

# Programmation concurrente

## TP2 : Le bandit manchot

### Objectif

Le but de ce travail est d'implémenter le jeu du bandit manchot en faisant intervenir quelques principes de concurrence.

### Description

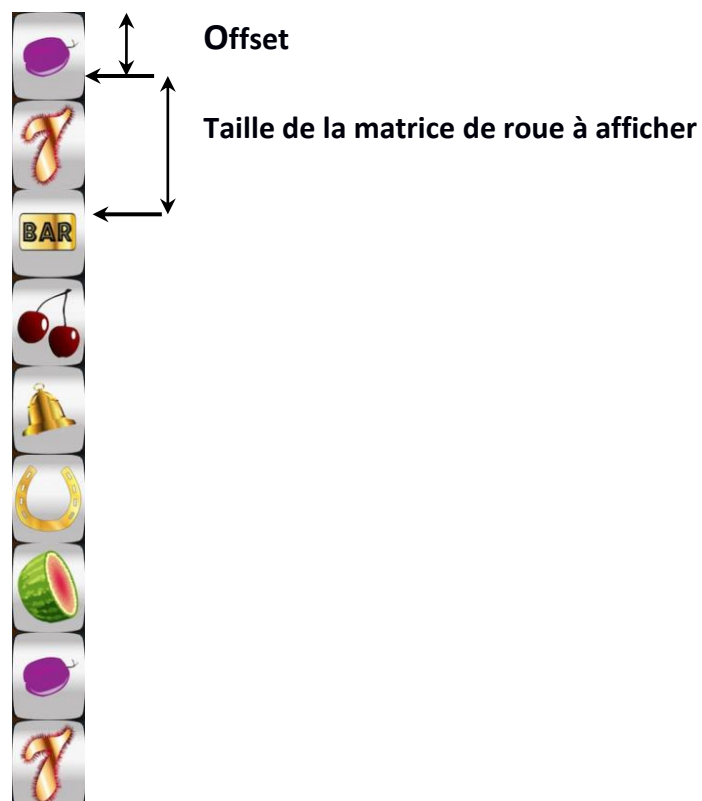
Il s'agit d'écrire un programme qui simule une partie du jeu du bandit manchot, un automate de jeu de hasard, que l'on pourrait trouver dans des casinos et autres salles de jeu. Toute partie ne peut débuter qu'après l'introduction d'une pièce (touche 's'). Le but du jeu consiste alors à presser trois fois sur la barre d'espace. A chaque pression de la touche, l'affichage de la fenêtre correspondante à la pression est figé sur le motif courant d'une roue qui tourne en affichant successivement 7 symboles différents.

Si deux fenêtres affichent le même symbole, le joueur gagne le double de la mise (deux pièces). Si trois fenêtres affichent le même symbole, le joueur remporte la moitié du jackpot (arrondi à l'inférieur). Le jackpot, étant la moitié de l'argent restant dans la caisse. La machine contient 30 pièces au départ et le joueur en a 10 dans son porte-monnaie.



Dans notre cas, l'interface utilisateur comporte :

- La touche 's' pour simuler l'introduction de la pièce (toujours une pièce) et le démarrage (ou redémarrage) de la partie
- La touche espace pour stopper les roues une à une de gauche à droite. **Note** : une roue doit toujours s'arrêter lorsque l'image d'un objet est centrée (comme la deuxième roue ci-dessus), même si la barre d'espace a été pressée juste avant. C'est-à-dire qu'elle doit continuer à tourner régulièrement jusqu'à sa position d'arrêt.
- L'affichage de l'état courant des 3 roues les unes à côté des autres.
- L'affichage du nombre de pièces présentes dans le porte-monnaie du joueur et dans la caisse de la machine
- Lorsque la partie se termine, l'affichage du nombre de pièces doit être ajusté comme indiqué dans les règles du jeu. Dans le cas de la double mise ou du jackpot, affichez le montant gagné et retranchez-le du coffre de la machine. Si le montant gagné devait dépasser celui de la caisse, seul le solde de la caisse serait versé.
- L'affichage des roues doit être progressif et avancer de 2 en 2 pixels. La hauteur de chaque symbole est de 128 pixels, mais la taille de la matrice à afficher est de 192 pixels. Les 7 motifs sont tous stockés successivement dans le fichier **objects.png** (voir image ci-dessous). Afin de faciliter l'affichage déroulant, qui consiste à toujours afficher un bout du motif précédent et/ou un bout du motif suivant, 2 répétitions du même objet sont présentes dans le fichier (en l'occurrence la prune et le '7'). Cela permet de lire l'extrait de l'image à l'endroit qui vous intéresse directement. Dans ce but, gérez l'offset en mémoire afin que la roue à afficher ne sorte jamais de l'image ci-dessous.



## Contraintes de développement

Comme le but de l'exercice est aussi d'exercer les principes de concurrence et de programmer « proprement », voici les points à respecter dans le cadre de ce développement :

- 3 tâches
- Définissez une tâche par roue (`wheel_func()`). Le nom de la fonction correspondant à ces tâches doit être unique et il est interdit de passer le numéro de la roue (`wheel_index`) en paramètre : vous devez donc trouver un moyen d'associer un index à chacune des roues autrement. Chaque roue tourne de 2 pixels toutes les  $6 - wheel\_index * 2$  [ms] (`wheel_index` : vaut 0, 1 ou 2).
  - Une tâche `display_func()` doit gérer tout ce qui concerne l'affichage (d'ailleurs la librairie SDL qui est utilisée pour cela l'impose). L'affichage de l'ensemble des matrices du jeu doit être effectué toutes les 20 ms.
  - Une tâche `keyboard_func()` doit gérer les touches du clavier
- Les 5 tâches décrites ci-dessus doivent être créées une seule fois dans le `main()` et détruites que lorsque l'utilisateur sort du programme.
  - Toutes les attentes doivent être passives
  - Les seules variables globales (à visibilité globale) admises sont celles qui sont déjà définies dans le fichier `display.c`
  - Écrivez votre code de manière générique, de façon à pouvoir par exemple augmenter le nombre de roues avec une constante (mais les conditions de fin de jeu n'ont pas besoin d'être génériques)

**Timings :** les temps d'attentes demandés peuvent être générés à l'aide de la fonction `wait_until()` (voir `utilities.h`)

### Bonus (facultatif)

Effectuer un affichage clignotant des objets **semblables** sur les roues, lorsque celles-ci sont toutes arrêtées. Les fonctions suivantes peuvent être utilisées à cet effet :

- `SDL_SetRenderDrawColor(renderer, 255, 255, 255, 255);`
- `SDL_Rect white_square ;`  
`SDL_RenderFillRect(renderer, &white_square);`

## Préparation

Téléchargez, importez et décompressez le fichier `one-armed_bandit_etu.zip`. Celui-ci contient :

- ***SDL2\_install\_notes.txt*** : note d'installation de la librairie SDL2 nécessaire au projet. Voir l'annexe pour une brève description de la gestion d'affichage de la librairie SDL.
- ***one\_armed\_bandit.c*** : contient le *main()* qui appelle une fonction de démo concernant l'affichage.
- ***display.c*** : contient la fonction *display\_func()*. Celle-ci initialise la SDL, puis donne un exemple d'affichage et de test de touche.
- ***utilities.c*** contient des fonctions utiles pour l'affichage et la gestion du temps. La description des fonctions se trouve dans ***utilities.h***
- un ***makefile*** et les fichiers d'images

## Rendu du travail

Ecrivez un mini-rapport qui contient :

- le nom du TP
- vos noms, prénoms et numéro de groupe
- le statut du projet rendu (fonctionnel ou non, bugs résiduels, etc.). Ce statut doit être exhaustif concernant le comportement du programme.
- Un MSC indiquant comment vous avez synchronisé vos tâches

Le mini-rapport doit rester court comme son nom l'indique. Ne répétez pas les éléments de cet énoncé, en revanche vous pouvez y faire référence si nécessaire.

Créez une seule archive contenant tous les fichiers du projet ainsi que le mini-rapport et nommez-la **G<numéro\_du\_groupe>\_bandit.zip**. Ce fichier doit être rendu sur Cyberlearn.

## Annexe : description simplifiée de l’affichage avec la librairie SDL

La SDL (ou SDL2) est une librairie contenant des utilitaires graphiques, de gestion de du son et du clavier principalement, dont la documentation complète se trouve [ici](#).

Son installation peut se faire sur Ubuntu avec la commande suivante :

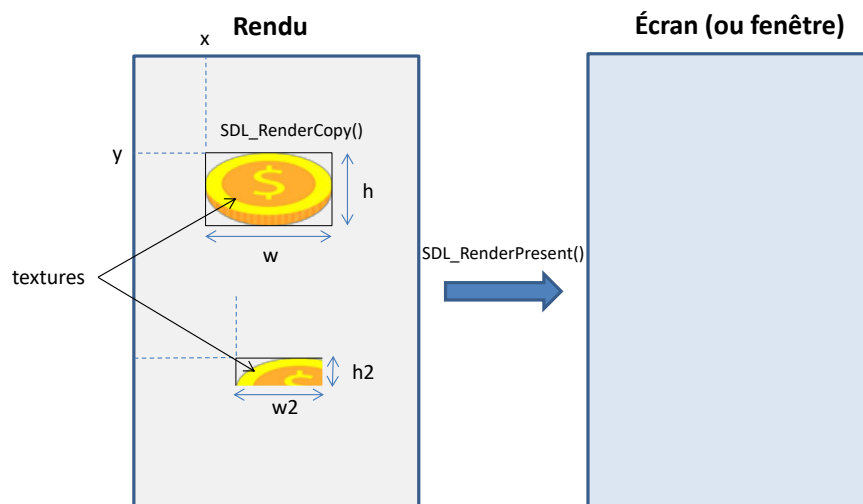
```
sudo apt-get install libghc-sdl2-dev libsdl2-image-dev
```

**Note** : le warning d’exécution suivant « *libpng warning: iCCP: known incorrect sRGB profile* » provient de la librairie SDL. Il n’est pas gênant pour l’exécution, mais il est compliqué à supprimer : veuillez donc l’ignorer.

Le but de cette de cette annexe est de vous expliquer le principe d’affichage de la SDL en quelques lignes. **L’initialisation de la librairie n’est pas décrite, car elle est fournie dans le code du projet de base. Faites attention de bien appeler `SDL_Init()` avant toute autre fonction qui utilise la librairie SDL !**

### Principe général d’utilisation de la SDL

La SDL permet de manipuler des « textures » : les textures sont des matrices stockées en mémoire qui contiennent une image. Ces textures peuvent être copiées sur un « rendu », qui consiste en une zone mémoire qui sert de tampon, et finalement ce rendu peut être affiché à l’écran.



Les textures pourront être chargées directement depuis des fichiers d’images grâce aux utilitaires fournis dans **utilities.c**.

La fonction **`SDL_RenderCopy()`** copie une texture sur le rendu. Sa taille et sa position sur le rendu sont données par les paramètres de la fonction `src_rect` et `dst_rect` qui sont de type `SDL_Rect`. `SDL_Rect` est une structure qui contient 4 paramètres :  $x, y, w, h$

$(x, y)$  sont les coordonnées du point haut gauche de la texture par rapport au rendu

(w,h) sont les dimensions (largeur, hauteur) de la texture à afficher

Un appel typique prend les paramètres suivants :

***SDL\_RenderCopy***(renderer, texture, &src\_rect, &dst\_rect);

Cela permet de copier une partie d'une texture sur une partie du rendu, dont les dimensions (et la position) sont fixées par *src\_rect* (côté texture) et *dst\_rect* (côté rendu). Si &src\_rect=NULL, la texture complète sera copiée (mais pourrait encore être sectionnée par les dimensions spécifiées dans *dst\_rect*).

**Notez que toutes les textures à afficher doivent faire l'objet d'un appel à *SDL\_RenderCopy()*, dans l'ordre dans lequel elles doivent se superposer (fond, roues, puis les pièces de monnaie en commençant par le bas). Ensuite, appelez la fonction suivante qui présente le rendu à l'écran :**

***SDL\_RenderPresent***(renderer)

## Création d'un rectangle de couleur unie

***SDL\_SetRenderDrawColor***(renderer, 255, 255, 255, 255) et

***SDL\_RenderFillRect***(renderer, &square\_rect) créent un rectangle blanc sur le rendu (utile pour implémenter le bonus)

## Gestion du clavier (et/ou des événements)

Pour la gestion du clavier, un exemple de code vous est fourni dans **display.c**. la fonction *SDL\_WaitEvent(&event)* attend un événement quelconque (attente passive). Les événements qui nous intéressent sont traités dans l'exemple : appui d'une touche et test de relâchement ou clic sur la fermeture de la fenêtre (voir **display.c**).

