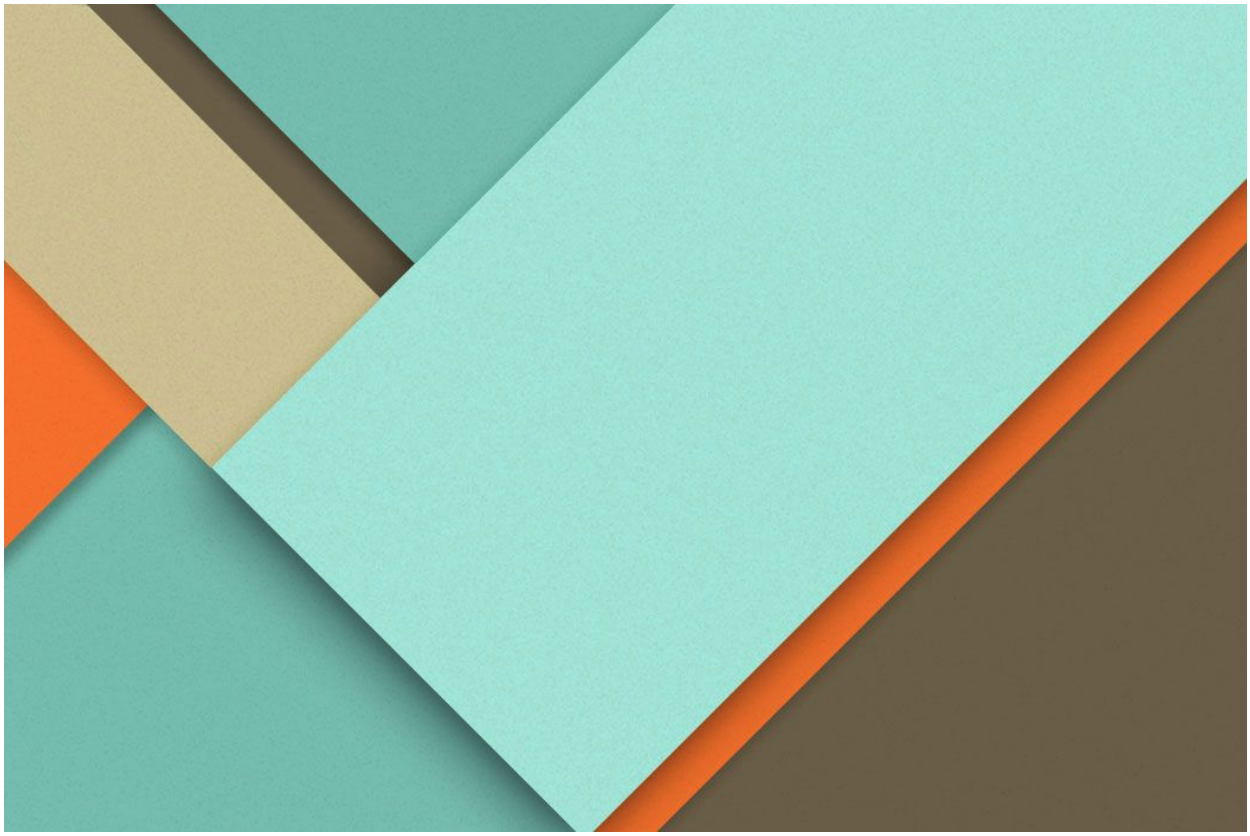


Les petits chevaux

Réalisé par :

- Abel MUWALA

- Nabil AISSAT

Encadré par :

M. petit Albin

TABLE DES MATIÈRES

INTRODUCTION

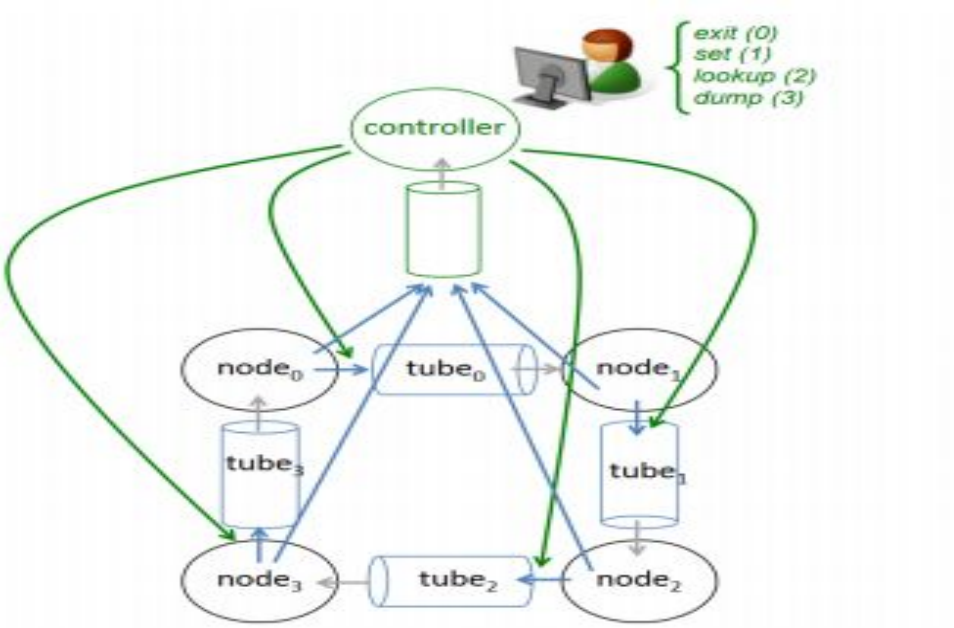
Nous avons réalisé une application en C qui permet de jouer au jeu de petits chevaux,chaque joueur est un processus distinct et les joueurs communiquent via des tubes.

Ce travail est essentiellement basé sur la gestion de la communication via des **“PIPES”** entre des processus **“DISTINCTS”.**

Nous avons également travaillé sur la **modélisation** du jeu et **l’implémentation** de quelques règles de ce jeu.

Architecture globale

On a adopté une architecture de type **MAÎTRE-ESCLAVE;** dans lequel le processus père (désigné par **controller** sur le schéma) s’assure de la synchronisation du jeu et de la gestion des conditions d’arrêt.



INSTRUCTIONS

Pour chaque version du programme, il y'a un dossier portant le nom de version contenant les dossiers suivants :

• **Bin** -> dossier contenant les exécutables compilés.

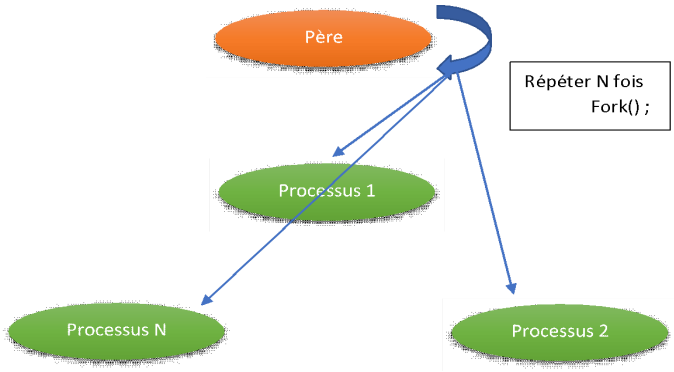
• **Headers**-> fichiers headers contenant les fctsProjet.h pour les versions qui l'utilisent .

• **Src** -> fichiers de source c et fctsProjet.c pour les versions qui l'utilisent chaque dossier contient aussi un makefile qui permet de compiler automatiquement les différentes versions à l'aide de la commande « make », on peut aussi supprimer automatiquement tous les fichiers objet à l'aide de la commande « make clean ».

RÉALISATION

Partie 1 création de N fils

Nous créons un réseau de N fils correspondant aux N joueurs à partir d'un processus père.



Pour cela nous avons simplement décidé d'utiliser un **fork** en boucle puis demander au fils de s'identifier à l'aide d'un printf pour vérifier qu'il est bien créé. On affiche aussi le pid du fils ainsi que le pid du père ce qui nous permet de nous assurer que nous ne créons pas de zombie.

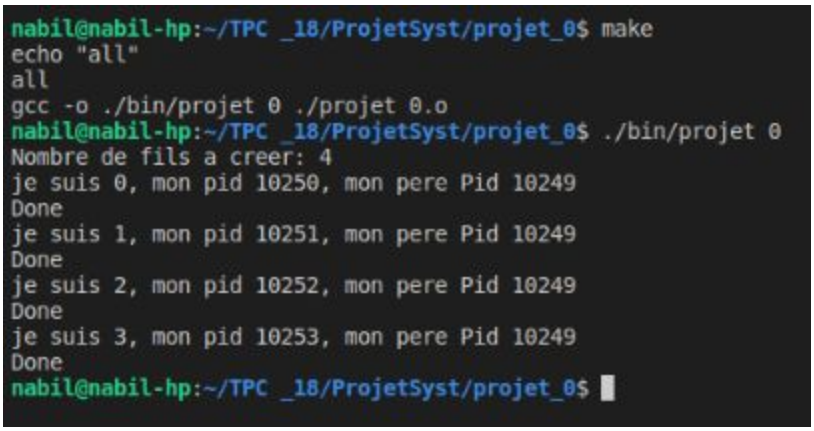
projet_0.c

CODE

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <errno.h>

int main(int argc, char *argv[])
{
    int processNumber;

   printf("Nombre de fils à créer: ");
   scanf("%d", &processNumber);
    for(int i = 0; i < processNumber; i++){
        switch (fork()){
            case -1 : // ERREUR
                fprintf(stderr, "Erreur de fork\n");
                exit(-1);
            case 0 : // Création des fils
```

```
                printf("je suis %d, mon pid %d, mon pere Pid %d\n", i,
getpid(), getppid());
                exit(-1);

/*********************************************************************************/
            default :
            /* comportement du père */
                while (wait(NULL)!=-1);
                printf("Done\n");
        }
    }
   return 0;
}
```

APERÇU



Partie 2 : communication entre les processus *Projet_1.c*

Cette partie est divisée en deux sous parties :

a) Communication entre les joueurs (processus fils) sous forme d'**ANNEAU**

b) communication entre les joueurs et le processus **MAÎTRE**

1- communication entre N fils :

Nous devons faire en sorte que nos fils soient capables de communiquer entre eux en se transmettant un message. Nous avons choisis d'initialiser une liste de pipe aussi longue que le nombre de processus que l'utilisateur indique. Nous avons séparé cette boucle et celle de la création des fils pour « alléger » le code de la boucle du fils et éviter de provoquer des comportements inattendus à cause du fonctionnement de fork

Nombre de pipes utiles = N.

```
/*------------- Création des pipes ----------*/
int **creationPipe(int processNumber){
   int i;
   int **pipeAnn = malloc(sizeof(*pipeAnn) * processNumber);
   for(int i=0;i<processNumber;i++){
       pipeAnn[i] = malloc(sizeof(**pipeAnn) * 2);
        if(pipe(pipeAnn[i])==-1){
       perror ("pipe") ;
       exit (-1) ;
    }
    }
   return pipeAnn;
}
```

Chaque fils écrit sur le pipe pipeAnn[i][1] (i numéro du fils) et lit le pipe pipAnn[i-1][0]. Sauf le fils 0 qui lit sur le pipe tab[processNumber-1][0]. La difficulté se trouve dans la gestion de la fermeture des différents fichiers. Si un fichier n'est pas fermé au bon endroit ou s'il est fermé alors qu'il ne devrait pas l'être on créera des erreurs telles que : bad file descriptor, ou même des plantages où un processus attend indéfiniment de pouvoir lire ou écrire sur un pipe.

```
/*envoi de l'information dans l'anneau*/
void envoiFilsFils(int val,int i,int ** pipeAnn){
       write(pipeAnn[i][1], &val, sizeof(int));
}
/*lecture de l'information dans l'anneau*/
int receptionFilsFils(int i, int processNumber,int ** pipeAnn){
```

```
    int val;
    if(i==0){
     read (pipeAnn[processNumber-1][0], &val, sizeof (int));
    }else{
      read (pipeAnn[i-1][0], &val, sizeof (int));
    }
    return val;
}
```

Pour tester le fonctionnement de la communication inter-processus nous avons fait en sorte que le premier processus écrive dans le pipe pipeAnn[i][1] un message de test (**son pid**) Ensuite chaque processus lit et écrit le résultat de sa lecture. On vérifie donc que la communication est fonctionnelle.

```
for (int i = 0; i < processNumber; i++)
   {
       switch (numFils[i] = fork())
       {
       case -1:
           /**--------Cas erreur fork----------*/
           perror("\x1B[31m Erreur de fork\n");
           exit(-1);
       case 0: /**--------Cas comportement fils*/
           //traitement des descripteurs
           closeDescripteurFils(i, processNumber, pipeAnn);

 if (i == 0)
  {
      envoiFilsFils(getpid(), i, pipeAnn);
      int retour = receptionFilsFils(i, processNumber, pipeAnn);
      printf("\x1b[33m je suis le fils %d, mon pid :%d j'ai reçu la valeur %d
                 :\n", i, getpid(), retour);
   }
  else
    {
    int jeu = receptionFilsFils(i, processNumber, pipeAnn);
    printf("\x1B[32m je suis le fils %d, mon pid :%d j'ai reçu la valeur %d
             :\n", i, getpid(), jeu);
    envoiFilsFils(jeu, i, pipeAnn);

     }
```

APERÇU

```
nabil@nabil-hp:~/TPC _18/ProjetSyst/projet_1$ ./bin/projet 1
Nombre de fils a creer : 4
 je suis le fils 1, mon pid :10187 j'ai recu la valeur 10186 :
 je suis le fils 2, mon pid :10188 j'ai recu la valeur 10186 :
 je suis le fils 3, mon pid :10189 j'ai recu la valeur 10186 :
 je suis le fils 0, mon pid :10186 j'ai recu la valeur 10186 :
 Done nabil@nabil-hp:~/TPC _18/ProjetSyst/projet_1$
```

2- communication entre N fils et le père : Nous avons choisis d'initialiser une liste de pipe aussi longue que le nombre de processus que l'utilisateur indique pour permettre la communication de père vers ses N fils . Nous avons aussi utiliser un pipe pour la communication des N fils vers le père .

Nombre de pipes utiles = N+1.

```c
/* Ecriture du père vers les fils */
void ecriturePereFils(int processNumber, int val, int **pipePereFils)
{
   for (int i = 0; i < processNumber; i++)
   {
       write(pipePereFils[i][1], &val, sizeof(int));
   }
}

/*Ecriture du fils au père*/
void ecritureFilsPere(int val, int pipeFilsPere[2])
{
   write(pipeFilsPere[1], &val, sizeof(int));
}

/* fonction qui permet au fils de lire ce qui est envoyé par le père */
static int lectureDuPere(int descr)
{
```

```
    int val;
     read(descr, &val, sizeof(int));
    return val;
}

/* fonction qui permet au père de lire  ce qui est envoyé par le fils*/
 static int lectureDuFils(int descr)
 {
    int val;
    read(descr, &val, sizeof(int));
    return val;
 }
```

Pour tester le fonctionnement de la communication inter-processus (père vers fils , fils vers père et fils vers fils) nous avons fait en sorte que le père envois à chacun de ses fils l'un des PID de ces derniers , chaque fils doit tester si le PID est le mien si c'est le cas il envois la valeur 100 à son voisin , et quand l'informations fait le tours des fils elle sera envoyé au père par le fils qui a le PID

```
/* Envoi de l'information par le père */
ecriturePereFils(processNumber, numFils[0], pipePereFils);
/* lecture de  l'information par chaque fils et tester la valeur lu  */
int val = lectureDuPere(pipePereFils[i][0]);
            printf("\x1B[32m je suis le fils %d, mon PID %d, j'ai reçu de père %d
\n",i, getpid(), val);
           if (val == getpid())
           {
               envoiFilsFils(100, i, pipeAnn);
               int retour = receptionFilsFils(i, processNumber, pipeAnn);
               printf("\x1B[32m Je suis le fils PID %d, et j'ai reçu %d\n",
getpid(), retour);
               ecritureFilsPere(retour, pipeFilsPere);

           }
           else
           {
               int jeu = receptionFilsFils(i, processNumber, pipeAnn);
               printf("\x1B[32m Je suis le fils PID %d, et j'ai reçu %d\n",
getpid(), jeu);
               envoiFilsFils(jeu, i, pipeAnn);
           }
```

```
/* lecture de l'information par le fils  */

int val = lectureDuFils(pipeFilsPere[0]);
   printf("\x1B[34m je suis le père et j'ai reçu du fils %d \n", val);
```

APERÇU

Partie 3 finale

```
nabil@nabil-hp:~/TPC _18/ProjetSyst/projet_1$ make
echo "projet 1.o"
projet 1.o
gcc -c ./src/projet 1.c -I ./headers
echo "all"
all
gcc -o ./bin/projet 1 ./projet 1.o
nabil@nabil-hp:~/TPC _18/ProjetSyst/projet_1$ ./bin/projet 1
Nombre de fils a creer : 4
 je suis le père et les PID de mes fils sont:   13326   13327   13328   13329
 Je suis le père et j'envoie à mes fils le PID : 13326
 je suis le fils 0, mon PID 13326, j'ai recu de père 13326
 je suis le fils 2, mon PID 13328, j'ai recu de père 13326
 je suis le fils 1, mon PID 13327, j'ai recu de père 13326
 je suis le fils 3, mon PID 13329, j'ai recu de père 13326
 Je suis le fils PID 13327, et j'ai recu 100
 Je suis le fils PID 13328, et j'ai recu 100
 Je suis le fils PID 13329, et j'ai recu 100
 Je suis le fils PID 13326, et j'ai recu 100
 je suis le père et j'ai recu du fils 100
 Done
nabil@nabil-hp:~/TPC _18/ProjetSyst/projet_1$
```

version

les version différentes

Enfin après « prototype » nous avons passé à la finalisation de l'application et de la mise en place de jeu des petits chevaux.

Nous avons, dans cette partie, réalisé :

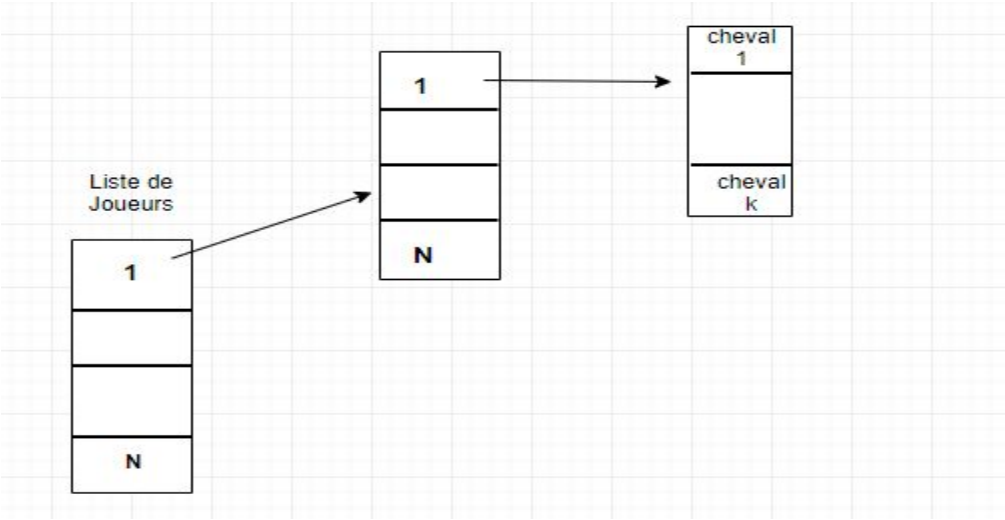
a) La modélisation du jeu
b) L'implémentation des règles
c) La gestion des conditions d'arrêt

A) Modélisation

Nous avons modélisé le jeu sous forme d'un tableau à 3 Dimensions.

Chaque case du tableau initiale contient un tableau 2D ,contenant pour chaque case l'état du jeu de chaque joueur. Enfin nous avons les cases représentant les chevaux pour chaque joueur.

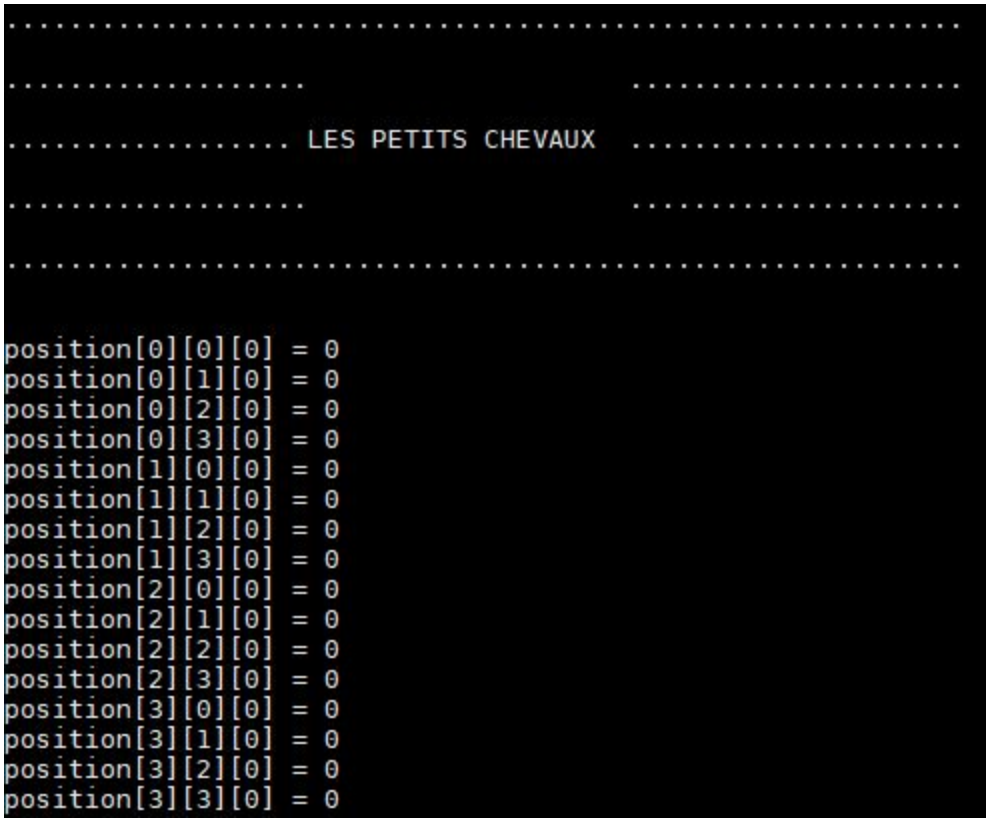
Nous avons pour cela utilisé une fonction d'initialisation du jeu ; pour la création de la table de stockage et la mise à zéro de toutes les cases.



Fonction d'initialisation du jeu

```
int*** initJeu(int processNumber){
    int  c=1,i,j,k;
    int ***A;
   A = (int***)malloc(sizeof(int**) *processNumber);
   for (i=0;i<processNumber;i++){
      A[i] = (int**)malloc(sizeof(int*) *processNumber);
         for (j=0;j<processNumber;j++){
           A[i][j] = (int*)malloc(sizeof(int)*1);
       }
   }
   for (i=0;i<processNumber;i++){
     for (j=0;j<processNumber;j++){
         for (k=0;k<c;k++){
             A[i][j][k]=0;
           printf("A[%i][%i][%i] = %d\n", i, j,k, A[i][j][k]); //
affichage des élément du tableau
         }
      }
 }
 return A;
}
```

APERÇU Pour 4 joueurs et 1 cheval pour chacun



B) IMPLEMENTATION

Une fonction de mise à jour des tables

Fonction sera exécutée par les processus fils après avoir effectué un **rand**

cette fonction retourne 1 si le cheval arrive à la case 56

sinon 0 et le jeu devra continuer

```
int majJeuPrincipal(int i,int de,int processNumber, int *** position
){
   int val= position[i][i][0]+de;
 // printf(" val %d\n", val);
   if(val>=56){
       position[i][i][0]=56;
       return 1;

   }else{
      for(int j=0;j<processNumber;j++){
```

```
            if(position[i][j][0]==val){
            position[i][j][0]=0;
             }
        }
         position[i][i][0]=val;

         return 0;
    }
}
```

FONCTION POUR LE jeu

```
int jouer(int i,int processNumber,int numPlayer, int **pipeAnn,int
**pipePereFils,int *** position){
   int de,controle;
   int sortie;

    if(numPlayer==getpid()){
       de=1+rand() % 6;
      int maj=majJeuPrincipal(i,de,processNumber,position);
            /*printf("\n
");
            printf("\n joueur %d |valeur dé = %d ",getpid(),de);

printf("\n*****************************************\n");*/
      if(maj==1)
      de=0;
      envoiFilsFils(i,i,pipeAnn);
      envoiFilsFils(de,i,pipeAnn);
      int r=receptionFilsFils(i,processNumber,pipeAnn);
      int retour=receptionFilsFils(i,processNumber,pipeAnn);

      if(retour==0){
             controle=1;
           //  printf ("\x1B[32m GAGANT %d\n",getpid()) ;

         }else if(retour==6){
```

```
            controle=2;
          }else{
               controle=0;
                   }
        }else{
            int num=receptionFilsFils(i,processNumber,pipeAnn);
            int jeu=receptionFilsFils(i,processNumber,pipeAnn);

              if(jeu==0){

                printf ("\x1B[32m PERDANT %d\n",getpid()) ;

              }else{
                     int val= position[i][num][0]+jeu;
                     position[i][num][0]=val;
                }
                envoiFilsFils(num,i,pipeAnn);
                envoiFilsFils(jeu,i,pipeAnn);
    }

  return controle;

}
```