TP 1 : Utilisation de Makefile et SDL

Le TP est décomposé en trois parties, la première est liée au `Makefile` (et un peu de `CMake`), la deuxième concerne la bibliothèque *SDL (Simple DirectMedia Layer)*[1] et la troisième est porte sur l'outil de travail collaboratif **github**[2].

Un `Makefile` est un fichier texte qui regroupe une série de commandes indiquant les règles à compiler à partir des fichiers d'un projet. Un `Makefile` peut être écrit à la main, ou généré automatiquement par un utilitaire comme `gmake` par exemple. Il est constitué d'une ou de plusieurs règles de la forme :

```
cible: dépendances
 commandes
```

Un `Makefile` est exécuté sur le terminal par la commande `make` ou `make cible`.

Exercice 1 — Un premier `Makefile`

Créez trois fichiers de sources :

- `addition.h` : un fichier header contenant la déclaration de la fonction `int somme(int a, int b)`.
- `addition.c` : un fichier source comportant la définition de la fonction `somme` déclarée précédemment ; cette fonction fait la somme de deux entiers.
- `main.c` contenant le programme principal `int main(void)` qui demande à l'utilisateur deux entiers et fait la somme de ces deux entiers en utilisant la fonction `somme` définie précédemment.

Le fichier `Makefile` de base pour compiler ce programme est :

```
main: addition.o main.o
 gcc -o main addition.o main.o
addition.o: addition.c
 gcc -o addition.o -c addition.c -W -Wall -ansi -std=c99
main.o: main.c addition.h
 gcc -o main.o -c main.c -W -Wall -ansi -std=c99
```

Maintenant, sur le terminal si vous tapez `make`, que passe-il ? Lancez l'exécutable du programme.

Nous allons ajouter quelques règles de "nettoyage" dans le `Makefile` précèdent

- `clean` : elle permet de supprimer tout les fichiers temporaires `*.o` et `* ~`.
- `mrproper` : elle supprime tout les fichiers régénérés par le `make` du projet.

1. `https://www.libsdl.org`
2. `https://github.com`

Et aussi la règle `all` qui sera exécutée par défaut et qui regroupe l'ensemble des exécutables à produire. Votre `Makefile` devient :

```
all: main
main: addition.o main.o
 gcc -o main addition.o main.o
addition.o: addition.c
 gcc -o addition.o -c addition.c -W -Wall -ansi -std=c99
main.o: main.c addition.h
 gcc -o main.o -c main.c -W -Wall -ansi -std=c99
clean:
 rm -rf *.o *~
mrproper: clean
 rm -rf main
```

Essayez de compiler le projet avec `make`, puis `make clean` et `make mrproper`, que passe-il ?

Exercice 2 _____ `Makefile` enrichi avec les variables

Nous pouvons définir des variables dans un `Makefile`. Elles se déclarent sous la forme `NOM = VALEUR` et sont appelées sous la forme `$(NOM)`.

Nous allons définir les quatre variables suivantes dans le `Makefile` :

— `CC` contenant le compilateur utilisée.
— `CFLAGS` regroupant les options de compilation.
— `LDFLAGS` regroupant les options de l'édition de liens.
— `EXEC` contenant les exécutables à générer.

Avec ces variables, votre `Makefile` devient :

```
CC = gcc
CFLAGS = -W -Wall -ansi -std=c99
LDFLAGS =
EXEC = main

all: $(EXEC)
main: addition.o main.o
 $(CC) -o main addition.o main.o $(LDFLAGS)
addition.o: addition.c
 $(CC) -o addition.o -c addition.c $(CFLAGS)
main.o: main.c addition.h
 $(CC) -o main.o -c main.c $(CFLAGS)
clean:
 rm -rf *.o *~
mrproper: clean
 rm -rf $(EXEC)
```

Comme tous les fichiers objets correspondent aux fichiers sources en remplaçant l'extension `.c` par l'extension `.o`, nous pouvons également définir les trois variables suivantes :

— `SRC` contenant la liste des fichiers sources du projet.
— `DEPS` contenant la liste des fichiers d'en-têtes.
— `OBJ` contenant la liste des fichiers objets.

Avec ces variables, votre `Makefile` devient :

```
CC = gcc
CFLAGS = -W -Wall -ansi -std=c99
LDFLAGS =
EXEC = main
SRC = addition.c main.c
DEPS = addition.h
OBJ = $(SRC:.c=.o)

all: $(EXEC)
main: $(OBJ)
 $(CC) -o main $(OBJ) $(LDFLAGS)
addition.o: addition.c
 $(CC) -o addition.o -c addition.c $(CFLAGS)
main.o: main.c $(DEPS)
 $(CC) -o main.o -c main.c $(CFLAGS)
clean:
 rm -rf *.o *~
mrproper: clean
 rm -rf $(EXEC)
```

Il existe aussi des *variables internes* au `Makefile`, utilisables dans les commandes. Par exemple :

— `$@` : nom de la cible ;
— `$<` : nom de la première dépendance ;
— `$^` : liste des dépendances ;
— `$?` : liste des dépendances plus récentes que la cible ;
— `$*` : nom d'un fichier sans son suffixe.

En utilisant ces variables internes, le `Makefile` devient

```
CC = gcc
CFLAGS = -W -Wall -ansi -std=c99
LDFLAGS =
EXEC = main
SRC = addition.c main.c
OBJ = $(SRC:.c=.o)

all: $(EXEC)
main: $(OBJ)
 $(CC) -o $@ $^ $(LDFLAGS)
addition.o: addition.c
 $(CC) -o $@ -c $< $(CFLAGS)
main.o: main.c addition.h
 $(CC) -o $@ -c $< $(CFLAGS)
clean:
 rm -rf *.o *~
mrproper: clean
 rm -rf $(EXEC)
```

Exercice 3 — `Makefile` avec les règles d'inférences

Dans un `Makefile`, nous pouvons créer des règles génériques (par exemple construire un .o à partir d'un .c). La règle se présente sous la forme :

```
%.o: %.c
 commandes
```

Avec cela, le `Makefile` est simplifié avec :

```
CC = gcc
CFLAGS = -W -Wall -ansi -std=c99
LDFLAGS =
EXEC = main
SRC = addition.c main.c
OBJ = $(SRC:.c=.o)

all: $(EXEC)
main: $(OBJ)
 $(CC) -o $@ $^ $(LDFLAGS)
%.o: %.c
 $(CC) -o $@ -c $< $(CFLAGS)
clean:
 rm -rf *.o *~
mrproper: clean
 rm -rf $(EXEC)
```

Exercice 4 — Compilation avec CMake

Vous pouvez également utiliser `CMake` [3] pour compiler votre programme. Il s'agit d'un système de construction logicielle multiplateforme. Il permet de déterminer les dépendances, de vérifier les pré-requis, etc. afin de générer les fichiers de compilation spécifiques à la plateforme utilisée. `CMake` permet de générer automatiquement le fichier `Makefile`.

Voici un exemple de fichier `CMakeLists.txt` [4] pour votre programme précédent :

```
cmake_minimum_required( VERSION 3.7.0 )

# Set the project name and version
project(main VERSION 1.0)

# Specify the C++ standard
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

# Define a variable with syntax: set(<variable> <value>)
# To use the defined variable: ${variable}
set(SRC_DIR "${CMAKE_CURRENT_SOURCE_DIR}")

# Generate a list of files with syntax: file(GLOB <variable> <files>)
# Specify the sources of the project with ${SRC_DIR}
file(GLOB SOURCES
    "${SRC_DIR}/*.h"
    "${SRC_DIR}/*.c"
)

# Add the executable with the sources in ${SOURCES}
add_executable(${PROJECT_NAME} ${SOURCES})
```

Créez ce fichier `CMakeLists.txt` dans le même dossier que votre TP. Pour compiler avec CMake, exécutez les instructions suivantes sur le terminal au niveau du répertoire contenant `CMakeLists.txt`, puis lancez l'exécutable du programme.

```
mkdir build
cd build
cmake ..
make
```

3. https://cmake.org/
4. https://github.com/ssloy/sdl2-demo

Simple DirectMedia Layer (SDL) est une bibliothèque écrite en langage C qui est utilisée pour créer des applications multimédias ; par exemple les jeux vidéo, les démonstrations graphiques, ... SDL contient les API permettant de gérer l'affichage, les événements, la gestion des périphériques comme le clavier ou la souris, ... Dans ce TP, nous allons utiliser SDL 2.0 [5].

Tout d'abord, vous devez télécharger les supports du TP dans un zip sur le site du cours sur Arche. Ce zip contient

— trois images : *fond.bmp*, *obj.bmp*, *sprites.bmp*
— la bibliothèque tierce `SDL_Image` pour les affichages d'images
— la bibliothèque tierce `SDL_ttf` et le fichier *arial.ttf* pour les affichages du texte

Pour la suite, vous devez placer ces éléments dans le même dossier que votre TP.

Exercice 5 — Première application avec SDL

Tout d'abord, pour utiliser la bibliothèque SDL, il faut inclure le header `SDL.h` dans le programme avec : `#include <SDL2/SDL.h>`. Puis, la création d'une fenêtre SDL est faite avec la fonction suivante :

```
SDL_Window* SDL_CreateWindow(const char* title,
 int x,
 int y,
 int w,
 int h,
 Uint32 flags);
```

Vous trouverez les fonctions et leurs paramètres dans la documentation de la bibliothèque SDL sur `https://wiki.libsdl.org/CategoryAPI`.

Nous allons maintenant créer un programme `main.c` permettant de créer une fenêtre avec la bibliothèque SDL [6].

Puis, créez un `Makefile` pour compiler le programme. Pour cela, vous devez ajouter les variables à la compilation :

— `LIBS = -L./SDL2_ttf/.libs -L./SDL2_image/.libs` : la localisation des bibliothèques tierces de SDL comme `SDL2_image` et `SDL2_ttf` (voir la suite).
— `LDFLAGS = ‘sdl2-config --cflags --libs‘ -lSDL2_ttf -lSDL2_image` : les liens vers les bibliothèques `SDL2`, `SDL2_image` et `SDL2_ttf`
— `INCLUDES = -I./SDL2_ttf -I./SDL2_image` : pour les includes.

Attention si vous faites une **copie-colle de Makefile depuis le PDF**, le caractère **d'apostrophe** dans **LDFLAGS** vient de la touche **AltGr + 7**!!!!

5. `https://www.libsdl.org`
6. Source : `https://fr.wikibooks.org/wiki/Programmation_avec_la_SDL/Les_fenêtres`

```
#include <SDL2/SDL.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

int main(int argc, char *argv[])
{
 SDL_Window* fenetre; // Déclaration de la fenêtre
 SDL_Event evenements; // Événements liés à la fenêtre
 bool terminer = false;
 if(SDL_Init(SDL_INIT_VIDEO) < 0) // Initialisation de la SDL
 {
  printf("Erreur d'initialisation de la SDL: %s",SDL_GetError());
  SDL_Quit();
  return EXIT_FAILURE;
 }
 // Créer la fenêtre
 fenetre = SDL_CreateWindow("Fenetre SDL", SDL_WINDOWPOS_CENTERED,
 SDL_WINDOWPOS_CENTERED, 600, 600, SDL_WINDOW_RESIZABLE);
 if(fenetre == NULL) // En cas d'erreur
 {
  printf("Erreur de la creation d'une fenetre: %s",SDL_GetError());
  SDL_Quit();
  return EXIT_FAILURE;
 }
 // Boucle principale
 while(!terminer){
   SDL_PollEvent( &evenements );
   switch(evenements.type)
   {
    case SDL_QUIT:
    terminer = true; break;
    case SDL_KEYDOWN:
    switch(evenements.key.keysym.sym)
    {
      case SDLK_ESCAPE:
      case SDLK_q:
      terminer = true; break;
    }
   }
 }
 // Quitter SDL
 SDL_DestroyWindow(fenetre);
 SDL_Quit();
 return 0;
}
```

Votre `Makefile` va être :

```
CC = gcc
CFLAGS = -W -Wall -ansi -std=c99 -g
LIBS =
LDFLAGS = `sdl2-config --cflags --libs`
INCLUDES =
EXEC = main
SRC = main.c
OBJ = $(SRC:.c=.o)
all: $(EXEC)
main: $(OBJ)
 $(CC) $(CFLAGS) $(INCLUDES) -o $@ $^ $(LIBS) $(LDFLAGS)
%.o: %.c
 $(CC) $(CFLAGS) -o $@ -c $<
clean:
 rm -rf *.o *~
mrproper: clean
 rm -rf $(EXEC)
```

Compilez avec `make`, puis lancez l'exécutable du programme.
Attention si vous faites une **copie-colle de Makefile depuis le PDF**, le caractère **d'apostrophe** dans **LDFLAGS** vient de la touche **AltGr + 7**!!!!

Votre `CMakeLists.txt` avec `SDL2` (si vous souhaitez compiler votre programme en utilisant `CMake`) va être :

```
cmake_minimum_required( VERSION 3.7.0 )
project( main )
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(SRC_DIR "${CMAKE_CURRENT_SOURCE_DIR}")
file(GLOB SOURCES "${SRC_DIR}/main.c")
add_executable(${PROJECT_NAME} ${SOURCES})

# Find installed packages with syntax: find_package(<PackageName>)
find_package( SDL2 )
# If the package is found, serveral variables associated to the
# library are created:  ${SDL2_FOUND} if SDL2 is found in the system
# ${SDL2_INCLUDE_DIRS} contains the path of SDL2 include folder
# ${SDL2_LIBRARIES} contains the path of SDL2 lib
if( ${SDL2_FOUND})
    message(STATUS "Found SDL2")
    # Include directories to build the project
    include_directories(${SDL2_INCLUDE_DIRS})
    # Specify libraries for linking to the project
    target_link_libraries(${PROJECT_NAME} ${SDL2_LIBRARIES})
else()
    message(STATUS "Could not locate SDL2")
endif()
```

Vous devez changer peut-être `#include <SDL2/SDL.h>` en `#include <SDL.h>`.

Exercice 6 — Afficher une image

Le chargement d'image en SDL 2.0 est faite avec les fonctions suivantes :

```
// Créer un contexte de rendu (renderer) pour l'image
SDL_Renderer* SDL_CreateRenderer(SDL_Window* window, int index, Uint32 flags) ;
// Charger une image
SDL_Surface* SDL_LoadBMP(const char* file) ;
 // Convertir la surface de l'image au format texture avant de l'appliquer
SDL_Texture* SDL_CreateTextureFromSurface(SDL_Renderer* renderer,
 SDL_Surface* surface) ;
// Copier (une partie de) la texture dans le renderer
int SDL_RenderCopy(SDL_Renderer* renderer,
 SDL_Texture* texture,
 const SDL_Rect* srcrect,
 const SDL_Rect* dstrect) ;
// Récupérer les attributs d'une texture
int SDL_QueryTexture(SDL_Texture* texture,
 Uint32* format,
 int* access,
 int* w,
 int* h) ;
// Libérer une surface
void SDL_FreeSurface(SDL_Surface* surface) ;
// Libérer une texture
void SDL_DestroyTexture(SDL_Texture* texture) ;
```

Vous trouverez les fonctions et leurs paramètres dans la documentation de la bibliothèque SDL sur `https://wiki.libsdl.org/CategoryAPI`. Vous pouvez utiliser la bibliothèque tierce nommée `SDL_Image`[7] pour plus de format d'images (png, jpg, gif, ...)[8].

Créez deux fichiers de sources :

- `fonctions_SDL.h` : un fichier header contenant la déclaration de la fonction :
  - `SDL_Texture* charger_image (const char* nomfichier, SDL_Renderer* renderer)` : charger une image et retourner la surface de texture associée.
- `fonctions_SDL.c` : un fichier source contenant les définitions des fonctions déclarées dans `fonctions_SDL.h`.

Implémentez cette fonction dans `fonctions_SDL.h` et `fonctions_SDL.c`. Puis, dans `main.c`, appelez la fonction de chargement d'image comme indiqué ci-dessous :

7. `https://www.libsdl.org/projects/SDL_image`

8. Pour utiliser `SDL_Image`, téléchargez le zip sur Arche, décompressez dans le même répertoire de votre code puis allez dans le répertoire décompressé et lance la commande `./configure` pour compiler la bibliothèque. N'oubliez pas d'ajouter `SDL2_Image` dans le makefile (voir l'exercice 4).

```
// Mettre en place un contexte de rendu de l'écran
SDL_Renderer* ecran;
ecran = SDL_CreateRenderer(fenetre, -1, SDL_RENDERER_ACCELERATED);

// Charger l'image
SDL_Texture* fond = charger_image( "fond.bmp", ecran );

// Boucle principale
while(!terminer)
{
 SDL_RenderClear(ecran);
 SDL_RenderCopy(ecran, fond, NULL, NULL);
 //SDL_PollEvent ...
 SDL_RenderPresent(ecran);
}
// Libérer de la mémoire
SDL_DestroyTexture(fond);
SDL_DestroyRenderer(ecran);
//Quitter SDL ...
```

Ajoutez `fonctions_SDL.c` dans **SRC du Makefile** : `SRC = main.c fonctions_SDL.c`
Nous avons besoin par la suite d'afficher des images avec la transparence. Chaque surface SDL possède un élément nommé *color key* qui indique la couleur à ne pas afficher lors de l'application d'une surface. Pour la transparence, nous utilisons les deux fonctions :

```
// Récupérer la valeur (RGB) du pixel au format donné.
Uint32 SDL_MapRGB(const SDL_PixelFormat* format, Uint8 r, Uint8 g, Uint8 b);
// Définir la couleur (pixel transparent) dans une surface.
int SDL_SetColorKey(SDL_Surface* surface, int flag, Uint32 key) ;
```

Pour le paramètre `SDL_PixelFormat* format` dans `SDL_MapRGB`, nous utilisons le même format que `Surface = SDL_LoadBMP(...)`, C-à-d `Surface->format`.
Ajouter la fonction `charger_image_transparente` dans `fonctions_SDL.h` et `fonctions_SDL.c` avec la couleur (RGB) dont on souhaite qu'elle devienne le color key (la couleur transparente). La déclaration de la fonction est :

```
SDL_Texture charger_image_transparente(const char* nomfichier,
 SDL_Renderer* renderer,
 Uint8 r, Uint8 g, Uint8 b) ;
```

Implémentez cette fonction `charger_image_transparente`. Testez la fonction implémentée en ajoutant le code dans le programme principal.

```
// Charger l'image avec la transparence
Uint8 r = 0, g = 255, b = 255;
SDL_Texture* obj = charger_image_transparente("obj.bmp", ecran,r,g,b);

SDL_Rect SrcR;
SrcR.x = 0;
SrcR.y = 0;
SrcR.w = objetW; // Largeur de l'objet en pixels (à récupérer)
SrcR.h = objetH; // Hauteur de l'objet en pixels (à récupérer)

DestR.x = 350;
DestR.y = 350;
DestR.w = objetW/3;
DestR.h = objetH/3;

// Boucle principale
while(!terminer)
{
 SDL_RenderClear(ecran);
 SDL_RenderCopy(ecran, fond, NULL, NULL);
 SDL_RenderCopy(ecran, objet_transp, &SrcR, &DestR);
 ...
 SDL_RenderPresent(ecran);
}
```

Exercice 7 — Feuille de sprites

Un sprite est une image avec une couleur transparente et lors de l'affichage cette couleur sera masquée. Dans cette partie, nous allons utiliser une feuille de sprites, c'est une image contenant plusieurs sprites (voir image *sprites.bmp*). Cette feuille est composée de six sprites et s'organise en 2 lignes et 3 colonnes. Nous allons la couper sprite par sprite afin de pouvoir les afficher individuellement. Pour cela, nous utilisons un tableau de six `SDL_Rect` qui stocke les positions et les dimensions de chacun des six sprites. Sachant que les sprites dans la feuille de sprites ont la même dimension, pour les séparer et calculer leur positions et dimensions, il suffit de diviser la taille de la feuille de sprites par rapport le nombre de lignes ou de colonnes. Ce tableau de `SDL_Rect` sera passé ensuite à la fonction `SDL_RenderCopy` en tant que paramètre `srcrect` pour l'affichage.

Ajouter le code dans `main.c` permettant de remplir le tableau de `SDL_Rect` pour les positions et dimensions des sprites. Ajouter également le code suivant pour le paramètre `dstrect`, puis les faire afficher avec la fonction `SDL_RenderCopy`.

```
SDL_Rect DestR_sprite[6];
for(int i=0; i<6; i++)
{
 DestR_sprite[i].x = i > 2 ? 60*(i+1)+100 : 60*(i+1);
 DestR_sprite[i].y = i > 2 ? 60 : 120;
 DestR_sprite[i].w = tailleW; // Largeur du sprite
 DestR_sprite[i].h = tailleH; // Hauteur du sprite
}
```

Exercice 8 — Afficher du texte

Parce que la bibliothèque SDL seule ne gère pas l'affichage de texte, nous allons avoir besoin d'une autre bibliothèque tierce nommée `SDL_ttf`[9] pour l'affichage du texte[10]. Tout d'abord, il nous faut inclure la bibliothèque avec : `#include <SDL2/SDL_ttf.h>`.

Pour l'affichage du texte avec `SDL_ttf`, nous allons utiliser les fonctions suivantes :

```
// Initialise SDL_ttf, elle retourne -1 s'il y a une erreur
int TTF_Init( ) ;
// Charger la police avec la taille donnée
TTF_Font *TTF_OpenFont(const char *file, int size) ;
// Écrire le texte sur une surface SDL
SDL_Surface *TTF_RenderText_Solid(TTF_Font *font, const char *text, SDL_Color fg) ;
// Fermer la police
void TTF_CloseFont(TTF_Font *font) ;
// Fermer SDL_ttf
void TTF_Quit() ;
```

Vous trouverez ces fonctions dans la documentation de la bibliothèque `SDL_ttf` sur `https://www.libsdl.org/projects/SDL_ttf/docs/SDL_ttf.html`.
Ajouter la fonction `charger_texte` dans `fonctions_SDL.h` et `fonctions_SDL.c` avec la déclaration suivante :

```
SDL_Texture* charger_texte(const char* message, SDL_Renderer* renderer,
 TTF_Font *font, SDL_Color color) ;
```

Implémentez cette fonction `charger_texte`. Testez la fonction implémentée en ajoutant le code suivant dans le programme principal.

9. `https://www.libsdl.org/projects/SDL_ttf`
10. Pour utiliser `SDL_ttf`, téléchargez le zip sur Arche, décompressez dans le même répertoire de votre code puis allez dans le répertoire décompressé et lancez la commande `./configure` pour compiler la bibliothèque. N'oubliez pas d'ajouter `SDL2_ttf` dans le makefile (voir l'exercice 4).

```
TTF_Init();
TTF_Font *font = TTF_OpenFont("./arial.ttf",28);
SDL_Color color = {0,0,0,0};
char msg[] = "TP sur Makefile et SDL";
SDL_Texture* texte = charger_texte(msg,ecran,font,color);
SDL_Rect text_pos; // Position du texte
text_pos.x = 10;
text_pos.y = 100;
text_pos.w = texteW; // Largeur du texte en pixels (à récupérer)
text_pos.h = texteH; // Hauteur du texte en pixels (à récupérer)
// Boucle principale
while(!terminer)
{
 ...
 //Appliquer la surface du texte sur l'écran
 SDL_RenderCopy(ecran,texte,NULL,&text_pos);
}
...
// Fermer la police et quitter SDL_ttf
TTF_CloseFont( font );
TTF_Quit();
```

Votre `makefile` va être :

```
CC = gcc
CFLAGS = -W -Wall -ansi -std=c99 -g
LIBS = -L./SDL2_ttf/.libs
LDFLAGS = `sdl2-config --cflags --libs` -lSDL2_ttf
INCLUDES = -I./SDL2_ttf
EXEC = main
SRC = main.c fonctions_SDL.c
OBJ = $(SRC:.c=.o)

all: $(EXEC)
main: $(OBJ)
 $(CC) $(CFLAGS) $(INCLUDES) -o $@ $^ $(LIBS) $(LDFLAGS)
%.o: %.c
 $(CC) $(CFLAGS) -o $@ -c $<
clean:
 rm -rf *.o *~
mrproper: clean
 rm -rf $(EXEC)
```

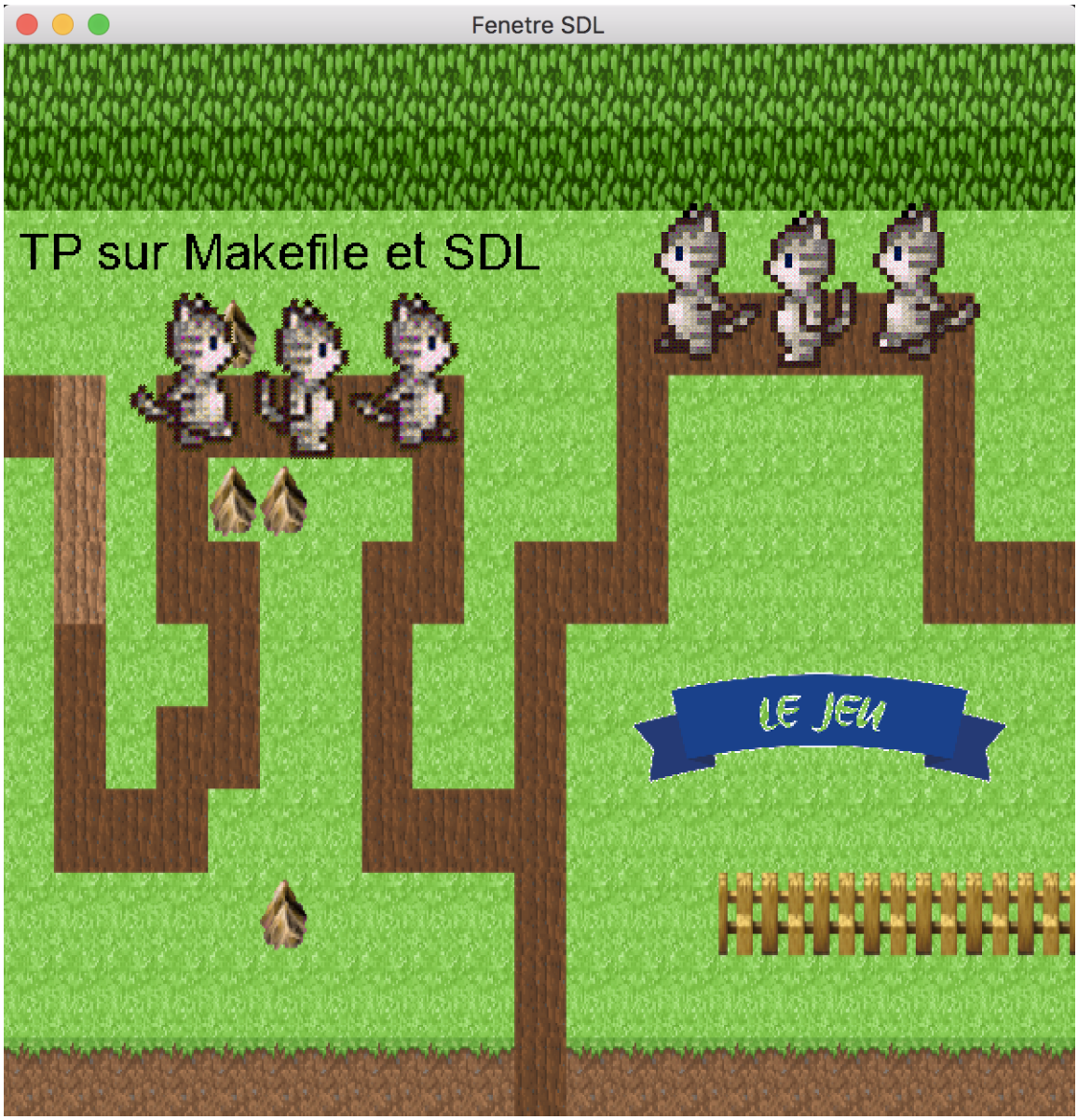
Attention si vous faites une **copie-colle de Makefile depuis le PDF**, le caractère **d'apostrophe** dans **LDFLAGS** vient de la touche **AltGr + 7**!!!!

Votre `CMakeLists.txt` avec `SDL2` et `SDL2_TTF` (si vous souhaitez compiler votre programme en utilisant `CMake`) va être :

```
cmake_minimum_required( VERSION 3.7.0 )
project( main )
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(SRC_DIR "${CMAKE_CURRENT_SOURCE_DIR}")

#Append elements to list with: list(APPEND CMAKE_MODULE_PATH <path>)
#CMAKE_MODULE_PATH contains the list of search paths for CMake modules
#It is used by include() or find_package()
list(APPEND CMAKE_MODULE_PATH ${SRC_DIR}/cmake/sdl2)

#Specify the sources of the project
file(GLOB SOURCES
    "${SRC_DIR}/*.h"
    "${SRC_DIR}/*.c"
)

#Add the executable
add_executable(${PROJECT_NAME} ${SOURCES})

#Find the packages SDL2 and SDL2_ttf with files *.cmake in
#${SRC_DIR}/cmake/sdl2 of CMAKE_MODULE_PATH
find_package( SDL2 )
find_package( SDL2_TTF )
if( ${SDL2_FOUND} AND ${SDL2_TTF_FOUND})
    message(STATUS "Found SDL2 and SDL2_TTF")

    # Add a directory to list of include directory for a target
    # The keyword PRIVATE indicating that the directory is directly
    # added to the target's include directories
    target_include_directories(${PROJECT_NAME} PRIVATE include)

    # Link the libraries to the project
    target_link_libraries(${PROJECT_NAME} SDL2::Main SDL2::TTF)
else()
    message(STATUS "Could not locate SDL2 or SDL2_TTF")
endif()
```

Pour compiler, vous devez télécharger `cmake.zip` sur Arche, décompressez et copiez-le dans votre dossier de TP. Mettez le fichier `arial.ttf` et les images le dossier `build`.

Après avoir compilé et exécuter, vous devez obtenir le résultat comme suivant :



Exercice 9 — Gestion de version : Github

Pour le projet de l'UE programmation avancée, vous allez le réaliser en binôme et travailler avec un dépôt git, qui est un outil de travail collaboratif. En particulier, nous utilisons le serveur **github** : `https://github.com`. L'objectif de cet exercice est de mettre en place un dépôt à distance sur `github` pour une première version du projet. Ce dépôt sera utilisé par la suite par votre binôme pour les versions suivantes du projet.

Si vous ne possédez pas encore un compte sur `https://github.com`, vous devez vous inscrire pour avoir un compte (et éventuellement déposer la clé publique). Après avoir créé un compte, vous pouvez créer un dépôt (repository) pour le projet et le partager avec votre binôme.

Voici quelques instructions utiles pour la manipulation du dépôt :

```
// Initialiser un dépôt
git init
// Cloner un dépôt
git clone <url>
// Voir l'état du dépôt
git status
// Ajouter et modifier le dépôt
git add <nom_fichier>
git commit -m "message"
git push
// Récupérer les données depuis le serveur
git pull
// Ajouter des étiquettes
git tag -a v0.0 -m "message"
// Revenir sur une version précédente
git checkout <id-commit>
git checkout tags/<id-tag>
```

Pour plus de détails sur l'utilisation de `github` et ses instructions, vous pouvez consulter : `http://rogerdudler.github.io/git-guide`.

Maintenant, nous allons ajouter le fichier `makefile` et les fichiers des sources du TP sur le dépôt git du projet en ajoutant l'étiquette "v.0.1". Vérifiez que tous sont bien sur le serveur et que vous pouvez les récupérer sur votre ordinateur.