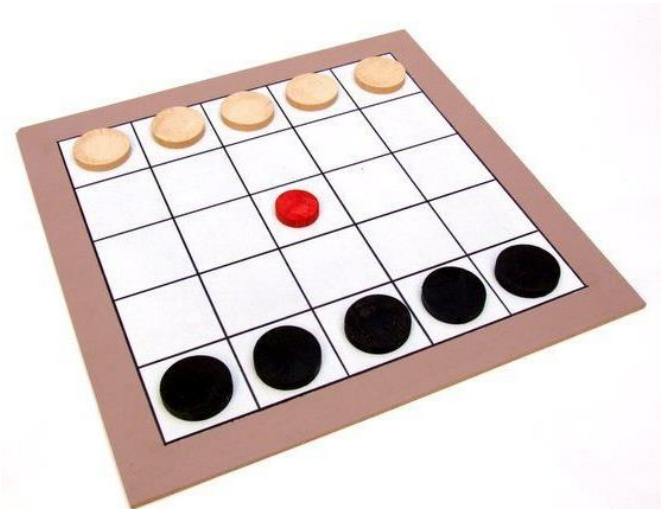
POLYTECH LILLE

Université de Lille

PROJET D'ALGORITHMIQUE ET PROGRAMMATION

LE NEUTRON



Claire Devisme - Julien Mahé

GIS 3

SOMMAIRE

Introduction

Le projet qui nous a été confié porte sur la création du jeu du Neutron. Il s'agit de mettre en application nos compétences en algorithmique et programmation afin de créer différents sous-programmes permettant d'assurer le bon fonctionnement du jeu. Ce jeu implique de gérer plusieurs éléments comme le déplacement des pions et du neutron, de déterminer lorsque qu'un joueur a gagné ou a perdu mais aussi de s'assurer que le joueur joue correctement par exemple.

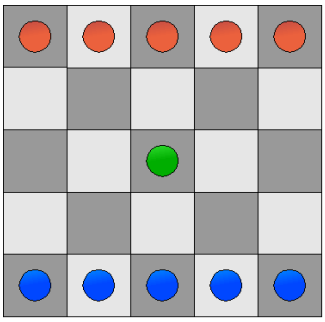
Dans ce rapport, nous établirons tout d'abord une présentation du jeu du Neutron puis nous expliciterons le cahier des charges de ce projet. Ensuite, nous analyserons le sujet et proposerons une méthode afin de mener à bien ce projet. Nous étudierons également les différents sous-programmes créés puis le déroulement et les problèmes rencontrés lors du projet. Enfin, nous présenterons les différentes contraintes et tests mis en places afin d'assurer le bon fonctionnement du jeu.

1. La présentation du jeu

But de jeu : Amener le neutron dans son propre camp.

Matériel : Un damier de 5×5 cases, 5 pions rouges, 5 pions bleus, ainsi qu'un pion vert pour le neutron.

Préparation : Chaque joueur reçoit 5 pions d'une couleur qu'il place sur le damier comme sur la figure ci-dessous. Les pions bleus sont disposés sur la première ligne, les pions rouges sur la dernière ligne et le neutron au centre du plateau.

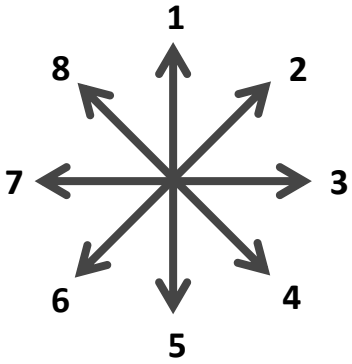


Déroulement de la partie :

- Le joueur bleu commence, il déplace un de ses pions uniquement.
- Puis à tour de rôle, les joueurs doivent déplacer le neutron, puis un de leurs pions.

Déplacement possibles :

Chaque joueur ne peut déplacer ses pions ou le neutron que selon les huit directions suivantes.



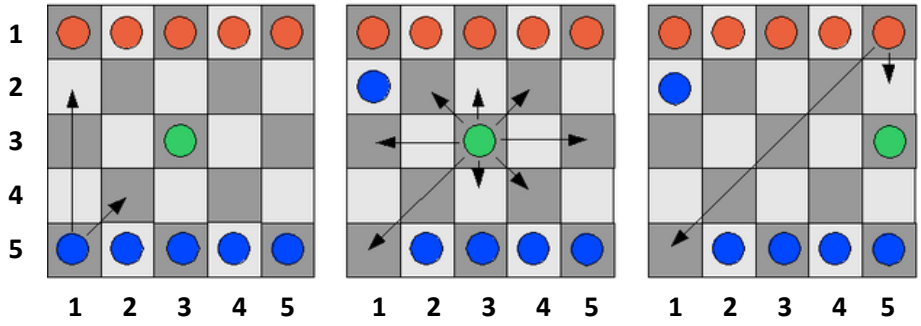
Les pions se déplacent en ligne droite uniquement, c'est-à-dire horizontalement, verticalement ou en diagonale. Un pion ne peut dépasser un autre pion, autrement dit un pion ne peut être déplacé que jusqu'à ce qu'il rencontre un autre pion ou le bord du plateau de jeu.

Lors du premier tour, le joueur bleu commence la partie et déplace uniquement un de ces pions, il ne peut pas déplacer le neutron.

Le gagnant :

Le joueur qui amène le neutron dans le camp adverse a gagné ! Si un joueur ne peut pas déplacer le neutron ou l'un de ses pions, il est bloqué, il a donc perdu la partie. De même, s'il amène le neutron dans le camp adverse.

Exemple de début de partie :



Le joueur bleu commence :

1. Il déplace son pion de coordonnées 1,1 dans la direction 4, son pion a désormais les coordonnées 4,1

Le joueur rouge poursuit :

2. Il déplace tout d'abord le neutron dans la direction 3
3. Puis il choisit de déplacer son pion soit dans la direction 5, soit dans la direction 6

2. Le cahier des charges

Nous avons en charge la réalisation en C de ce jeu. Voici les différents éléments à prendre en compte.

Tout d'abord, notre programme doit permettre à deux joueurs de s'affronter tout en vérifiant la validité du jeu. Nous devrons modéliser le jeu, analyser au mieux le problème afin de déterminer une méthode de résolution mais aussi effectuer la conception avec cette méthode choisie au préalable.

Contraintes :

- Affichage :

Nous devrons afficher le plateau de jeu entre chaque coup joué.

- Déplacements du neutron :

La saisie de déplacement du neutron doit se faire uniquement en indiquant la direction de son déplacement, c'est-à-dire en entrant un chiffre de 1 à 8.

- Déplacement des pions des joueurs :

La saisie des coups des joueurs doit se faire en deux étapes distinctes :

1. Saisie des coordonnées du pion à déplacer selon le format suivant : ligne,colonne
2. Saisie de la direction du déplacement : chiffre entre 1 et 8

- Programmation :

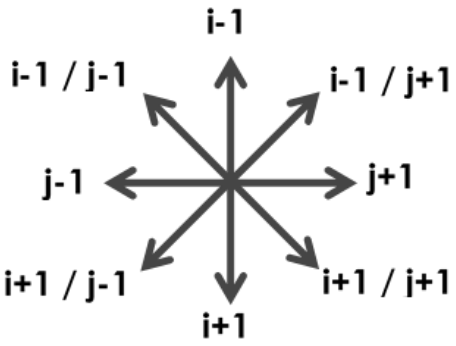
Dans la partie programmation, nous utiliserons des couleurs suivantes pour la visualisation des pions sur le plateau de jeu. Nous nous inspirerons du programme suivant :

```c
#include <stdio.h>

#define reset "\033[0m"
#define reverse "\033[7m"

#define rouge "\033[;31m"
#define vert "\033[;32m"
#define bleu "\033[;34m"

int main () {
        printf ("%s%s %s %s%s %s\n",rouge,reverse,reset,bleu,reverse,reset)
        return 0;
  }
```

3. L'analyse du problème

Le jeu du Neutron suppose qu'il est nécessaire de pouvoir déplacer chacun des pions des joueurs ainsi que le neutron. Pour cela, il est important que l'utilisateur puisse choisir la direction de chaque déplacement, à condition qu'elle appartienne aux huit directions proposées. Afin d'appliquer les déplacements à un pion, nous devons parcourir la matrice choisie pour représenter le plateau de jeu.

Selon chacune des directions, il sera nécessaire parcourir la matrice en incrémentant ou décrémentant les variables de parcours utilisées (ici i et j) comme présenté ci-dessous.



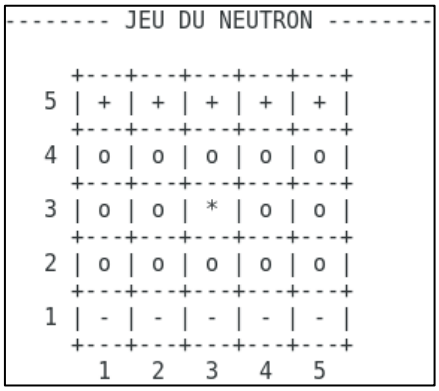
Ceci nous permettra de savoir comment incrémenter les boucles pour pourvoir parcourir selon un déplacement uniquement. Nous déplacerons un pion ou le neutron de cette manière. C'est-à-dire que nous parcourrons dans la direction jusqu'à ce que nous trouvions un obstacle, un pion ou le bord du plateau.

En revanche, il faudra au préalable établir les conversions entre la matrice en mémoire et le plateau de jeu. Nous pouvons remarquer que l'indice de ligne du tableau en mémoire correspond à 6 déduit de l'indice de ligne du plateau de jeu. De plus, il est important que le parcours de la matrice ne franchisse pas limites du plateau réel, il faudra faire attention de ne pas dépasser les indices de bout de parcours de la matrice de taille 5x5.

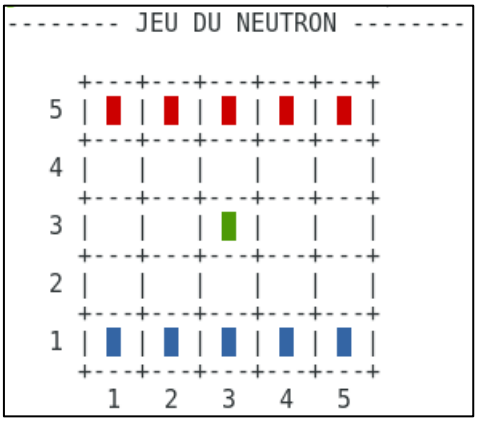
Ensuite, la partie se termine si un joueur perd ou si un joueur gagne, ce qui revient au même finalement car l'un entraîne l'autre. La partie est perdue si tous les pions du joueur concerné sont bloqués, c'est-à-dire qu'il est nécessaire de vérifier que chacun de ses pions sont entourés. Il est également possible de perdre la partie si le joueur adverse bloque le neutron et qu'il n'est plus possible de le déplacer. En revanche, la partie est perdu lorsque le joueur place le neutron dans le camp adverse. Au contraire, on gagne la partie s'il on réussit à placer le neutron dans son propre camp où s'il on oblige l'autre à le faire. Pour cela, il faut vérifier la position du neutron au moment de chaque tour joué. Si c'est le joueur bleu qui joue et qu'il place le neutron dans la ligne 5, il gagne. Par contre, il perd s'il le place dans la ligne 1. Si c'est au tour du joueur Rouge et qu'il place le neutron dans la ligne 1, il gagne. Mais si celui-ci se retrouve à le placer sur la ligne 5, il perd.

4. Description des structures utilisées

Lors de la programmation du jeu du Neutron, nous avons choisi d'utiliser une matrice de caractères de taille 5x5 pour simuler le plateau de jeu. En ce qui concerne les pions, nous avons choisi d'attribuer au joueur rouge les pions '+', au bleu les pions '-' et le pion '*' pour le neutron comme ci-dessus.



L'utilisation d'une matrice pour la programmation du jeu s'explique par le fait qu'une matrice est similaire au plateau de jeu et donc cela nous a permis de mieux visualiser les étapes du déroulement du jeu à réaliser. Ainsi, lors du déroulement du jeu, nous obtenons la matrice suivante.



5. Les sous-programmes

Pour mener à bien ce projet, nous avons été amenés à réaliser différents sous-programmes permettant d'effectuer différents traitements.

5.1. L'initialisation du plateau de jeu

Tout d'abord, il est nécessaire de créer le plateau de jeu du Neutron à l'aide d'une matrice composée de caractères. Pour cela, nous avons créé une fonction « initialisation ».

- **Informations manipulées**

- i et j, entiers, indices de parcours de la matrice M : Données
- M, matrice de taille [5][5] de caractères : Résultat

- **Résolution**

Cette fonction permet de créer une matrice M composée de caractères et de taille 5x5 en la remplissant avec le caractère '+' sur la première ligne (pions rouges), avec le caractère '-' sur la dernière ligne (pions bleus) et en remplissant la case centrale de la matrice avec le caractère '*' (neutron). Les cases des trois lignes centrales de la matrice contiennent le caractère 'o', hormis la case du neutron.

- **Algorithme**

```
Fonction initialisation () : Tableau de caractères
      Locales: i,j : entiers {indices de parcours du tableau}
           M : tableau [i][j] de caractères

    {Parcours de la matrice}
      Pour j de 1 à 5 Faire
        {Mise en place des pions rouges}
        M[1][j]='+'
        {Remplissage des cases vides au centre du plateau}
        Pour i de 2 à 4 Faire
            M[i][j]='o'
        Fait
        {Mise en place des pions bleus}
        M[5][j]='-'
    Fait
    {Mise en place du neutron}
    M[3][3]='*'

    Retourner(M)
FinFonction
```

Ainsi, grâce à ce sous-programme, le plateau de jeu du neutron est initialisé. Il maintenant nécessaire de construire un sous-programme permettant de déplacer les différents pions et le neutron sur le plateau de jeu.

5.2. Le déplacement

Nous avons donc créé une action « deplacement » avec différents paramètres comme la matrice M du plateau de jeu, l'indice de la ligne et celui de la colonne du pion à déplacer ainsi que la direction du déplacement.

- **Informations manipulées**

- i et j, entiers, indices de parcours de la matrice M : Locales
- M, matrice de taille [5][5] de caractères : Donnée/Résultat
- indlig, entier, indice de ligne du pion à déplacer : Données
- indcol, entier, indice de colonne du pion à déplacer : Données
- dir, entier, direction du déplacement : Données

- **Analyse**

Tout d'abord, lorsque l'utilisateur souhaite déplacer un pion, il entre l'indice de la ligne et celui de la colonne du plateau où se situe le pion. Cependant, les indices de lignes du plateau et ceux de notre matrice M sont différents. Il est donc nécessaire d'établir la correspondance des lignes entre les deux.

Une fois cette correspondance établie, le déplacement du pion peut être réalisé selon la direction indiquée. Pour une direction choisie, tant que la case située après celle du pion dans cette direction est vide (contient donc 'o'), on incrémente l'indice de ligne, de colonne ou les deux jusqu'à trouver une case non vide. Ainsi, une fois qu'une case contenant un pion est décelée, on peut déplacer le pion choisi jusqu'à la case précédent la case non vide trouvée.

Par exemple si la direction 1 est choisie, on décrémente la valeur de l'indice de la ligne du pion à déplacer afin de savoir si la case au-dessus contient la valeur 'o'. Ensuite, tant que la valeur de la case au-dessus est 'o', on décrémente l'indice de ligne (sans sortir de la matrice). Ainsi, nous obtenons l'indice ligne de la première case non vide dans la direction 1. Il faut ensuite décrémenter cette valeur et ajouter à cet emplacement le pion à déplacer et remplacer son ancien emplacement par 'o'. En revanche, ce déplacement n'est effectué que si l'indice de ligne du pion est différent de celui trouvé suite au parcours de la matrice.

- **Algorithme**

```
Action deplacement (M,indlig,indcol,dir)
    Locales : i,j : entiers {indices de parcours du tableau}
    Donnée/Résultat : M : tableau [i][j] de caractères
    Données : indlig,indcol : entiers {indices du pion à déplacer}
              dir : entier {nombre correspondant à la direction de
              déplacement)

{Correspondance entre le numéro de ligne du plateau et le numéro de ligne de
notre tableau}

indlig ← 6-indlig

{Parcours et déplacement selon la direction 1}
Si dir=1 Alors
    i ← indlig-1
    Tantque (i>0) et (M[i][indcol]='o') Faire
        i ← i-1
    Fait
    i ← i+1
    {Déplacement du pion}
    Si i≠indlig Alors
        M[i][indcol] ← M[indlig][indcol]
        M[indlig][indcol] ← 'o'
    FinSi
FinSi

{Parcours et déplacement selon la direction 5}
Sinon Si dir=5 Alors
    i ← indlig+1
    Tantque (i<=5) et (M[i][indcol]='o') Faire
        i ← i+1
    Fait
    i ← i-1
      {Déplacement du pion}
    Si i≠indlig Alors
        M[i][indcol] ← M[indlig][indcol]
        M[indlig][indcol] ← 'o'
    FinSi
FinSi

{Parcours et déplacement selon la direction 3}
Sinon Si dir=3 Alors
    j ← indcol+1
    Tantque (j<=5) et (M[indlig][j]='o') Faire
        j ← j+1
    Fait
    j ← j-1

    {Déplacement du pion}
    Si j≠indcol Alors
        M[indlig][j] ← M[indlig][indcol]
        M[indlig][indcol] ← 'o'
    FinSi
FinSi

{Parcours et déplacement selon la direction 7}
Sinon Si dir=7 Alors
```

```
    j ← indcol-1
    Tantque (j>0) et (M[indlig][j]='o') Faire
        j ← j-1
    Fait
    j ← j+1

    {Déplacement du pion}
    Si j≠indcol Alors
        M[indlig][j] ← M[indlig][indcol]
        M[indlig][indcol] ← 'o'
    FinSi
FinSi

{Parcours et déplacement selon la direction 2}
Sinon Si dir=2 Alors
    i ← indlig-1
    j ← indcol+1
    Tantque (i>0) et (j<=5) et (M[i][j]='o') Faire
        i ← i-1
        j ← j+1
    Fait
    i ← i+1
    j ← j-1

      {Déplacement du pion}
    Si (i≠indlig) et (j≠indcol) Alors
        M[i][j] ← M[indlig][indcol]
        M[indlig][indcol] ← 'o'
    FinSi
FinSi

{Parcours et déplacement selon la direction 4}
Sinon Si dir=4 Alors
    i ← indlig+1
    j ← indcol+1
    Tantque (i<=5) et (j<=5) et (M[i][j]='o') Faire
        i ← i+1
        j ← j+1
    Fait
    i ← i-1
    j ← j-1

      {Déplacement du pion}
    Si (i≠indlig) et (j≠indcol) Alors
        M[i][j] ← M[indlig][indcol]
        M[indlig][indcol] ← 'o'
    FinSi

FinSi
{Parcours et déplacement selon la direction 6}
Sinon Si dir=6 Alors
    i ← indlig+1
    j ← indcol-1
    Tantque (i<=5) et (j>0) et (M[i][j]='o') Faire
        i ← i+1
        j ← j-1
    Fait
    i ← i-1
    j ← j+1
```

```
      {Déplacement du pion}
    Si (i≠indlig) et (j≠indcol) Alors
        M[i][j] ← M[indlig][indcol]
        M[indlig][indcol] ← 'o'
    FinSi

FinSi

{Parcours et déplacement selon la direction 8}
Sinon Si dir=8 Alors
    i ← indlig-1
    j ← indcol-1
    Tantque (i>0) et (j>0) et (M[i][j]='o') Faire
        i ← i-1
        j ← j-1
    Fait
    i ← i+1
    j ← j+1

      {Déplacement du pion}
    Si (i≠indlig) et (j≠indcol) Alors
        M[i][j] ← M[indlig][indcol]
        M[indlig][indcol] ← 'o'
    FinSi
FinSi

FinAction
```

5.3. La position du neutron

Lors du déroulement du jeu, chaque joueur est amené à déplacer le neutron en saisissant une direction. Afin de le déplacer, il est nécessaire de connaitre les coordonnées du neutron.

- **Informations manipulées**

- trouve, booléen, vrai si le neutron est trouvé dans la matrice M : Locales
- M, matrice de taille [5][5] de caractères : Donnée
- indlig, entier, indice de ligne du neutron suite au parcours de la matrice M : Résultat
- indcol, entier, indice de colonne du neutron suite au parcours de la matrice M : Résultat

- **Analyse**

Afin de permettre à un joueur de déplacer le neutron sur le plateau, nous avons choisi d'utiliser l'action « deplacement » ci-dessus pour réaliser ce déplacement. Pour utiliser cette action, il est nécessaire de connaitre l'indice de ligne et de colonne du neutron pour ensuite les passer en paramètres dans cette dernière action. Nous avons donc créé une action « position_neutron » afin de déterminer la position du neutron. Pour cela, il est nécessaire de parcourir la matrice M tant que la case contenant la valeur '*' (neutron) n'a pas été trouvée. Le résultat de cette action permet d'obtenir l'indice de ligne et de colonne du neutron.

Pour réaliser cette action, nous avons utilisé une variable « trouve » que nous avons initialisé à faux. Nous parcourons la matrice M tant que la variable « trouve » est à faux et que la case du parcours ne contient pas '*'.

- **Algorithmique**

```
Action position_neutron (M,indlig,indcol)
    Locale : trouve : booléen
    Donnée : M : tableau [i][j] de caractères
    Résultats : indlig,indcol : entiers {indices du neutron}

    trouve ← FAUX
    indlig ← 1

    {parcours de la matrice tant que la valeur * n'est pas trouvée}
    Tantque (indlig<=5) et (trouve=FAUX) Faire
        indcol ← 1
        Tantque (indcol<=5) et (trouve=FAUX) Faire
            Si M[i][j]≠'*' Alors
                indcol ← indcol+1
            Sinon
                trouve ← VRAI
            FinSi
        Fait
        indlig ← indlig+1
    Fait
    indlig ← 6-indlig

FinAction
```

5.4. Les tests de déplacements

La partie du jeu du Neutron peut se terminer lorsque le neutron ou les pions d'un joueur ne peuvent pas être déplacés. Pour cela il faut savoir, pour chacun de ces pions, s'il est entouré de cases vides afin de pouvoir le déplacer. Nous avons donc créé une fonction « test_deplacement ».

- **Informations manipulées**

- i et j, entiers, indices de parcours de la matrice M : Locales
- dir, entier, variable des directions : Locale
- compteur, entier, le nombre de cases vides dans le voisinage du pion : Locale
- M, matrice de taille [5][5] de caractères : Donnée
- indlig, entier, indice de ligne du pion à déplacer : Donnée
- indcol, entier, indice de colonne du pion à déplacer : Donnée

- **Analyse**

Cette fonction « test_deplacement » renvoie un booléen, vrai si le pion peut être déplacé et faux si le pion est bloqué.

Pour un pion donné d'indice « indlig » et « indcol », on teste dans l'ensemble des huit directions (à condition que les cases associées appartiennent à la matrice M) si des cases vides sont situées sur son voisinage. A chaque fois qu'une case vide est détectée, la variable compteur est incrémentée. Ainsi, à la fin des tests sur les huit directions, si la variable compteur est strictement positive alors le pion peut être déplacé et si sa valeur est nulle alors il n'est pas possible de déplacer le pion car aucune case vide n'a été trouvée sur son voisinage.

- **Algorithme**

```
Fonction test_deplacement (M,indlig,indcol) : Booléen
    Locales : i,j : entiers {indices de parcours du tableau}
        dir : entier {nombre correspondant à la direction de déplacement)
         compteur : entier
    Données : M : tableau [i][j] de caractères
        indlig,indcol : entiers {indices du pion à déplacer}

      {Correspondance entre le numéro de ligne du plateau et le numéro de
ligne de notre tableau}

      indlig ← 6-indlig

      compteur ← 0

{Test sur les 8 directions possibles}
Pour dir de 1 à 8 Faire

      Si dir=1 Alors
            i ← indlig-1
            Si (i>0) et (M[i][indcol]='o') Alors
               compteur ← compteur + 1
            FinSi
```

```
            Sinon Si dir=5 Alors
                i ← indlig+1
                Si (i<=5) et (M[i][indcol]='o') Alors
                    compteur ← compteur + 1
                FinSi

            FinSi

            Sinon Si dir=3 Alors
                j ← indcol+1
                Si (j<=5) et (M[indlig][j]='o') Alors
                    compteur ← compteur + 1
                FinSi
            FinSi

            Sinon Si dir=7 Alors
                j ← indcol-1
                Si (j>0) et (M[indlig][j]='o') Alors
                    compteur ← compteur + 1
                FinSi
            FinSi

            Sinon Si dir=2 Alors
                i ← indlig-1
                j ← indcol+1
                Si (i>0) et (j<=5) et (M[i][j]='o') Alors
                    compteur ← compteur + 1
                FinSi
            FinSi

            Sinon Si dir=4 Alors
                i ← indlig+1
                j ← indcol+1
                Si (i<=5) et (j<=5) et (M[i][j]='o') Alors
                    compteur ← compteur + 1
                FinSi

            FinSi

            Sinon Si dir=6 Alors
                i ← indlig+1
                j ← indcol-1
                Si (i<=5) et (j>0) et (M[i][j]='o') Alors
                    compteur ← compteur + 1
                FinSi

            FinSi

            Sinon Si dir=8 Alors
                i ← indlig-1
                j ← indcol-1
                Si (i>0) et (j>0) et (M[i][j]='o') Alors
                    compteur ← compteur + 1
                FinSi
            FinSi
        Fait

        Retourner (compteur>0)
FinFonction
```

5.5. Connaître si un joueur a perdu

Afin de déterminer si un joueur a perdu, il faut connaître si l'ensemble de ses pions sont bloqués et ne peuvent pas être déplacés.

- **Informations manipulées**

- i et j, entiers, indices de parcours de la matrice M : Locales
- entroure, booléen, vrai si le pion est entouré : Locale
- M, matrice de taille [5][5] de caractères : Donnée
- pion, caractère, type de pions recherché dans la matrice M : Donnée

- **Analyse**

Pour savoir si les pions d'un joueur ne peuvent pas être déplacés, nous utilisons la fonction « test_deplacement » dans une nouvelle fonction « perdu ». Cette nouvelle fonction fonctionne grâce à différents paramètres comme la matrice M et le type de pion. En effet, pour que cette fonction fonctionne il faut connaître quel est le type de pion pour lequel on souhaite savoir s'ils sont tous bloqués.

Ainsi, on suppose que l'ensemble des pions du type « pion » sont bloqués et la variable « entoure » est initialisée à vrai. Ensuite, il est nécessaire de parcourir la matrice tant que la variable « entoure » est vraie. Lors de ce parcours, si la case contient la valeur « pion » et que la valeur de la fonction « test_deplacement » est vrai (une case est libre sur le voisinage) alors la valeur de la variable « entoure » devient faux. En effet, si ce pion peut être déplacé alors l'ensemble des pions du type « pion » ne sont pas bloqués et le joueur ayant ces pions n'a donc pas perdu.

- **Algorithme**

```
Fonction perdu (M,pion) : Booléen
    Données : M : tableau [i][j] de caractères
              pion : caractère {type de pion du joueur concerné}
    Locales : i,j : entiers {indices de parcours du tableau}
              entoure : booleen {vrai si le pion ne peut pas être déplacé}

    i ← 1
    entoure ← VRAI

    Tantque (i<=5) et (entoure) Faire
        j ← 1
        Tantque (j<=5) et (entoure) Faire
          Si (M[i][j]=pion) et (test_deplacement(M,6-i,j)=VRAI) Alors
            entoure ← FAUX
          Sinon
            j ← j+1
          FinSi
        Fait
        i ← i+1
    Fait
    Retourner(entoure)
FinFonction
```

5.6. La lecture d'un pion

Afin d'optimiser le programme principal, nous avons créé une action « lecture_pion » qui permet de saisir les coordonnées du pion que l'on souhaite déplacer suivant le format « ligne,colonne ».

- **Informations manipulées**

- indlig, entier, indice de ligne du pion : Résultat
- indcol, entier, indice de colonne du pion : Résultat

- **Analyse**

Cette action permet la lecture des coordonnées du pion et détermine en tant que résultats, les valeurs des variables « indlig » et « indcol » (coordonnées du pion). La lecture de ces deux variables s'effectue tant que les valeurs saisies par l'utilisateur sont correctes, c'est à dire si elles appartiennent à la matrice M (indices compris entre 1 et 5).

- **Algorithme**

```
Action lecture_pion(indlig,indcol)
    Résultats :indlig,indcol : entiers {coordonnées du pion}

    Afficher(Veuillez entrer les coordonnées du pion à déplacer sous la
forme i,j : )
    Lire(indlig,incol)

    {Lecture de l'indice de ligne et de colonne lorsque les valeurs saisies
sont incorrectes}
    Tantque (indlig<=0)ou(indlig<5)ou(indcol<=0)ou(indcol<5)Faire
        Afficher(Les coordonnées ne sont pas valides)
        Afficher(Veuillez entrer les coordonnées du pion à déplacer sous la
forme i,j : )
        Lire(indlig,incol)
    Fait
FinAction
```

5.7. La lecture d'une direction

Nous avons créé une action permettant la lecture d'une direction pour réaliser un déplacement.

- **Information manipulée**

- dir, entier, direction du déplacement : Résultat

- **Analyse**

Afin de saisir le numéro de la direction d'un déplacement d'un pion ou d'un neutron, nous avons créé une action « lecture_direction ». Cette action détermine en tant que résultat, la valeur de variable « dir ».

- **Algorithme**

```
Action lecture_direction(dir)
    Résultat :dir : entier {direction du déplacement}

    Afficher(Veuillez entrer la direction du déplacement : )
    Lire(dir)

    {Lecture du numéro de la direction lorsque la valeur saisie est
incorrecte}
    Tantque (dir<=0)ou(dir>8)Faire
        Afficher(Veuillez entrer une direction valide (entre 1 et 8) pour le
déplacement :)
        Lire(dir)
    Fait
FinAction
```

6. Avancement du projet et problèmes rencontrés

Afin de cerner au mieux les attentes à propos de ce projet, nous avons choisi dans premier temps d'analyser le jeu du Neutron. En effet, nous avons étudié tous les scénarios possibles et toutes les contraintes qu'il est nécessaire de vérifier afin d'assurer son bon déroulement.

Nous avons ensuite décidé de réaliser les algorithmes nécessaires au fonctionnement en pseudo-code puis de les coder en C afin de tester le bon fonctionnement de nos sous-programmes. Cela nous a permis de mettre en évidence plusieurs améliorations que nous avons pu apporter.

L'un des problèmes majeurs auxquels nous avons été confrontés a été de déterminer une démarche permettant de savoir si le neutron ou des pions sont bloqués. Cela nous a demandé un temps important de réflexion afin de réaliser les différents sous-programmes associés.

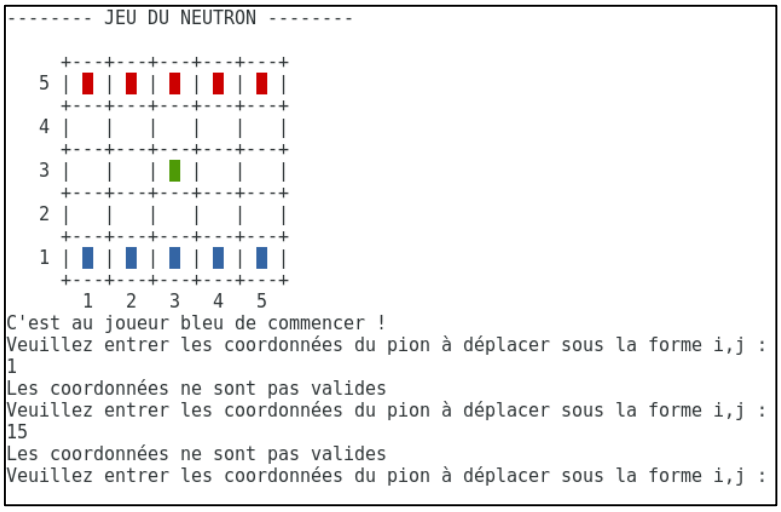
De plus, la conversion des lignes et des colonnes entre le plateau de jeu et notre matrice a nécessité beaucoup d'attention. En effet, il s'agissait de pas mélanger les différentes représentations, entre celle affichée pour le joueur et celle que nous utilisons pour la représentation grâce à notre matrice.

7. Les tests effectués

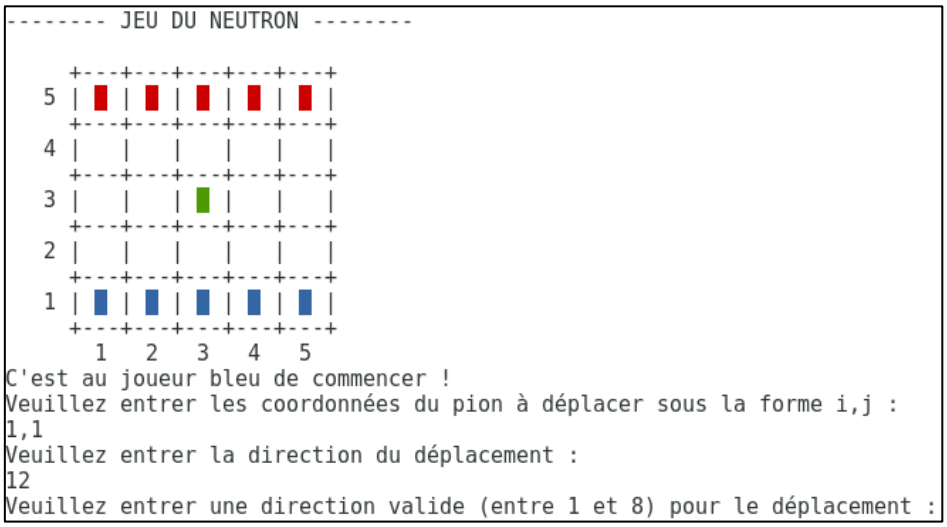
Afin que le jeu du Neutron s'exécute normalement, nous avons ajouté différentes contraintes et réalisé plusieurs tests.

7.1. Les contraintes

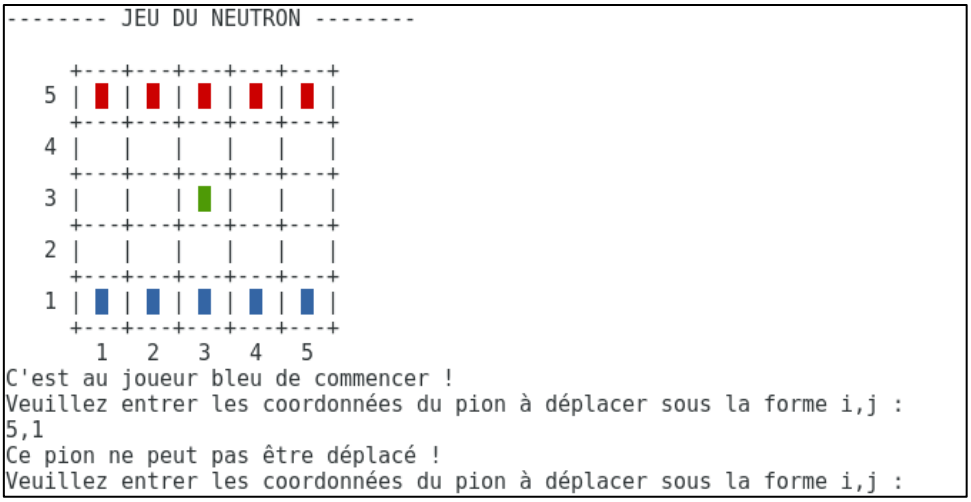
Tout d'abord, lorsque que le joueur entre les coordonnées du pion qu'il souhaite déplacer, il est nécessaire qu'il les renseigne de la manière « ligne,colonne ». Il faut également, que ces deux valeurs appartiennent au plateau de jeu, c'est-à-dire qu'ils appartiennent à l'intervalle [1 ; 5]. Si ces deux conditions ne sont pas réunies, un message d'erreur est affiché à l'utilisateur afin qu'il renseigne des coordonnées valides.



De la même manière, si le joueur saisie une direction incorrecte qui n'appartient pas aux huit directions possibles numérotées de 1 à 8, un message d'erreur est affiché et le joueur est amené à renseigner de nouvelles coordonnées tant que ce qu'il renseigne est incorrect.



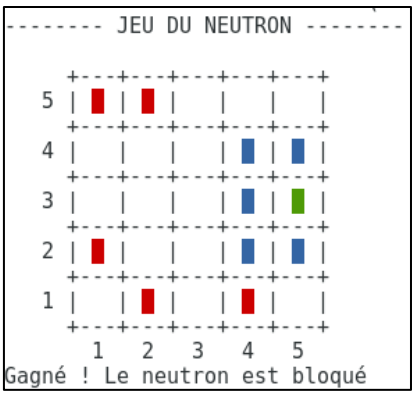
Lors du déroulement du jeu, les joueurs bleu et rouge jouent l'un après l'autre. Il est donc nécessaire de s'assurer que chaque joueur déplace un de ses pions et non celui de l'adversaire ou une case ne contenant aucun pion et ayant pour valeur 'o'.



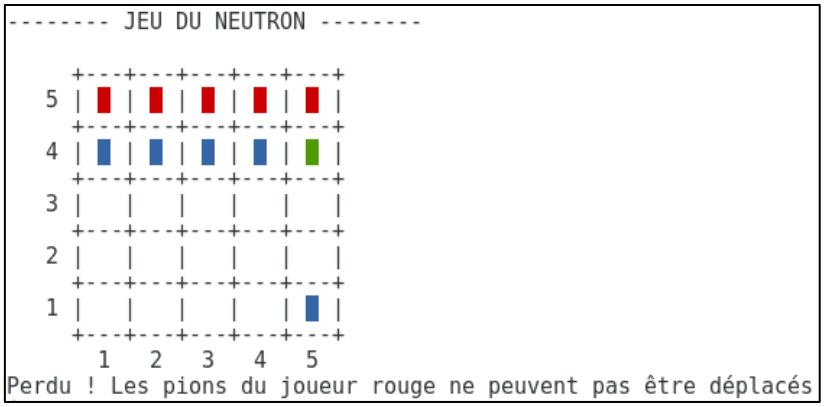
7.2. Les scénarios possibles

Le jeu du Neutron prend fin lorsque le neutron ne peut pas être déplacé, lorsque les pions d'un joueur sont bloqués ou encore lorsque le neutron se trouve dans l'un des camps des deux joueurs.

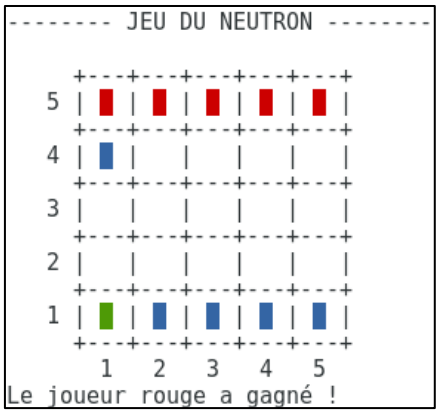
Lorsque le neutron, est entouré de pions comme sur le plateau ci-dessous, alors le joueur rouge ne peut pas le déplacer. Ainsi, le joueur bleu gagne la partie.



Il se peut aussi que tous les pions d'un joueur soient bloqués, comme dans le cas suivant. Dans ce cas, la partie d'arrête et le joueur ayant les pions rouges a perdu.



Il est également possible que le neutron soit déplacé dans l'un des camps des joueurs comme ci-dessous. Dans ce cas, le joueur rouge a gagné puisque le neutron se situe dans le camp bleu.



8. La réalisation C

Une fois l'ensemble des algorithmes créés, nous avons réalisé en C les différents sous-programmes.

```c
// LE JEU DU NEUTRON
# include <stdio.h>
# define NBLIG 5
# define NBCOL 5

# define PION_BLEU '+'  //représentation choisie pour les pions bleus
# define PION_ROUGE '-'//représentation choisie pour les pions rouges
# define NEUTRON '*'//représentation choisie pour le neutron
# define VIDE 'o'//représentation choisie pour les cases vides

# define VRAI 1
# define FAUX 0

# define reset "\033[0m"
# define reverse "\033[7m"

# define rouge "\033[;31m"
# define vert "\033[;32m"
# define bleu "\033[;34m"

void initialisation (char M[][NBCOL]){
    int i,j; //indices de parcours du tableau

    /* Parcours du damier */
    for (j=0;j<NBCOL;j++){
        M[0][j]=PION_BLEU; //mise en place des pions bleus
        for (i=1;i<NBLIG-1;i++){
            M[i][j]=VIDE; //mise en place des cases vides
        }
        M[4][j]=PION_ROUGE; //mise en place des pions rouges
    }
    M[2][2]=NEUTRON; //mise en place du neutron
}

void affichage_plateau_couleurs(char M[][NBCOL]){
    int i,j; //indices de parcours du tableau

    printf("-------- JEU DU NEUTRON --------\n\n");

    /* Parcours du damier */
    for (i=0;i<5;i++){
        printf("     +---+---+---+---+---+\n");
        printf("%4d",5-i);
        for (j=0;j<5;j++){
            if (M[i][j]==PION_BLEU){
                printf(" | %s%s %s",rouge,reverse,reset);
            }
            else if (M[i][j]==PION_ROUGE){
                printf(" | %s%s %s",bleu,reverse,reset);
            }
            else if (M[i][j]==NEUTRON){
                printf(" | %s%s %s",vert,reverse,reset);
            }
            else if (M[i][j]==VIDE){
```

```
                printf(" |  ");
            }
        }
        printf(" |\n");
    }
    printf("     +---+---+---+---+---+\n    ");
    for (j=1;j<=5;j++){
        printf("%4d",j);
    }
    printf("\n");
}

void deplacement (char M[][NBCOL],int indlig,int indcol,int dir){
    int i,j; //indices de parcours du tableau

    /* Conversion de indlig et indcol par rapport à la matrice*/
    indlig=5-indlig;
    indcol=indcol-1;

    /* Quel déplacement ? */

    if (dir==1){
        i=indlig-1;

        /* Parcours des cases dans la direction 1 jusqu'à l'obstacle */
        while ((i>=0) && (M[i][indcol]==VIDE)){
            i--;
        }
        i++; //on récupère l'indice de la case avant le vide
        if (i!=indlig){
            M[i][indcol]=M[indlig][indcol];
            M[indlig][indcol]=VIDE;
        }
    }

 else if (dir==2){
        i=indlig-1;
        j=indcol+1;
        /* Parcours des cases dans la direction 2 jusqu'à l'obstacle */
        while ((i>=0) && (j<5) && (M[i][j]==VIDE)){
            i--;
            j++;
        }
        /*on récupère l'indice de la case avant le vide*/
        i++;
        j--;
        /* déplacement possible ? */
        if ((i!=indlig) && (j!=indcol)){
            M[i][j]=M[indlig][indcol];
            M[indlig][indcol]=VIDE;
        }
    }

    else if (dir==3){
        j=indcol+1;
        /* Parcours des cases dans la direction 3 jusqu'à l'obstacle */
        while ((j<5) && (M[indlig][j]==VIDE)){
            j++;
        }
        j--; //on récupère l'indice de la case avant le vide
```

```
        /* déplacement possible ? */
        if (j!=indcol){
            M[indlig][j]=M[indlig][indcol];
            M[indlig][indcol]=VIDE;
        }
    }

     else if (dir==4){
        i=indlig+1;
        j=indcol+1;
        /* Parcours des cases dans la direction 7 jusqu'à l'obstacle */
        while ((i<5) && (j<5) && (M[i][j]==VIDE)){
            i++;
            j++;
        }
        /*on récupère l'indice de la case avant le vide*/
        i--;
        j--;
        /* déplacement possible ? */
        if ((i!=indlig) && (j!=indcol)){
            M[i][j]=M[indlig][indcol];
            M[indlig][indcol]=VIDE;
        }
    }

     else if (dir==5){
        i=indlig+1;
        /* Parcours des cases dans la direction 5 jusqu'à l'obstacle */
        while ((i<5) && (M[i][indcol]==VIDE)){
            i++;
        }
        i--;//on récupère l'indice de la case avant le vide
        /* déplacement possible ? */
        if ((i!=indlig)){
            M[i][indcol]=M[indlig][indcol];
            M[indlig][indcol]=VIDE;
        }
    }

    else if (dir==6)     {
        i=indlig+1;
        j=indcol-1;
        /* Parcours des cases dans la direction 6 jusqu'à l'obstacle */
        while ((i<5) && (j>=0) && (M[i][j]==VIDE)){
            i++;
            j--;
        }
        /*on récupère l'indice de la case avant le vide*/
        i--;
        j++;
        /* déplacement possible ? */
        if ((i!=indlig) && (j!=indcol)){
            M[i][j]=M[indlig][indcol];
            M[indlig][indcol]=VIDE;
        }
    }


    else if (dir==7){
        j=indcol-1;
```

```
        /* Parcours des cases dans la direction 7 jusqu'à l'obstacle */
        while ((j>=0) && (M[indlig][j]==VIDE)){
            j--;
        }
        j++;//on récupère l'indice de la case avant le vide
        /* déplacement possible ? */
        if (j!=indcol){
            M[indlig][j]=M[indlig][indcol];
            M[indlig][indcol]=VIDE;
        }
    }

    else if (dir==8){
        i=indlig-1;
        j=indcol-1;
        /* Parcours des cases dans la direction 6 jusqu'à l'obstacle */
        while ((i>=0) && (j>=0) && (M[i][j]==VIDE)){
            i--;
            j--;
        }
        /*on récupère l'indice de la case avant le vide*/
        i++;
        j++;
        /* déplacement possible ? */
        if ((i!=indlig) && (j!=indcol)){
            M[i][j]=M[indlig][indcol];
            M[indlig][indcol]=VIDE;
        }
    }
}

void position_neutron (char M[][NBCOL],int* indlig,int* indcol){
    int trouve;

    /* Initialisation */
    trouve=FAUX;
    *indlig=0;

    /* Parcours du damier à la recherche du neutron*/
    while ((*indlig<5)&&(trouve==FAUX)){
        *indcol=0;
        while ((*indcol<5)&&(trouve==FAUX)){
            if (M[*indlig][*indcol]!=NEUTRON){
                (*indcol)++;
            }
            else{
                trouve=VRAI;
            }
        }
        (*indlig)++;
    }

    /* Conversion des coordonées de la matrice */
    (*indlig)=6-(*indlig);
    (*indcol)=(*indcol+1);
}

int test_deplacement (char M[][NBCOL],int indlig,int indcol){
int i,j; //indices de parcours du tableau
int dir; //nombre correspondant à la direction de déplacement
```

```
int compteur; //total de cases voisines vides

    /* Conversion des coordonées */
    indlig=5-indlig;
    indcol=indcol-1;

    /* Initialisation */
    compteur=0;

    /* Parcours des cases voisines */
    for (dir=1;dir<=8;dir++){
        if (dir==1){
            i=indlig-1; //coordonnées de la case voisine dans la direction
1
            /* case voisine innoccupée ? */
            if ((i>=0)&&(M[i][indcol]==VIDE)){
                compteur++;
            }
        }
        else if (dir==2){
            /* coordonnées de la case voisine dans la direction 2 */
            i=indlig-1;
            j=indcol+1;
            /* case voisine innoccupée ? */
            if ((i>=0)&&(j<5)&&(M[i][j]==VIDE)){
                compteur++;
            }
        }
        else if (dir==3){
            j=indcol+1; //coordonnées de la case voisine dans la direction
3
            /* case voisine innoccupée ? */
            if ((j<5)&&(M[indlig][j]==VIDE))
            {
                compteur++;
            }
        }
        else if (dir==4){
            /* coordonnées de la case voisine dans la direction 4 */
            i=indlig+1;
            j=indcol+1;
            /* case voisine innoccupée ? */
            if ((i<5)&&(j<5)&&(M[i][j]==VIDE)){
                compteur++;
            }
        }
        else if (dir==5){
            i=indlig+1; //coordonnées de la case voisine dans la direction
5
            /* case voisine innoccupée ? */
            if ((i<5)&&(M[i][indcol]==VIDE)){
                compteur++;
            }
        }

        else if (dir==6){
            /* coordonnées de la case voisine dans la direction 6 */
            i=indlig+1;
            j=indcol-1;
            /* case voisine innoccupée ? */
```

```
            if ((i<5)&&(j>=0)&&(M[i][j]==VIDE)){
                compteur++;
            }
        }

        else if (dir==7){
            j=indcol-1; //coordonnées de la case voisine dans la direction
7
            /* case voisine innoccupée ? */
            if ((j>=0)&&(M[indlig][j]==VIDE)){
                compteur++;
            }
        }

        else if (dir==8){
            /* coordonnées de la case voisine dans la direction 8 */
            i=indlig-1;
            j=indcol-1;
            /* case voisine innoccupée ? */
            if ((i>=0)&&(j>=0)&&(M[i][j]==VIDE)){
                compteur++;
            }
        }
    }
    return(compteur>0); //si nombre de voisins>0 déplacement possible
}

int perdu(char M[][NBCOL], char pion){
    int i,j; // indices de parcours
    int entoure; // indique si le pion est entouré

    /* Initialisation */
    i=0;
    entoure=VRAI;

    /* Parcours du damier */
    while((i<5)&&(entoure)){
        j=0;
        while((j<5)&&(entoure)){
            /* Pion entouré ? */
            if((M[i][j]==pion)&&(test_deplacement(M, (5-i), (j+1))==VRAI)){
// si le pion n'est pas entouré et si le pions correspond au joueur
concerné
                entoure=FAUX;
            }
            else{
                j++;
            }
        }
        i++;
    }
   return(entoure);
}

void lecture_pion(int *indlig, int *indcol){

    printf("Veuillez entrer les coordonnées du pion à déplacer sous la
forme i,j :\n");
    scanf("%d,%d",indlig, indcol);
```

```
    /*lorsque les coordonnées du pion indlig, indcol sont incorrectes*/
    while((*indlig<=0)||(*indlig>5)||(*indcol<=0)||(*indcol>5)){
        printf("Les coordonnées ne sont pas valides\n");
        printf("Veuillez entrer les coordonnées du pion à déplacer sous la
forme i,j :\n");
        scanf("%d,%d",indlig, indcol);
    }
}

void lecture_direction(int *dir){
    printf("Veuillez entrer la direction du déplacement :\n");
    scanf("%d",dir);

    /*lorsque le numéro de la direction est incorrecte*/
    while((*dir<1)||(*dir>8)){
        printf("Veuillez entrer une direction valide (entre 1 et 8) pour le
déplacement :\n");
        scanf("%d",dir);
    }
}

 int main (){
    char M[NBLIG][NBCOL]; // tablau représentant le plateau de jeu
    int indlig,indcol; // indices de ligne et de colonne
    int dir; // chiffre correspondant à la direction
    int i,j;//indices de position du neutron
    int tour;//numéro du tour au fil du jeu

    /* Initialisation */
    initialisation(M);
    affichage_plateau_couleurs(M);

    /*Début du jeu : les bleus déplacent leur pion*/
    tour =0;
    printf("C'est au joueur bleu de commencer !\n");
    lecture_pion(&indlig,&indcol);

    while ((M[5-indlig][indcol-1]==PION_BLEU)||(M[5-indlig][indcol-
1]==VIDE)){
        printf("Ce pion ne peut pas être déplacé !\n");
        lecture_pion(&indlig,&indcol);
    }

    lecture_direction(&dir);
    deplacement(M,1,indcol,dir);
    affichage_plateau_couleurs(M);
    tour++;

    /* Poursuite du jeu, avec alternativement les rouges et les bleus qui
jouent
    et ceci tant que le neutron n'est pas dans un des camps, que les pions
des joueurs ne sont pas bloqués et que le neutron n'est pas bloqué*/
    while
((i!=1)&&((i!=5)&&(perdu(M,PION_BLEU)==FAUX)&&(perdu(M,PION_ROUGE)==FAUX)&&
(perdu(M,NEUTRON)==FAUX))){
        /* Les ROUGES jouent */
        if (tour%2 != 0){
            printf("C'est au joueur rouge de jouer !\n");

            /*Déplacement du Neutron*/
```

```
            position_neutron(M,&i,&j);
            lecture_direction(&dir);
            deplacement (M,i,j,dir);
            affichage plateau couleurs(M);
            position_neutron(M, &i, &j);

            if (i==5){//Le neutron est dans son camp
                printf("Le joueur rouge a gagné !\n");
            }
            else if(i==1){//Le joueur rouge a mis le neutron dans le camp
bleu
                printf("Le joueur bleu a gagné !\n");
            }
            else{
                /*Déplacement du Pion*/
                lecture_pion(&indlig,&indcol);

                while ((M[5-indlig][indcol-1]==PION_ROUGE)||(M[5-
indlig][indcol-1]==VIDE)){
                    printf("Ce pion ne peut pas être déplacé !\n");
                    lecture_pion(&indlig,&indcol);
                }
                lecture_direction(&dir);
                deplacement (M,indlig,indcol,dir);
                affichage_plateau_couleurs(M);
            }
        }

        /* Les BLEUS jouent */
        else{
            if (tour%2 == 0){
                printf("C'est au joueur bleu de jouer !\n");

                /* Déplacement du Neutron */
                position_neutron(M,&i,&j);
                lecture_direction(&dir);
                deplacement (M,i,j,dir);
                affichage_plateau_couleurs(M);
                position_neutron(M, &i, &j);

                if (i==1){////Le joueur bleu a mis le neutron dans son camp
                    printf("Le joueur bleu a gagné ! \n");
                }
                else if(i==5){//Le joueur bleu a mis le neutron dans le
camp rouge
                    printf("Le joueur rouge a gagné !\n");
                }
                else{
                    /* Déplacement du Pion */
                    lecture_pion(&indlig,&indcol);

                    while ((M[5-indlig][indcol-1]==PION_BLEU)||(M[5-
indlig][indcol-1]==VIDE)){
                        printf("Ce pion ne peut pas être déplacé !\n");
                        lecture_pion(&indlig,&indcol);
                    }

                    lecture_direction(&dir);
                    deplacement (M,indlig,indcol,dir);
                    affichage_plateau_couleurs(M);
```

```
                }
            }
        }
        tour++;
    }

    /* Test sur la fin de partie : neutron bloqué, pions bloqués */
    if (perdu(M,NEUTRON)){
        printf("Gagné ! Le neutron est bloqué\n");
    }
    else if (perdu(M,PION_ROUGE)){
            printf("Perdu ! Les pions du joueur bleu ne peuvent pas être
déplacés\n");
    }
    else if(perdu(M,PION_BLEU)){
            printf("Perdu ! Les pions du joueur rouge ne peuvent pas être
déplacés\n");
    }
return 0;
}
```

Conclusion

Ce projet est une expérience enrichissante car il nous a permis de mettre en œuvre l'ensemble des compétences acquises durant ce semestre en algorithmique et en programmation. Nous avons aussi pu améliorer nos savoirs et nos compétences dans la création de sous-programmes afin de les rendre lisibles et compréhensibles par tous à l'aide de commentaires.

Par ailleurs, ce projet nous a permis de de nous rendre compte qu'il est important de prendre le temps de bien analyser le sujet puis d'écrire les algorithmes et qu'il ne faut pas se lancer sans réfléchir dans la programmation en C. Cela nous a apporté un véritable gain en efficacité lors de la réalisation de ce projet.

Il nous a également permis de nous mettre à la place de l'utilisateur et d'anticiper les différents scénarios possible en jouant au jeu du Neutron.