ISIMA
INSTITUT D'INFORMATIQUE
Université Clermont Auvergne

# Compte rendu - La Bataille

Lilian Hiault

11 mai 2019

## Table des matières



## Introduction

« La bataille » est un exercice disponible sur Codingame (`https://www.codingame.com/ide/puzzle/winamax-battle`). C'est un jeu de carte très connu (et qui demande beaucoup de stratégie !).
Ici, on nous donne la distribution des jeux de cartes des deux joueurs et le but de de savoir lequel des joueurs sera le gagnant et combien de manches seront nécesaires à sa victoire.

## 1 Le programme principal

### 1.1 Structure

Afin de stocker et de mettre à jour facilement les tas de cartes j'ai créé une structure tas qui dispose de plusieurs champs.
« cartes » est un tableau d'entier dans lequel je dispose les cartes du joueur. Le champs taille correspond à la taille du tableau, qui sera importante lorsqu'on aura besoin de l'augmenter.
Début et fin sont deux curseurs qui correspondent aux indices du début et de la fin du jeu du joueur (les cartes qui sont dans son tas). Début correspond donc à la carte qui est au dessus de la pile tandis que fin correspond à l'indice de la prochaine case à remplir lorsqu'il faudra rajouter des cartes.

```
typedef struct tas {
  int debut;
  int fin;
  int taille;
  int *cartes;
} tas;
```

## 1.2 Les cartes en nombres

Afin de simplifier l'éxecution du programme j'ai créé une fonction qui transforme les chaînes de caractères reçues en entrées en des nombres entiers. Par exemple le roi de cœur noté : "KH" deviendra simplement un 13.

```
int conversionCartes(char carte[])
{
  int valeur = -1;
  switch (carte[0]) {
    case '2':
    valeur = 2;
    break;
    case '3':
    valeur = 3;
    break;
    case '4':
    valeur = 4;
    break;

    ...

    case 'K':
    valeur = 13;
    break;
    case 'A':
    valeur = 14;
    break;
  }
  return valeur;
}
```

## 1.3 Le programme principal

Une fois que l'on obtient la taille de la pile de cartes du joueur, on peut créer une structure tas pour le joueur, qu'on initialise. Une fois fait, il faut replir la tableau correspondant aux cartes du joueur. On le rempli de la dernière à la première case car la première carte distribuée est la dernière dans la pile de cartes.

```
int main()
{
  int n=0;
  scanf("%d", &n);
  tas j1 = {0, n, n, (int*) malloc(n*sizeof(int))};
  for (int i = n-1; i >= 0; i--) {
    char cartesJ1[4]; // les n cartes du joueur 1
    scanf("%s", cartesJ1);
    j1.cartes[i] = conversionCartes(cartesJ1);
  }
```

On utilise aux variables près le même code pour le joueur 2.

Je crée un compteur pour les manches et des pointeurs vers les structures qui désignent les tas de cartes des joueurs.

```
  int nbManches = 0;
  tas *pJ1 = &j1;
  tas *pJ2 = &j2;
```

Il y a des manches jusqu'à ce qu'un des deux joueurs n'ait plus de cartes. Dans ce cas l'autre est déclaré vainqueur.

```
while ((j1.fin != j1.debut) || (j2.fin != j2.debut))
  {
    manche(pJ1, pJ2);
    nbManches ++;
  }

  if ((j1.debut == j1.fin) && (j2.debut != j2.fin))
  {
    printf("2 %d\n", nbManches);
  }
  else if ((j2.debut == j2.fin) && (j1.debut != j1.fin))
  {
    printf("1 %d\n", nbManches);
  }
  else
  {
    printf("PAT %d\n", nbManches);
  }
  return 0;
}
```

# 2 Fonctionement d'une manche

## 2.1 Avoir des tableaux assez grands

A chaque fois qu'un joueur gagne des cartes, il faut les stocker à la suite dans le tableau. Or s'il perd des cartes puis en regagne à nouveau, les curseurs début et fin avancent tout de même. On arrive alors rapidement à la fin du tableau. Pour y remédier, j'ai créé une fonction qui

lorsqu'elle est appelée sur une structure tas, modifie le tableau afin de doubler sa taille.

```
void tailleTab(tas *j)
{
  if (j->fin >= j->taille-1 || j->debut >= j->taille-1)
  {
    int *tab = (int*) malloc(2*j->taille*sizeof(int));
    j->taille = 2*j->taille;
    int i;
    for (i=0; i<j->taille/2; i++)
    {
      tab[i] = j->cartes[i];
    }
    free (j->cartes);
    j->cartes = tab;
  }
}
```

## 2.2 Une manche classique

On compare la carte du dessus de la pile, c'est à dire la valeur du tableau de cartes à l'indice début, avec l'autre joueur. Celui qui remporte le pli gagne alors deux cartes (fin augmente de 2) et les deux joueurs perdent la carte du dessus de leur bibliothèque (début est incrémenté pour les deux joueurs). A ce moment j'utilise ma fonction pour vérifier que les tableaux ont une taille suffisante et, le cas échéant augmenter leur taille. Une fois qu'il y a assez de place, on peut ajouter les nouvelles cartes à la fin du tableau du vainqueur.

```
void manche (tas *j1, tas *j2)
{
  if (j1->cartes[j1->debut] > j2->cartes[j2->debut]) // J1 gagne la manche
  {
    j1->fin += 2;
    j1->debut ++;
    j2->debut ++;
    tailleTab(j1);
    tailleTab(j2);
    j1->cartes[j1->fin -2] = j1->cartes[j1->debut-1];
    j1->cartes[j1->fin -1] = j2->cartes[j2->debut-1];
  }
```

Si c'est le joueur 2 qui emporte la manche, le programme qui est pratiquement le même en changeant juste le joueur.

## 2.3 La bataille

Si les deux joueurs ont une carte de même valeur alors s'ensuit une bataille.

```
  else if (j1->cartes[j1->debut] == j2->cartes[j2->debut])
  {
    bataille();
  }
}
```

Les deux joueurs se défaussent de trois cartes puis comparent à nouveau une nouvelle carte jusqu'à ce qu'un de joueurs l'emporte et s'empare de toutes les cartes défaussées de cette façon. Malheureusement je n'ai pas trouvé de moyen d'implémenter cela dans mon programme.

# Conclusion

Mon programme ne tient pas compte des batailles lorsque les joueurs ont une carte de même valeur. Toutefois pour des cas basiques où cela n'apparaît pas, il fonctionne.