# Projet de Programmation – Pac-Man

Soukaina Ihirri
soukaina.ihirri@bordeaux-inp.fr
Philippe Swartvagher
philippe.swartvagher@enseirb-matmeca.fr
Amina Teboulbi
amina.teboubli@bordeaux-inp.fr

2023 – 2024

Vous allez implémenter un jeu de Pac-Man, à l'aide de la bibliothèque SDL.

Pour celles et ceux qui ne connaîtrait pas, Pac-Man est un jeu où on déplace un Pac-Man dans un labyrinthe, tout en évitant les fantômes qui s'y déplacent également.

## 1 Un mot rapide sur la SDL

La SDL (*Simple DirectMedia Layer*) est une bibliothèque bas-niveau qui permet de créer des fenêtres, dessiner dedans, gérer les événements du clavier, de la souris, ... C'est une bibliothèque multiplateforme et distribuée sous licence libre. Voyez par exemple la page Wikipédia de la bibliothèque pour des exemples de logiciels qui l'utilisent.

Quelques ressources :

- Le site Internet officiel : `https://www.libsdl.org/`
- Le wiki officiel : `https://wiki.libsdl.org/SDL2/FrontPage`
- Liste des fonctions de la SDL : `https://wiki.libsdl.org/SDL2/CategoryAPI`
- `https://fr.wikibooks.org/wiki/Programmation_avec_la_SDL`
- `https://zestedesavoir.com/tutoriels/1014/utiliser-la-sdl-en-langage-c/`
- `https://jeux.developpez.com/tutoriels/?page=prog-2d#sdl-2`

Petit point d'attention : les versions 1.2 et 2 de la SDL coexistent toujours parallèlement. Nous allons utiliser la version 2 de la SDL.

La SDL toute seule ne permet d'afficher que des images au format BMP. Pour afficher des images dans d'autres formats (notamment PNG pour les images que nous allons utiliser), il faut utiliser en plus une autre bibliothèque : SDL_Image.

### Installation

Tout est déjà installé sur les machines de l'école.

Sur un système basé sur Debian :

```
sudo apt install libsdl2-2.0-0 libsdl2-dev libsdl2-image-2.0-0
   libsdl2-image-dev
```

On installe les paquets `-dev` car on va compiler un programme qui va utiliser ces bibliothèques (et donc on a besoin, par exemple, des fichiers d'en-tête).

Pour ces deux bibliothèques, des fichiers `pkg-config` sont fournis pour connaître les options à utiliser pour compiler un programme avec ces bibliothèques.

## 2 Tâches à réaliser

Partez du code de base fourni sur mon site. Ce code affiche un labyrinthe et un Pac-Man. En appuyant sur une flèche, le Pac-Man se déplace dans la direction de la flèche. Même si ce code ne fait pas beaucoup de choses, il vous montre l'utilisation de la base de la SDL et l'ensemble des fonctions dont vous allez avoir à vous servir.

**Remarque :** ce code est moche ! Il est possible de l'améliorer sur de nombreux points. Vous êtes libres (et même encouragés) à modifier ce que vous voulez dans ce code de base : renommer des variables, introduire des fonctions, changer des algorithmes, découper le code en plusieurs fichiers sources, ...

Vous pouvez modifier ce que vous voulez dans le code, à part les inclusions des fichiers d'en-tête pour la SDL (ne supprimez pas le préfixe `SDL2/`, par exemple).

Vous pouvez organiser votre dépôt comme bon vous semble, mais il faut que en étant à la racine du dépôt, on puisse faire :

```
make
./pacman
```

En même temps que vous réalisez les différentes tâches, pensez à *factoriser* votre code : par exemple, créer des fonctions pour le code dupliqué à plusieurs endroits.

Voici les différentes tâches à réaliser pour le projet :

### Tâche 1 : ajout d'un `Makefile`

Créez un fichier `Makefile` qui compile le code source du programme. Les deux règles suivantes seront définies :

- `all` (règle par défaut) ou `pacman` : construit le programme `pacman` (le programme construit **doit** se nommer `pacman`)
- `clean` : efface tout ce qui a été construit (`pacman`, fichiers `.o`, ...)

La présence de ce `Makefile` et le bon respect du nom des règles sont essentiels pour les tests automatiques qui seront réalisés sur la forge.

## Tâche 2 : restreindre Pac-Man dans le labyrinthe

Pour l'instant, Pac-Man peut bouger n'importe où dans la fenêtre. On souhaite le restreindre dans les chemins du labyrinthe. Il va donc falloir faire la *gestion des collisions* : avant chaque déplacement de Pac-Man, s'assurer qu'il peut aller là où on veut le faire aller.

Pac-Man ne peut changer de direction que lorsqu'il est exactement aligné avec le milieu du chemin où il souhaite se diriger. Il est malheureusement peu probable qu'on arrive à appuyer sur touche de flèche directionnelle pour changer de direction exactement au moment où Pac-Man est à la bonne position pour s'engager dans la nouvelle direction souhaitée. Pour rendre le changement de direction plus facile, on va sauvegarder quelle direction est souhaitée lorsqu'une touche flèche est appuyée. Puis, avant de mettre à jour la position de Pac-Man, on regarde s'il est possible de diriger Pac-Man vers la nouvelle direction souhaitée. Si c'est le cas, on change la direction que suit Pac-Man vers la direction souhaitée, sinon la direction suivie reste la même. Enfin, on change la position de Pac-Man. De cette façon, il est possible de retenir la nouvelle direction souhaitée et l'appliquer quelques trames plus tard, lorsqu'il devient possible d'aller dans cette direction. Ce long paragraphe traduit en pseudo-code donne l'algorithme suivant :

```
1:  while is_running do
2:      Récupère un évenement
3:      if une touche flèche est appuyée then
4:          direction_souhaitée ← direction de la flèche
5:      end if
6:      if il est possible d'aller vers direction_souhaitée then
7:          direction ← direction_souhaitée
8:      end if
9:      if il est possible d'aller vers direction then
10:         Mise à jour de la position de Pac-Man
11:     end if
12:     Effacer l'écran
13:     Tout redessiner
14: end while
```

## Tâche 3 : orienter Pac-Man dans la bonne direction

Pour l'instant, l'image chargée pour représenter Pac-Man l'oriente vers la gauche. Utilisez les trois autres images de Pac-Man à votre disposition pour toujours afficher Pac-Man pointant dans la direction où il se déplace.

## Tâche 4 : ajouter les fantômes

Ajoutez quatre fantômes dans le labyrinthe. On indiquera leurs positions initiales dans la carte, comme c'est fait pour la position initiale de Pac-Man. Pour l'instant, les fantômes se déplaceront aléatoirement : quand ils ont un choix de directions possibles, ils feront le choix aléatoirement. Cependant, la direction d'où ils viennent sera exclue de ce

choix, pour éviter qu'ils reviennent sur leurs pas. Les fantômes commencent à bouger dès que Pac-Man commence à bouger. Lorsqu'un fantôme rencontre Pac-Man, on a perdu, la partie est terminée : on fige ce qui est affiché à l'écran, les touches flèches n'ont plus d'effet.

## Tâche 5 : ajouter les pac-gommes

Ajoutez les pac-gommes sur les chemins du labyrinthe. Pac-Man mange ces pac-gommes : lorsqu'il passe sur une pac-gomme, la pac-gomme disparaît et on incrémente le nombre de pac-gommes mangées.

Pour dessiner les pac-gommes, vous pouvez dessiner des rectangles jaunes avec la fonction `SDL_RenderFillRect()` (ou `SDL_RenderFillRects()`).

## Tâche 6 : manger les fantômes

Une fois un certain nombre de points obtenus (un certain nombre de pac-gommes mangées), les fantômes deviennent vulnérables pendant un court instant. Si Pac-Man rencontre un fantôme pendant ce laps de temps, Pac-Man mange ce fantôme et le fantôme disparaît.

Pour mesurer le temps écoulé, on pourra utiliser la fonction `SDL_GetTicks()`.

## Tâche 7 : diriger les fantômes vers Pac-Man

Mettez en place un algorithme pour diriger les fantômes vers Pac-Man : lorsqu'un fantôme doit choisir une direction, il prendra la direction qui lui permet d'atteindre Pac-Man plus rapidement. On pourra réflechir à plusieurs algorithmes, discuter leurs avantages et inconvénients respectifs [1].

## Tâche 8 : charger le labyrinthe depuis un fichier

Pour l'instant, le labyrinthe est défini directement dans le code. Ajoutez la possibilité de charger un plan de labyrinthe depuis un fichier. Le contenu d'un fichier définissant un labyrinthe ressemblera à ce qui suit :

```
WWWWWWWWWWWWWWWWWWWWWWWW
WS                      W
W  WW  WWWWWWWWWWWWWWWW  W
W      WWWWWWWWWWWWWWWW  W
W  WW  WWWWWWWWWWWWWWWW  W
W  G  G  G  G            W
WWWWWWWWWWWWWWWWWWWWWWWW
```

Le fichier contenant le labyrinthe sera passé comme argument du programme.

1. À propos de l'itinéraire suivi par les fantômes : `https://www.youtube.com/watch?v=ataGotQ7ir8` (je ne vous demande pas forcément d'implémenter cet algorithme en particuliter).

### Tâche 9 : passer d'un côté à l'autre

Un labyrinthe peut ne pas être entièrement délimité par des murs et avoir des « ouvertures » symétriquement opposées, par exemple :

```
WWWWWWWWWW  WWWWWWWWWWWWW
WS                      W
W  WW  WWWWWWWWWWWWWWWWW  W
       WWWWWWWWWWWWWWWWW
W  WW  WWWWWWWWWWWWWWWWW  W
W  G  G  G  G             W
WWWWWWWWWW  WWWWWWWWWWWWW
```

Dans ce cas, si Pac-Man ou un fantôme va dans une « ouverture », il se téléportera dans l'ouverture opposée. Par exemple : si Pac-Man se déplace vers le haut, dans l'ouverture au nord, il continuera à aller vers le haut, mais depuis l'ouverture au sud.

### Ordre des tâches

L'ordre dans lequel vous devez réalisé les tâches est illustré par la figure 2. Il n'est pas nécessaire de réaliser toutes les tâches pour valider le projet.

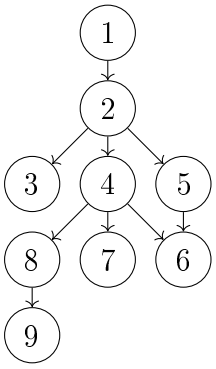


Figure 1 – Ordre de réalisation des tâches

# 3 Évaluation

- Rendu du code
  - Code fonctionnel
  - Code propre (style cohérent, commentaires, ...)
  - Pas de fuite mémoire
- Rapport
  - Il n'est pas nécessaire de répéter le sujet.
  - Expliquez et justifiez les choix que vous avez fait (structures de données, algorithmes, ...)

    - **Pas de code dans le rapport, et encore moins de capture d'écran de code**
    - Il n'est pas forcément nécessaire de parler de toutes les tâches.
- Soutenance lors de la dernière séance de projet
    - Avec des diapositives
    - Reprenez les éléments de votre rapport
    - Une démo de votre version de Pac-Man sera fortement appréciée !

Plus de détails seront fournis plus tard.