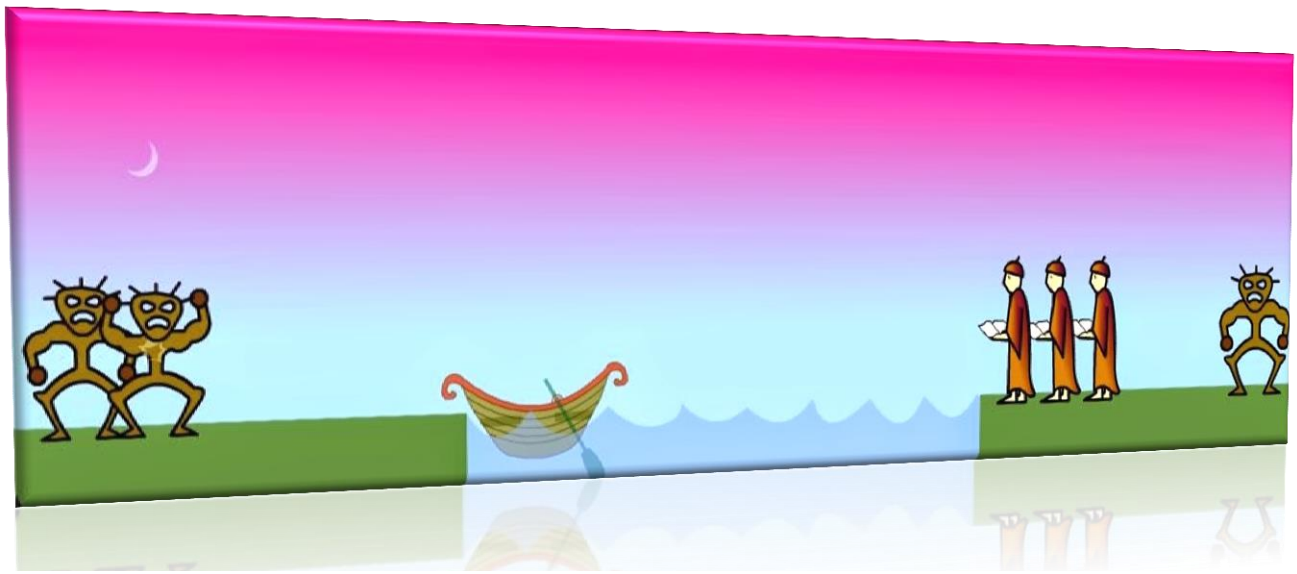
RAPPORT DE TP



# TP DES MISSIONNAIRES ET DES CANNIBALES

ASSAFI ISSAM 1GI

# I – MODÉLISATION DU PROBLÈME

## • PRÉSENTATION DE JEU

Vous devez faire traverser un fleuve à 3 cannibales et 3 missionnaires. Si à un endroit ou un autre, il y'a plus de cannibales que de missionnaires, les premiers mangent les seconds. La barque qui leur permet de passer d'une rive à l'autre ne peut contenir que 2 personnes. Comment les faire traverser en toute sécurité, sachant qu'il faut quand même toujours quelqu'un pour ramener la barque?

## • ESPACE D'ÉTATS

L'espace des état sera décrit par des triplets (P,X,Y) où :

P : la position de la barque (elle sera de type char, 'G' pour gauche et 'D' pour droit)

X : Nombre des missionnaires sur la rive droite (0<=X<=3)

Y : Nombre des missionnaire sur la rive gauche (0<=Y<=3)

Suivant les règles de jeu, la condition suivante doit être satisfaite :

X=0 ou X>=Y

## • ÉTAT INITIAL ET FINAL

Etat Initial = (D,3,3)

Etat Final = (G,0,0)

But = Trouver le chemin qui mène de l'état initial vers l'état final

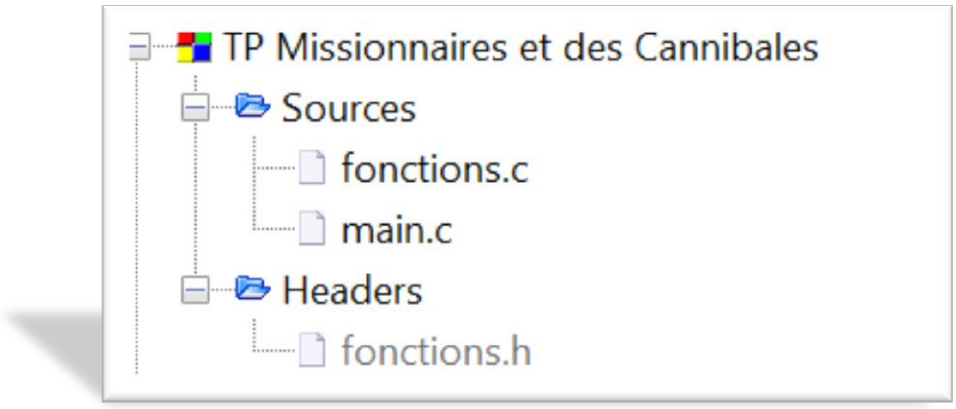
## • RÈGLES À SUIVRE

Vu que seulement 2 au maximum peuvent être sur la barque en même temps, donc voici les différentes possibilités :

| Règle | Missionnaire | Cannibales |
|---|---|---|
| R10 | 1 | 0 |
| R01 | 0 | 1 |
| R02 | 0 | 2 |
| R20 | 2 | 0 |
| R11 | 1 | 1 |

# II – CODE SOURCE ET IMPLÉMENTATION

## • STRUCTURE DE PROJET




Le code source est divisé en différentes partie, chacune et sa fonction :

**Main.c :** c'est le fichier principal où on déclenche la recherche de solution.

**Fonctions.c :** c'est le fichier contenant toutes les fonctions (et leurs descriptions) nécessaires au fonctionnement de programme

**Fonctions.h :** C'est le fichier contenant les déclarations des fonctions et les structures de données qu'on utilise.

## • CONTENU DE MAIN.C

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "fonctions.h"


int main()
{
    /* Création de la pile contenant les noeuds */
    Pile *listeNoeuds=NULL;
    listeNoeuds=(Pile*)malloc(sizeof(Pile));

    /* Déclaration des états : initial et final */
    etat etatInitial={'D',3,3};
    etat etatFinal={'G',0,0};

    /* Application de la recherche en profondeur */
    RechercheProfondeur(etatInitial, etatFinal, generesuccesseurs);

    return 0;
}
```

On déclare une pile vu qu'on travaille avec une recherche en profondeur, et on définit l'état initial et final, puis on appelle la fonction RechercheProfondeur qui s'occupera d'afficher le chemin menant à la solution.

- **CONTENU DE FONCTIONS.H**

```c
#ifndef FONCTIONS_H_INCLUDED
#define FONCTIONS_H_INCLUDED

typedef struct
{
    char position; // Position de la barque, 2 valeurs sont possibles 'G' pour gauche et 'D' pour droite.
    int missionn,cannib;// Nombre des cannibales et missionnaire se trouvant dans la rive droite

}etat;

typedef struct Noeud
{
    struct Noeud *suivant;
    etat Etat;
}noeud;

typedef noeud* Pile;

Pile listeNoeuds; // Déclaration de la pile officielle qui contiendra les noeuds

/* C'est l'algorithme de recherche en profondeur en utilisant une pile */
void RechercheProfondeur( etat etatInitial, etat etatFinal,void generesuccesseurs());
int etatSolution(etat etatCourant);             // Retourne 1 si etatCourant est une solution effective
void generesuccesseurs(etat E);                 // génération des successeurs à partir d'un noeud en suivant les règles de jeu
int appartient(etat etatCourant, Pile liste);   // Vérifi si etatCourant appartient à la pile, on retourne 1 dans ce cas, sinon 0
etat extraire(Pile *listeNoeuds);               // Opération de dépiler
void insertion(etat etatCourant,Pile *liste);   // Insertion d'un nouveau noeud dans la pile
int vide(Pile listeCourante);                   // Véerifie si la pile est vide, retourne 1 dans ce cas sinon 0
void affichageGraphique(etat E);                // un affichage graphique montrant les diffirentes étapes pour atteindre le résultat f

#endif // FONCTIONS_H_INCLUDED
```

On crée les différentes structure de données, et on déclare les multiple fonctions qu'on va employer (c'est leurs prototypes).

Structure de données :

« **Etat** » elle est définie comme une structure ayant 3 membres :

- *Position* : indique la position de la barque.
- *Missionn* : nombre de missionnaire dans la rive droite.
- *Cannib* : nombre de cannibales dans la rive droite.

Les fonctions employées :

- *RechercheProfondeur :* C'est l'algorithme général de recherche systématique en profondeur.
- *etatSolution* : Retourne 1 si etatCourant est une solution effective
- *generesuccesseurs :* génération des successeurs à partir d'un nœud en suivant les règles de jeu
- *appartient :* Vérifie si etatCourant appartient à la pile, on retourne 1 dans ce cas, sinon 0
- *extraire :* Opération de dépiler
- *insertion* : Insertion d'un nouveau nœud dans la pile
- *vide* : Vérifie si la pile est vide, retourne 1 dans ce cas sinon 0
- *affichageGraphique* : un affichage graphique montrant les différentes étapes pour atteindre le résultat final

- **CONTENU DE FONCTIONS.C**

C’est la programmation des différentes fonctions :

Par exemple la fonction ***generesuccesseurs*** qui prend en charge les règles et contraintes de jeu afin de générer des successeurs pour un nœud donné.

La condition qui doît être respecté est :

*((etatFils.missionn==0 ) || (etatFils.missionn==3) || (etatFils.missionn==etatFils.cannib))*

Cela garantie qu’on a jamais plus de cannibales que de missionnaires dans les deux rives.

```
void generesuccesseurs(etat E)
{
    char pos=E.position;
    int x=E.missionn;
    int y=E.cannib;
    etat etatFils;
    if(pos=='D')
    {
        etatFils.position='G';
        if(x>0)
        {
            etatFils.missionn=x-1;
            etatFils.cannib=y;
            if ((etatFils.missionn==0 ) || (etatFils.missionn==3) || (etatFils.missionn==etatFils.cannib))

                insertion(etatFils,&listeNoeuds);
            if(x>1)
            {
                etatFils.missionn=x-2;
                etatFils.cannib=y;
                if ((etatFils.missionn==0 ) || (etatFils.missionn==3) || (etatFils.missionn==etatFils.cannib))
                    insertion(etatFils,&listeNoeuds);
            }
            if(y>0)
```

Et les fonctions classiques (appartient, extraire, insertion...Etc.)

```
int appartient(etat etatCourant, Pile liste)
{
    Pile p=liste;
    while(p!=NULL)
    {
        if(p->Etat.cannib==etatCourant.cannib && p->Etat.missionn==etatCourant.missionn && p->Etat.position==etatCourant.position)
            return 1;
        p=p->suivant;
    }
    return 0;
}

etat extraire(Pile *listeNoeuds)
{
    etat etatResultat=(*listeNoeuds)->Etat;
    (*listeNoeuds)=(*listeNoeuds)->suivant;
        return etatResultat;
}


void insertion(etat etatCourant,Pile *liste)
{
    noeud *nouveau;
    nouveau=(noeud*)malloc(sizeof(noeud));
    nouveau->Etat=etatCourant;
    nouveau->suivant=*liste;
```

## • EXÉCUTION DE PROGRAMME

```
(D, 3, 3)    ( )              ------------------- |||||          (XXX OOO)
(G, 3, 1)    (XX )           ||||| -------------------          (X OOO)
(D, 3, 2)    (X )             ------------------- |||||          (XX OOO)
(G, 3, 0)    (XXX )          ||||| -------------------          ( OOO)
(D, 3, 1)    (XX )            ------------------- |||||          (X OOO)
(G, 1, 1)    (XX OO)         ||||| -------------------          (X O)
(D, 2, 2)    (X O)            ------------------- |||||          (XX OO)
(G, 0, 2)    (X OOO)         ||||| -------------------          (XX )
(D, 0, 3)    ( OOO)           ------------------- |||||          (XXX )
(G, 0, 1)    (XX OOO)        ||||| -------------------          (X )
(D, 0, 2)    (X OOO)          ------------------- |||||          (XX )
(G, 0, 0)    (XXX OOO)       ||||| -------------------          ( )

X= Cannibal, O= Missionnaire, ||||| = La Barque

Succes! Arret sur le noeud: (G, 0, 0)

Nombre de noeuds explores: 17, Noeuds traites: 11
```

- On a l'affichage de chemin à suivre pour résoudre le problème, et cela sous forme de multiple triplets (P,X,Y) qui se suivent (la signification de triplet est traité dans la partie modélisation), chaque triplet constitue une étape pour arriver à l'état final.

- le chemin solution est l'enchainement de ces étapes de haut vers le bas.

- Près de chaque étape, on a implémenté un affichage graphique sur la console qui permet de mieux décrire ce qui déroule lors de chaque mouvement. (veuillez noter que X = Cannibal, O=Missionnare et |||| = La barque ).

  Cela est assuré par la fonction *affichageGraphique*

```
void affichageGraphique(etat E)
{
    char pos = E.position;
    int x = E.missionn;
    int y = E.cannib;

    char barqueG[] = "||||| --------------------";
    char barqueD[] = " -------------------- |||||";

    int i;
    printf("(");
    for (i=0; i<3-y; i++) printf("X");
    printf(" ");
    for (i=0; i<3-x; i++) printf("O");
    printf(")\t\t");

    if (pos=='G') printf("%s",barqueG);
    else printf("%s",barqueD);

    printf("\t\t(");
    for (i=0; i<y; i++) printf("X");
    printf(" ");
    for (i=0; i<x; i++) printf("O");
    printf(")\n\n");


}
```