TP 2 : Gestion d'entrées sorties avec les fichiers et SDL

Pour réaliser ce TP, vous devez télécharger les supports du TP dans un zip sur le site du cours sur Arche. Ce zip contient

— trois fichiers de textes : `tabCaracteres.txt`, `tabChiffres.txt` et `terrain.txt`

— les fichiers d'images : *fond.bmp*, *pavage.bmp* et *sprites.bmp*

Vous devez placer ces fichiers dans le dossier de votre projet (code source).

Exercice 1 — Lecture des fichiers

Dans la première partie du TP, nous allons faire la lecture et l'écriture de fichiers. Pour cela, créer les trois fichiers suivants :

— `fonctions_fichiers.h` : un fichier header contenant les déclarations des fonctions suivantes :

  — `char** allouer_tab_2D(int n, int m)` : allouer un tableau de caractères de taille $n \times m$, où n et m sont les nombres de lignes et de colonnes, et initialiser ce tableau avec le caractère espace (' ').

  — `void desallouer_tab_2D(char** tab, int n)` : libérer un tableau à deux dimensions, de n lignes.

  — `void afficher_tab_2D(char** tab, int n, int m)` : afficher le contenu d'un tableau de caractères de taille $n \times m$.

  — `void taille_fichier(const char* nomFichier, int* nbLig, int* nbCol)` : compter le nombre max de lignes (`nbLig`) et de colonnes (`nbCol`) dans le fichier dont le nom est `nomFichier`.

  — `char** lire_fichier(const char* nomFichier)` : lire un fichier dont le nom est `nomFichier`, et retourner le tableau qui contient les caractères du fichier tel qu'une ligne du tableau correspond à une ligne du fichier.

— `fonctions_fichiers.c` : un fichier source concernant la définition des fonctions déclarées précédemment.

— `main.c` contenant le programme principal `int main(void)` qui teste les fonctions définies dans `fonctions_fichiers.h` et `fonctions_fichiers.c`

Créer le `Makefile` pour le projet et compiler le projet avec `make`, puis tester avec les différents fichiers de texte, donnés dans le support du TP2 (`tabRect.txt`, `tabChiffres.txt` et `tabChiffres.txt`). Voici un exemple d'affichage avec le fichier `tabChiffres.txt`.

```
taille du fichier : 3 x 11
Le contenue du tableau (3 x 11) est :
1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2
3 3 3 3 3
```

Exercice 2 — Écriture des fichiers

Ajouter dans `fonctions_fichiers.h` et `fonctions_fichiers.c` les fonctions suivantes :

- `char** modifier_caractere(char** tab, int n, int m, char ancien, char nouveau)` : Retourner un nouveau tableau, dans lequel toutes les occurrences du caractère `ancien` sont remplacées par le caractère `nouveau`.
- `void ecrire_fichier(const char* nomFichier, char** tab, int n, int m)` : Écrire le tableau `tab` de taille $n \times m$ dans un fichier dont le nom est `nomFichier`.

Tester ces fonctions en ajoutant le code dans le programme principal. Voici un exemple d'affichage avec le fichier `tabChiffres.txt` après avoir remplacé '1' par 'a' en utilisant la fonction `modifier_caractere`.

```
Le contenue du tableau après avoir remplacé 1 par a est :
a a a a a a a
2 2 2 2 2 2 2 2 2 2
3 3 3 3 3
```

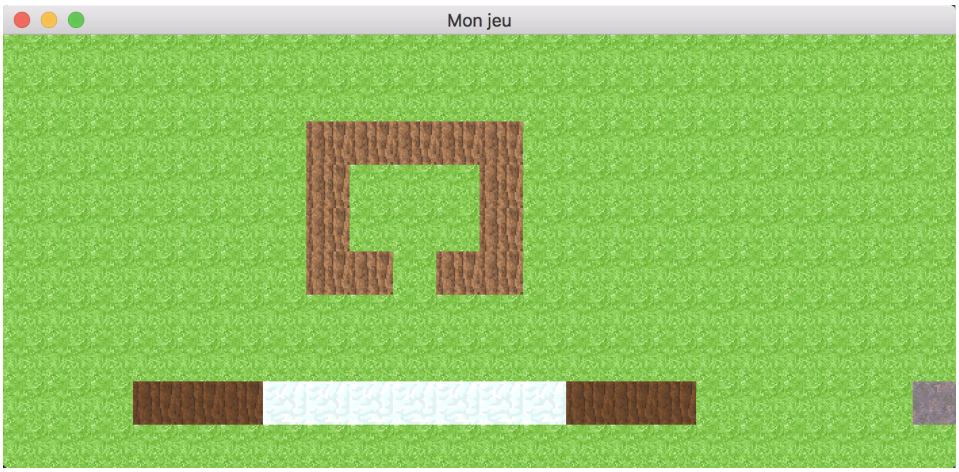
Exercice 3 — Affichage avec SDL

Nous allons dans la suite afficher le graphique associé au contenu d'un tableau lu à partir d'un fichier texte. Cette affichage graphique est réalisé grâce à la bibliothèque SDL.

Modifier le programme principal pour réaliser les tâches suivantes :

- Lire le fichier texte `terrain.txt` et le stocker dans un tableau.
- Créer une fenêtre SDL associée à un rendu (`SDL_Renderer`) grâce à la fonction `SDL_CreateRenderer`. (voir TP1)
- Charger les sprites dans le fichier `pavage.bmp`.
- Faire une boucle sur le tableau des caractères lus à partir du fichier `terrain.txt`, pour chaque caractère afficher le sprite associé tel que : le caractère '0' correspond au sprite d'indice 0, '1' correspond au sprite 1, et ainsi de suite. Notez que pour les caractères autres que les chiffres, on affiche le sprite d'indice 0 (par défaut).

Pour le fichier `terrain.txt`, vous devez obtenir le résultat suivant :



Modifier le fichier `terrain.txt` et relancer le programme, que passe-il ?

Exercice 4 — Événement avec le clavier

Dans cette partie, nous allons placer un personnage dans le jeu (par exemple le 1er sprite de `sprites.bmp`) et déplacer ce personnage sur l'herbe grâce aux touches des flèches du clavier : $\uparrow, \rightarrow, \uparrow$ et $\downarrow$ pour aller à gauche, droite, haut et bas. Pour cela, nous allons utiliser la gestion des événements du clavier [1] de SDL avec la fonction suivante :

```
int SDL_PollEvent(SDL_Event* event) ; // Attendre un évènement
```

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

Modifier le programme principal pour réaliser les tâches suivantes :

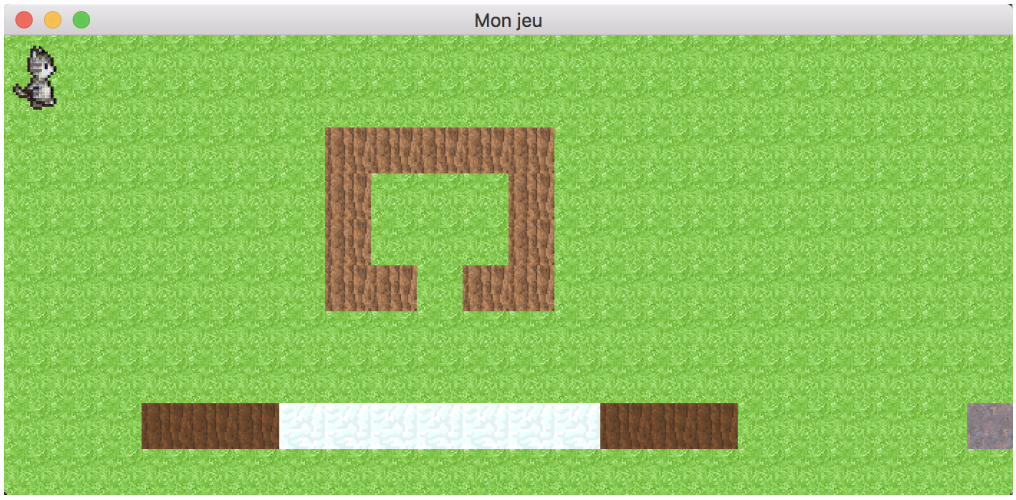
— Placez le personnage en haut à gauche (position 0,0) de la fenêtre contenant le terrain de jeu (voir la figure de l'exercice 3).

— Utilisez la fonction `SDL_PollEvent` pour gérer les déplacements du personnage. Votre boucle sur `SDL_PollEvent` aura la forme suivante :

```
while( 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;
        case SDLK_UP:
          ...
          break;
        case SDLK_DOWN:
          ...
          break;
        case SDLK_LEFT:
          ...
          break;
        case SDLK_RIGHT:
          ...
          break;
      }
  }
}
```

1. `https://wiki.libsdl.org/SDL_Keycode`

Voici le résultat obtenu :



Vous pouvez ajouter les contrôles sur le déplacement du personnage en sorte qu'il ne traverse pas les murs mais seulement l'herbe. Pour cela, il faut faire la détection de collision entre le sprite du personnage et le sprite sur lequel il se déplace.

Exercice 5 — Événements avec la souris

Dans cette partie, nous allons modifier des objets sur le terrain du jeu grâce à la souris. Pour cela, nous allons utiliser la gestion des événements de la souris [2] de SDL avec la fonction `SDL_PollEvent`.

Ajouter le code dans le programme pour réaliser les tâches suivantes :

— Si on clique sur le sprite représentant l'herbe (le sprite numéro 0) avec le bouton gauche, celui-ci devient une mur (le sprite numéro 4)

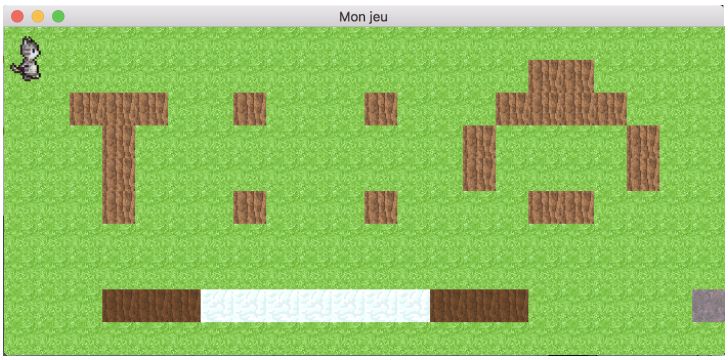
— et à l'inverse, si on clique sur le sprite représentant un mur, avec le bouton droit, celui-ci devient de l'herbe.

Voici à quoi ressemblera votre boucle `SDL_PollEvent` :

2. `https://wiki.libsdl.org/SDL_MouseButtonEvent`

```
while( SDL_PollEvent( &evenements ) ) {
  switch(evenements.type)
  {
    ...
    case SDL_MOUSEBUTTONDOWN:
      switch(evenements.button.button)
      {
        case SDL_BUTTON_LEFT:
          ...
          break;
        case SDL_BUTTON_RIGHT:
          ...
          break;
      }
   }
}
```

Après avoir modifié le terrain du jeu avec la souris, sauvegarder le tableau associé en fichier texte avec la fonction `ecrire_fichier` (voir l'exercice 2) lors de l'appui sur la touche `s`. Par exemple, pour le terrain modifié, le fichier texte associé sera comme suit :




```
00000000000000000000
00000000000000440000
00444004000400444000
00040000000040000400
00040004000400000400
00040004000400440000
00000000000000000000
00000000000000000000
00033311111113330000 2
00000000000000000000
```