avoir la même valeur pour chaque composante de couleur.

Vous pouvez utiliser et devriez utiliser t_bunny_color.

09 – Flou

Mettre dans le dossier : blur/

Maintenant, programmez la fonction suivante :

```
t_bunny_pixelarray *std_blur(t_bunny_pixelarray *picture,
                             int                neighbour);
```

Cette fonction génère une nouvelle image contenant, pour chaque pixel de celle ci, la moyenne des pixels situés dans picture de -neightboor à +neightboor autour du pixel dans la largeur et la hauteur.

Le flou se calcule composante par composante de couleur et non comme étant un unique entier. Vous pouvez et devriez utiliser t_bunny_color.