TP 11 : Simulation du jeu de la bataille

Notions : Type abstrait, Files, Piles

1 Règle du jeu de la bataille

On distribue les 52 cartes une par une aux 2 joueurs qui les rassemblent en paquet, face cachée, devant eux. On appellera ces deux paquets les jeux des joueurs.

L'ordre des cartes est le suivant : le roi est la plus forte carte, puis vient la dame, le valet, le 10, 9,...3,2,As (On peut aussi jouer avec l'ordre As, Roi, Dame,...3,2.)

Un tour de jeu se déroule de la manière suivante : Chacun tire la carte du dessus de son jeu et la pose sur la table, face découverte, à coté de son jeu. Cette carte constitue la pile de bataille. Celui qui a la carte la plus forte ramasse toutes les cartes des 2 piles de bataille.

Lorsque les deux joueurs posent deux cartes de même valeur sur leur pile de bataille, il y a "bataille". Lorsqu'il y a bataille, les joueurs tirent une nouvelle carte de leur jeu sans la regarder et la posent, face cachée, sur la pile de bataille. Puis ils tirent une deuxième carte qu'ils posent cette fois-ci face découverte sur la pile de bataille. C'est cette dernière carte découverte qui départagera les joueurs. Si ces cartes sont aussi identiques, il y a à nouveau bataille et chaque joueur tire à nouveau une carte qu'il pose face cachée sur les précédentes, puis chaque joueur tire une nouvelle carte qu'il pose face découverte sur les précédentes : la plus forte remporte toutes les cartes posées, et s'il y a à nouveau bataille, on recommence tant que les cartes découvertes sont égales.

Le gagnant est celui qui finit par remporter toutes les cartes. Lorsqu'un des deux joueurs n'a pas assez de cartes pour continuer une bataille en cours (c'est la notion de famine), ce joueur a perdu.

La simulation, sans tenir compte de la famine (un joueur n'a plus de cartes) dans un premier temps, peut s'écrire alors sous la forme l'algorithme 1 :

2 Structure de données

Pour simuler le jeu, les structures de données utilisées sont des files et des piles.

Le jeu de chaque joueur peut être représenté par une file. Au début du tour de jeu, le joueur prend la carte du dessus (defiler) et s'il gagne, il prend les cartes jouées sur la pile de bataille et les ajoute par dessous (enfiler).

Les paquets de bataille sont des piles, car on ajoute toujours sur le sommet du paquet et on regarde et compare la carte du sommet. Lorsqu'un joueur gagne, il prend les cartes des piles en commençant par celle du dessus.

3 Travail à réaliser

1. Décompressez le fichier td11.zip. Il contient les fichiers `Makefile`, `distribution.c` et `distribution.h`. Il contient aussi les fichiers `exemple.c`, `graph_print.c` et `graph_print.h` pour réaliser une version graphique.
   Le fichier `Makefile` suppose que les fichiers suivants existent et comportent les fonctions débuggées :
   — Les fonctions sur les listes sont dans les fichiers `list.c` et `list.h`,

Algorithm 1 Algorithme de la bataille

tant que le jeu 1 n'est pas vide et le jeu 2 n'est pas vide **faire**
  Prendre la carte du dessus du jeu 1 et la retourner
  Mettre cette carte sur le dessus de la pile de bataille 1
  Prendre la carte du dessus du jeu 2 et la retourner
  Mettre cette carte sur le dessus de la pile de bataille 2
  tant que les 2 cartes du dessus des piles de bataille sont égales **faire**
    Prendre la carte du dessus du jeu 1
    Mettre cette carte face cachée sur le dessus de la pile de bataille 1
    Prendre la carte du dessus du jeu 1 et la retourner face visible
    Mettre cette carte sur le dessus de la pile de bataille 1
    Prendre la carte du dessus du jeu 2
    Mettre cette carte face cachée sur le dessus de la pile de bataille 2
    Prendre la carte du dessus du jeu 2 et la retourner face visible
    Mettre cette carte sur le dessus de la pile de bataille 2
  fin tant que
  Joueur gagnant ce tour : celui qui a la plus forte carte en sommet de pile de bataille
  tant que la pile de bataille 1 n'est pas vide **faire**
    Prendre la carte sur le dessus de la pile de bataille 1
    La glisser sous le paquet du gagnant
  fin tant que
  tant que la pile de bataille 2 n'est pas vide **faire**
    Prendre la carte sur le dessus de la pile de bataille 2
    La glisser sous le paquet du gagnant
  fin tant que
fin tant que
Le gagnant est celui qui a toutes les cartes dans son jeu

   — les fonctions sur les piles sont dans les fichiers `lifo.c` et `lifo.h`,
   — les fonctions sur les files sont dans les fichiers `fifo.c` et `fifo.h`,
   — les fonctions sur les cartes sont dans les fichiers `element.c` et `element.h`
   — les fonctions de distribution sont dans le fichier `distribution.c` et `distribution.h`
   — le programme principal est dans le fichier `bataille.c`

2. Ecrire le programme dans le fichier `bataille.c` qui simule le jeu de la bataille tel que proposé dans l'algorithme précédent en utilisant les fonctions réalisées précédemment et la fonction de distribution donnée.
3. Compiler à l'aide de la commande `make bataille` et tester votre programme.
4. Ajouter la gestion de la famine : lorsqu'un joueur n'a plus aucune carte, il perd la partie

Voici les fonctions utiles, disponibles dans le fichier `distribution.c`

— `int distribution(fifo_t* ajeu1, fifo_t* ajeu2, int alea, int nbcarte)`
  Cette fonction effectue les actions suivantes :
  1. Création des cartes,
  2. Mélange de ces cartes
  3. Distribution des cartes dans les files `*ajeu1` et `*ajeu2`.

  Les paramètres ont la signification suivante :

  nbcarte est le nombre de cartes dans chaque couleur ($0 < nbcarte < 14$). Utiliser ce paramètre pour tester et debugger votre code. Commencez par des valeurs faibles (4 par exemple) pour bien suivre le déroulement de la partie.

  alea est la valeur initiale du générateur pseudo-aléatoire. Le générateur pseudo aléatoire (fonction `int rand()`) est une suite $U_n = f(U_{n-1})$ pour laquelle il est difficile de prévoir (pour un

humain) le prochain nombre et dont la distribution est uniforme. Tous les nombres sont équiprobables. `alea` est la valeur initiale U_0 de cette suite U_n. Pour debugger, utiliser une valeur non nulle pour `alea`. Pour obtenir des tirages différents, utiliser ensuite des valeurs différentes à chaque exécution.

Attention : cette fonction utilise la fonction `fifo_t fifo_enqueue(CARTE e, fifo_t f)` que vous devez avoir écrite et tester lors du tp précédent.

4 Version graphique

Des fonctions permettant d'afficher des files et des piles de cartes sous forme graphique sont disponibles. Pour les files, ces fonctions utilisent une représentation par liste circulaire.

4.1 Structure d'une carte en version graphique

La structure de données représentant une carte contient alors son rang, sa couleur, si elle est face cachée ou non et un nouveau champ `im` de type `ICONE` qui contient une image 32 bits d'une carte ainsi que la dimension de cette image. La structure de données est définie dans le fichier `element.h` qui permet la compilation conditionnelle. La définition de la constante `MODEGRAPHIQUE` par une ligne du type `#define MODEGRAPHIQUE 1` définit le type `CARTE` avec un champ `im` de type `ICONE` alors que si cette constante `MODEGRAPHIQUE` n'est pas définie, le type `CARTE` reste identique à celui qui a été utilisé jusqu'à présent.

```
#ifndef _ELEMENT
#define _ELEMENT

typedef enum { TREFLE,CARREAU,COEUR,PIQUE} COULEUR;

#ifdef MODEGRAPHIQUE
// Definition de la structure element_t en mode graphique, avec un champ qui contient l'image de la
   carte et une champ qui contient l'image du dos de la carte.
typedef struct { unsigned int** data; int lig, col; } ICONE;
typedef struct {
  int rang;
  char visible;
  COULEUR couleur;
  ICONE im,dos;
  }  element_t;
#else
/ Definition de la structure element\_t en mode classique
typedef struct {
  int rang;
  char visible;
  COULEUR couleur;
  }  element_t;

typedef element_t CARTE;
#endif

typedef CARTE element_t;

void element_print (CARTE e);
void element_init (CARTE* e);
CARTE element_empty();
int element_is_empty(CARTE e);
int element_equal(CARTE*, CARTE*);
#endif
```

4.2 Fonctions d'affichage graphique des piles et files de cartes

Les fonctions suivantes sont utiles pour afficher dans une fenêtre graphique.

affichage d'une file `void fifo_draw(fifo_t L, SDL_PHWindow* f1, int x, int y)` ; La fonction dessine la file de cartes Ldans la fenêtre `f1` à partir de la position `x,y`

affichage d'une pile `void lifo_draw(fifo_t L, SDL_PHWindow* f1, int x, int y)` ; La fonction dessine la pile de cartes Ldans la fenêtre `f1` à partir de la position `x,y`

effacement d'une file `void fifo_erase(fifo_t L, SDL_PHWindow* f1, int x, int y)` ; La fonction efface la file de cartes Ldans la fenêtre `f1` à partir de la position `x,y`

effacement d'une pile `void lifo_erase(fifo_t L, SDL_PHWindow* f1, int x, int y)` ; La fonction efface la file de cartes Ldans la fenêtre `f1` à partir de la position `x,y`

Ces fonctions sont définies dans le fichier `graph_print.h` et leur code est dans le fichier `graph_print.c`.

```
// Fichier graph_print.h
#ifndef _GRAPH_PRINT
#define _GRAPH_PRINT

#include "fifo.h"
#include "lifo.h"
#include <SDL2/SDL_phelma.h>

  /* Decalage entre 2 cartes d'une pile ou file
  lors de l'affichage de celles-ci
  x est la position en abcisse de la pile ou file
  y est la position en ordonnee de la pile ou file
  le point de coordonnees 0,0 est en haut a gauche
  */
#define DECALAGE 10

  /* Affichage graphique d'une carte */
void card_draw(CARTE* e, SDL_PHWindow*  f1,int x, int y);

  /* Efface une carte */
void card_erase(CARTE* e, SDL_PHWindow*  f1,int x, int y);

  /* Affiche une pile */
void lifo_draw(lifo_t p, SDL_PHWindow*  f1, int x, int y) ;

  /* Efface une pile */
void lifo_erase(lifo_t p, SDL_PHWindow*  f1, int x, int y);

  /* Affiche une file implemente par une liste circulaire */
void fifo_draw(fifo_t L, SDL_PHWindow*  f1, int x, int y) ;

  /* Efface toute une file */
void fifo_erase(fifo_t L, SDL_PHWindow*  f1, int x, int y) ;
#endif
```

4.3 Exemple d'affichage de pile et file de cartes en version graphique

Le fichier `Makefile` fournit prend en compte la possibilité d'un mode graphique sans modifier le fichier `carte.h`. Il suffit de compiler avec la commande `make bataille MODE=GRAPHIQUE`.

Lorsque vous passez d'un mode à un autre (graphique à non graphique et réciproquement), il faut absolument supprimer les anciennes versions des codes compilés (fichiers `*.o`) par la comande `make clean`.

Le fichier `exemple.c` illustre l'utilisation des fonctions graphiques qui permettent d'afficher ou d'effacer une pile ou une file. Il effectue les actions suivantes :

1. Cree 2 piles vides, 2 files vides

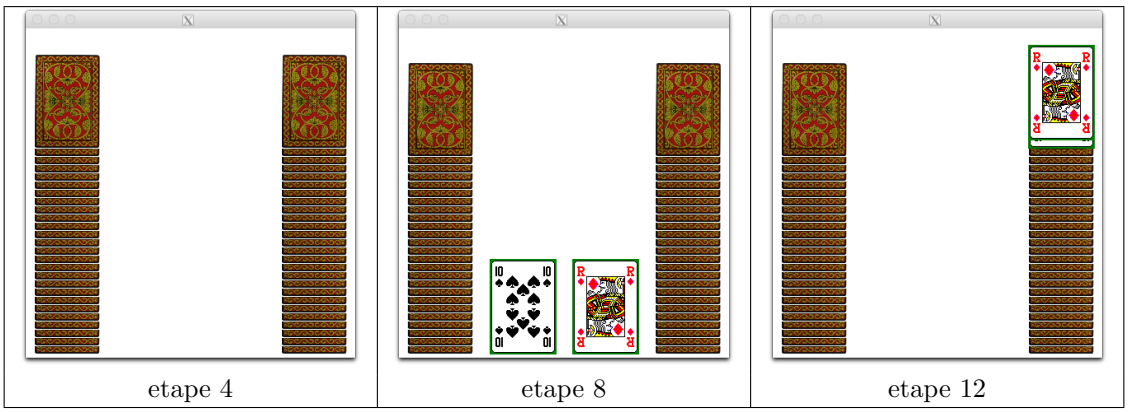

FIGURE 1 – 3 etapes du fichier exemple.c

2. Ouvre une fenêtre graphique
   `f1=newfenetregraphique(DIMX,DIMY);`
3. Distribue les cartes dans les 2 jeux
   `distribution(&j1,&j2,1,13);`
4. Affiche les 2 piles et les 2 files
   `fifo_draw(j1,f1,POS_X_FILE1,POS_Y_PILE_FILE);`
5. Efface les 2 piles et les 2 files
   `fifo_erase(j1,f1,POS_X_FILE1,POS_Y_PILE_FILE);`
6. Défile une carte du jeu 1, l'empile face visible sur la pile 1
   `c=fifo_dequeue(&j1); c.visible=1 ; p1=lifo_push(c,p1);`
7. Défile une carte du jeu 2, l'empile face visible sur la pile 2
8. Affiche les 2 piles et les 2 files
9. Dé finit est le gagnant
10. Efface les 2 piles
11. Dépile la pile 1 et enfile la carte dans le jeu gagnant
12. Dépile la pile 2 et enfile la carte dans le jeu gagnant
13. Affiche les 2 files

Pour compiler, il faut utiliser la commande `make exemple MODE=GRAPHIQUE` ou bien définir la constante `MODEGRAPHIQUE` dans le fichier `carte.h` par une ligne du type `#define MODEGRAPHIQUE 1`

```
#include "fifo.h"
#include "lifo.h"
#include "element.h"
#include "distribution.h"
#include "graph_print.h"

/* Dimension de la fenetre */
#define  DIMX 500
#define  DIMY 700

/* Posiiton en X du jeu du joueur 1 */
#define POS_X_FILE1 50
/* Posiiton en X de la pile de batailleu du joueur 1 */
#define POS_X_PILE1 150
/* Posiiton en X du jeu du joueur 2 */
#define POS_X_FILE2 350
/* Posiiton en X de la pile de batailleu du joueur 2 */
```

```
#define POS_X_PILE2 250
#define POS_Y (DIMY-120)


int  main(int argc, char** argv) {  CARTE c;
  fifo_t j1=fifo_new(),j2=fifo_new();
  lifo_t p1=lifo_new(),p2=lifo_new();
        // La variable pour manipuler une fenetre graphique
  SDL_PHWindow* f1=NULL;
        /* On cree une fenetre graphique */
  f1=SDL_PH_CreateWindow(DIMX,DIMY);
  if (f1==NULL) { puts("Impossible d'ouvrir une fenetre"); exit(EXIT_FAILURE); }

        /* On cree et melange toutes les cartes (ici 6*4), avec leur representation sous forme d'image
   dans la file jeu*/
   if (distribution(&j1,&j2,1,13)==0) {
     printf("Tous les fichiers images de cartes ne sont pas disponibles\n");
     exit(EXIT_FAILURE);
   }

        /* On affiche le jeu e du joueur 1*/
  fifo_draw(j1,f1,POS_X_FILE1,POS_Y);
        /* On affiche le jeu e du joueur 2*/
  fifo_draw(j2,f1,POS_X_FILE2,POS_Y);
       /* On affiche la pile du joueur 1*/
  lifo_draw(p1,f1,POS_X_PILE1,POS_Y);
        /* On affiche la pile du joueur 2*/
  lifo_draw(p2,f1,POS_X_PILE2,POS_Y);

  /* Juste pour ne pas  quitter tout de suite*/
  puts("Taper Return pour continuer");    getchar();
  puts("On prend une carte de chaque joueur");
  puts("On la met sur sa pile");

  /* On efface les piles et les files des joueurs */
  lifo_erase(p1,f1,POS_X_PILE1,POS_Y);
  lifo_erase(p2,f1,POS_X_PILE2,POS_Y);
  fifo_erase(j1,f1,POS_X_FILE1,POS_Y);
  fifo_erase(j2,f1,POS_X_FILE2,POS_Y);

  /* On remplit les  piles avec les premieres cartes face visible  de chaque joueur */
  c=fifo_dequeue(&j1); c.visible=1 ; p1=lifo_push(c,p1);
  c=fifo_dequeue(&j2); c.visible=1 ; p2=lifo_push(c,p2);

  /* On affiche les nouvelles piles et files des joueurs */
  lifo_draw(p1,f1,POS_X_PILE1,POS_Y);
  lifo_draw(p2,f1,POS_X_PILE2,POS_Y);
  fifo_draw(j1,f1,POS_X_FILE1,POS_Y);
  fifo_draw(j2,f1,POS_X_FILE2,POS_Y);
  /* On attend */
  puts("Taper Return pour continuer");    getchar();
  /* la file pointee par gagnant designe le joueur qui gnagne les 2 cartes */
  fifo_t* gagnant=NULL;
  if (lifo_peek(p1).rang > lifo_peek(p2).rang) {
    puts("le joueur 1 gagne les cartes");
    gagnant=&j1;
  }
  else if (lifo_peek(p1).rang < lifo_peek(p2).rang) {
    puts("le joueur 2 gagne les cartes");
    gagnant=&j2;
  }
  else {
    puts("bataille");
  }

  /* Gain des cartes par le gagnant s'il existe */
  if (gagnant) {
    /* On efface les piles et les files gagnees par un joueur  */
    lifo_erase(p1,f1,POS_X_PILE1,POS_Y);
    lifo_erase(p2,f1,POS_X_PILE2,POS_Y);
```

```
    /* On met les cartes des piles dans la file du gagnant
    On garde les cartes face visible pour montrer leurs
    destinations sur cet exemple */
    c=lifo_pop(&p1); c.visible=1; *gagnant=fifo_enqueue(c,*gagnant);
    c=lifo_pop(&p2); c.visible=1; *gagnant=fifo_enqueue(c,*gagnant);
    /* On affiche les nouvelles  files des joueurs  */
    fifo_draw(j1,f1,POS_X_FILE1,POS_Y);
    fifo_draw(j2,f1,POS_X_FILE2,POS_Y);
  }

  puts("Taper Return pour continuer");    getchar();

}
```

4.4 Pour réaliser une version graphique

1. Le répertoire qui contient les fichiers images des cartes est spécifié dans le fichier `distribution.h` par la ligne `#define IMAGE_PATH /users/prog1a/C/librairie/cartes/`. Ne pas le changer à moins de travailler sur sa propre machine : dans ce cas, il doit indiquer votre répertoire contenant ces fichiers.
2. Supprimer les anciennes versions compilées par la commande `make clean`
3. Compiler l'exemple à l'aide de la commande `make exemple MODE=GRAPHIQUE`
4. Comprendre le code et exécuter l'exemple ci dessous
5. Gérer un affichage graphique des différents éléments du jeu de bataille en modifiant votre programme `bataille.c` à l'aide des fonctions utilisées de l'exemple ci dessus. Il faut simplement créer une fenêtre graphique et remplacer l'affichage en mode texte par les fonctions de dessin et d'effacement des piles et files au bon endroit.
6. Vous pouvez compiler à l'aide de la commande `make bataille MODE=GRAPHIQUE`