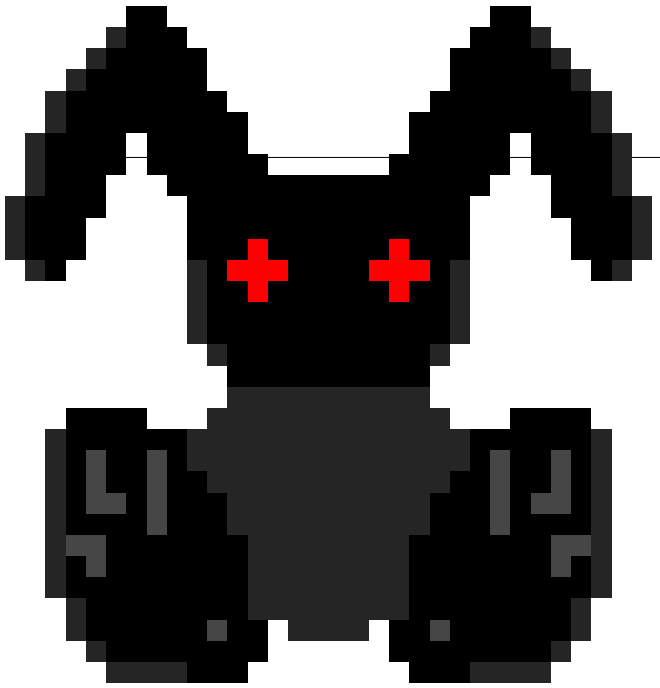
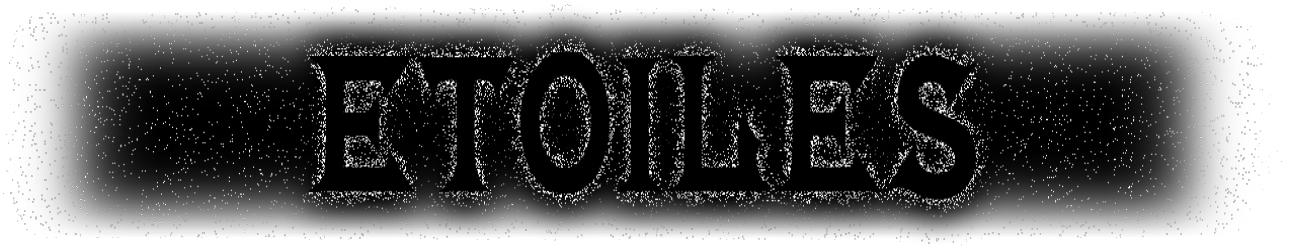


LAPINS NOIRS

- La Caverne Aux Lapins Noirs -



Asynchronisme

Vous allez réaliser votre premier logiciel dynamique

- La Caverne Aux Lapins Noirs -

INDEX

01 – Détails administratifs

Votre travail doit être rendu via le dossier **~/tp/etoiles/**.

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 seul. Vous pouvez bien sur échanger avec vos camarades, néanmoins vous devez être l'auteur de votre travail. Utiliser le code d'un autre, 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
- 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 – Étoiles

Vous allez programmer un logiciel affichant des étoiles allant de la droite de l'écran vers la gauche à des vitesses différentes. Nous allons donc avoir besoin de trois fonctions : une d'initialisation, une pour dessiner les étoiles et une pour les faire avancer.

Pour commencer, définissez ce qu'est une étoile. Nous utiliserons la structure suivante :

```
typedef struct      s_bunny_star
{
      int           y;
      float         x;
      float         x_speed;
}                   t_bunny_star;
```

Placez cette structure dans le fichier stars.h dans include/. Vos autres fichiers seront dans src/.

Cette structure définie la position (x, y) de l'étoile ainsi que sa vitesse sur l'axe x : x_speed. Remarquez l'utilisation de float : la vitesse peut en effet être inférieure à 1 sans être égale à 0. Pour représenter cette vitesse, il faut donc utiliser un nombre à virgule. De la même façon, nous utiliserons pour x un nombre à virgule, car cette position étant augmenté de x_speed, cet attribut a besoin de représenter des valeurs très petit. En effet, un nombre entier augmenté de moins de 1 ne change pas de valeur...

Programmez la fonction suivante :

```
void      std_init_stars(t_bunny_pixelarray  *pix,
                         t_bunny_star        *stars,
                         unsigned int        nbr_stars);
```

Cette fonction initialise toutes les étoiles à des positions aléatoires à l'aide de la fonction rand. Elle calcule également une vitesse aléatoire comprise entre 0 et -5. stars est un tableau d'étoile et nbr_stars sa longueur. Le tableau stars proviendra du main de test.

L'image pix est passé en paramètre de manière à ce que les valeurs valides pour les positions soient connues. Toutes les étoiles doivent être dans l'image.

Programmez la fonction suivante :

```
void std_draw_stars(t_bunny_pixelarray  *pix,
                    t_bunny_star        *stars,
                    unsigned int        nbr_stars);
```

Cette fonction dessine les étoiles dans l'image. Elle ne dessine rien d'autre ! **Rien**.

Programmez la fonction suivante :

```
void std_move_stars(t_bunny_pixelarray  *pix,
                    t_bunny_star        *stars,
                    unsigned int        nbr_stars);
```

Cette fonction ajoute à la position en **x** de chaque étoile leur vitesse **x_speed**.

Si une étoile sort de la fenêtre, sa position en **x** est réinitialisé à la largeur de **l'image - 1**.

Ces trois fonctions sont à rendre.
La suite du TP est à faire, mais ne sera pas évaluée automatiquement.
Ne rendez pas de fonction main.

07 – « Asynchrone »

Que signifie « asynchrone » ? Dans notre contexte, il s'agit d'actions ne se déroulant pas en même temps mais les unes après les autres. Le rafraîchissement de l'écran et la captation des événements utilisateurs (clavier, souris) donnant l'impression que tout est synchrone : que tout se passe en même temps.

Comment faire ?

La **LibLapin** dispose d'un ensemble de fonctions permettant de fonctionner de manière asynchrone. La fonction **bunny_loop** est la pierre angulaire de son système : elle nécessite la fenêtre, une fréquence de fonctionnement ainsi qu'un paramètre libre.

Cette fréquence de fonctionnement est le nombre d'appel moyen garanti à la fonction passée en paramètre à **bunny_set_loop_main_function**. On appellera cette fonction la **boucle principale**.

La fonction **bunny_set_display_function** sert à définir une fonction qui servira au dessin. On l'appelle **fonction d'affichage**. Cette fonction est appelée après la boucle principale et ne *l'est que si celle-ci a été appelée*. A quoi cette fonction sert-elle ? En cas de ralentissement, plusieurs appels à la boucle principale peuvent être faits. Dans ces cas là, un unique appel à la fonction d'affichage est réalisé afin d'économiser le plus possible de temps machine et ne pas afficher des données qui seront immédiatement écrasé… Et oui, c'est ce mécanisme qui fait que les jeux peuvent « ramer » au lieu de ralentir.

Il existe énormément de fonctions servant à répondre à des événements utilisateurs ou machine : une touche pressée sur le clavier, un mouvement de la souris, un bouton d'une manette de jeu… une commande réseau, un déclencheur temporel, etc.

Ces fonctions ont toutes le même format de nom : bunny_set_*_response, où * est a remplacer par le type d'événement. Voici quelques exemples :

- bunny_set_key_response
- bunny_set_move_response
- bunny_set_click_response

Écrivez une fonction **main** qui initialisera les étoiles, établira la fonction de boucle principale, la fonction d'affichage ainsi qu'une fonction de réponse au clavier qui quittera le programme si l'on appui sur la touche d'échappement. La fonction d'affiche s'occupera de noircir l'image avant d'appeler la fonction dessinant les étoiles. La fonction de boucle principale fera avancer les étoiles.

Vous devriez voir un champ d'étoile défilant de droite à gauche avec une légère impression de profondeur s'afficher.

08 – Mise en beauté

Nous allons rendre votre programme plus élégant à l'exécution.

Au lieu de mettre en noir complet l'image dans la fonction d'affichage, parcourez l'image pixel par pixel et pour chaque composante de couleur valant plus de 0, retirez une quantité arbitraire, par exemple, 50.

De cette manière il faudra plusieurs tour de boucle principal pour effacer complètement une étoile, elles auront donc une traînée derrière elle.