Challenge moteur de jeux

AVALAM

Compte rendu livrable 1

Groupe : La Valeur

Participants :

- Debroucker Baptiste
- OUAZZANI CHAHDI Oualid
- Desvigne Arthur
- Vorobieff--Nawrot Clément

Introduction

Ce compte-rendu a pour objectif premier de présenter, évaluer et tester l'évolution du développement des deux programmes standalone.exe et diag.exe. En effet, dans le projet challenge-moteur de jeu, le but premier de ce livrable est de rendre un programme qui permette l'exécution d'un jeu fonctionnel avec un affichage effectuée par une page HTML, et d'un programme de diagramme qui permette l'affichage de diagrammes spécifiques à l'aide d'une page HTML dédiée à cette précise mission.

À côté de ce compte-rendu, vous aurez notamment l'occasion de retrouver tout notre projet, dont l'organisation est restée inchangée par rapport au répertoire fourni aux différents groupes sur Moodle. Ce compte-rendu sera divisé selon l'organisation suivante : tout d'abord un paragraphe portant sur le début du projet, puis tout un pan sur le développement du fichier standalone.exe, ainsi qu'un pan sur le développement du fichier diag.exe pour terminer sur une conclusion générale du projet et du livrable 1.

Début du projet :

En date du 8 février 2024 nous avons agencé l'emploi du temps de projet et avons distribué les différentes tâches de travail afin de garantir un bon avancement du projet. Nous avons étudié le fonctionnement du jeu via un premier test ; premier test réalisé via le fichier avalam-standalone.html. Un deuxième test a ensuite été réalisé via un exemplaire de la plateforme de jeu Avalam. Tous les participants se sont essayés au jeu afin de se familiariser avec le fonctionnement de celui-ci. Nous avons eu l'occasion de réaliser un tournoi entre les différents membres du groupe afin que tout le monde se familiarise bien avec le jeu, et essayer de déterminer quelles étaient les fins possibles d'une partie de Avalam Evolution. Cela fait, nous avons pu nous pencher sur un début d'idée pour un algorithme pour enfin se pencher sur le développement en tant que tel. La partie programmation a pu ainsi débuter.

Développement du fichier standalone.exe : début 14/02/2024

Notre équipe a débuté par le développement du fichier standalone.exe. Nous avons entamé la programmation par une phase d'étude de l'exécutable mis a disposition afin de comprendre le fonctionnement de celui-ci. Nous avons dans un premier abord rencontré des difficultés de compréhension de l'objectif précis de cet exécutable. Après réflexion nous avons pu débuter la programmation de celui-ci.

Description technique :

Nous avons débuté la phase de programmation du fichier standalone.c, dans la fonction main, par la déclaration des variables et de la fonction writeJS, bien sûr après la vérification de la déclaration de toutes les #include nécessaires, puis les initialisations des fonctions (à utiliser dans ce qui suit) déclarées dans les librairies.

A l'aide de la fonction writeJS () (codée en bas) qui est responsable de l'écriture sous forme des données pour la visualisation graphique, on initialise d'abord l'emplacement du fichier JSON qui sera modifié au cours de chaque appel de la fonction :

```
writeJS(position, score, "./web/refresh-data.json");
```

A la déclaration du début de la partie, le code donne la possibilité d'interacteur avec le jeu aux deux joueurs en leur demandant préciser le placement de leurs bonus et malus en respectant des critères logiques représentées sous forme des boucles while (la couleur choisie doit correspondre à la couleur du joueur, un pion ne peut pas être un malus et un bonus en même temps). Voici par exemple une boucle while que nous utilisons pour vérifier la position de là où le joueur veut placer son bonus/malus :

```
printf("\n\tAu tour du joueur 1 de donner la position de son bonus :");
    scanf("%hhd", &position.evolution.bonusJ);            //entering the
position of the bonus of the player 1

    while(position.cols[position.evolution.bonusJ].couleur==ROU)
    {
        printf("\n\tImpossible de placer le bonus sur une case occupée par un
malus ou d'une autre couleur, veuillez choisir une autre case:");
        scanf("%hhd", &position.evolution.bonusJ);
    }
    writeJS(position, score, "./web/refresh-data.json");
```

Apres validation des bonus et malus, en passant au coup du premier jour, à l'aide d'une grande boucle while qui représente la totalité du jeu *(*le critère d'arrêt étant le fait qu'il n'y ai plus aucun coup légal jouable), le premier joueur entre sa case de départ puis celle d'arrive, qui ne peuvent pas être une chaine de caractères ou un nombre non existant, si le code détecte la validité du coup, le score évalue et on passe au tour du deuxième joueur et ainsi de suite.

Pour la récupération des cases d'arrivées et de départ, nous nous sommes aperçus que lorsqu'un utilisateur se trompe et qu'il entre une chaine de caractères, le programme plantait en boucle. Ainsi, il a été nécessaire de développer une robustesse à ce problème à l'aide d'un do while. Le code rafraichie les coups légaux dans le fichier JSON à la fin de chaque coup. Vous pouvez retrouver un exemple d'un do while ci-dessous :

```
     do
       {
           printf("\n\tEntrez la case de départ : "); // asking for the case
of departure
           result = scanf("%hhd", &coup.origine); // asking for the case of
ending


           if (result != 1)        // if scanf didn't succeed to read a
number, empty the input buffer
           {
               printf("Entrée invalide. Veuillez entrer un nombre.\n");
               while (getchar() != '\n'); // emptying the input buffer
           }

       }
```

```
        while (result != 1); // repeating until the input is a number
```

Pour finir le jeu, en sortant du boucle while des coups, le code détermine la fin du jeu et annonce le gagnant en calculant le score de chaque joueur selon les critères du jeu. Pour cela, nous utilisons un assemblage de structure if qui affiche la fin de la partie :

```
if(score.nbJ > score.nbR)
//if the score of the yellow is superior to the score of the red
    {
        printf("\n\t!--!--!--!--!--! LE JOUEUR JAUNE GAGNE !! !--!--!--!--!--
!\n");         //showing winning of the yellow
    }
    else if(score.nbJ < score.nbR)
//if the score of the red is superior to the score of the yellow
    {
        printf("\n\t!--!--!--!--!--! LE JOUEUR ROUGE GAGNE !! !--!--!--!--!--
!\n");         //showing winning of the red
    }
    else
//if scores are equals
    {
        if(score.nbJ5 > score.nbR5)
            printf("\n\t!--!--!--!--! LE JOUEUR JAUNE GAGNE GRACE A SES PILES
DE 5 !--!--!--!--!");     //we check the piles of 5

        if(score.nbJ5 < score.nbR5)
            printf("\n\t!--!--!--!--! LE JOUEUR ROUGE GAGNE GRACE A SES PILES
DE 5 !--!--!--!--!");

        else
            printf("\n\t!--!--!--!--! ÉGALITÉ !--!--!--!--!");
//showing equality
    }
```

Commentaires :

- Nous avons constaté que des fois, la page HTML avalam.refresh n'arrive pas à suivre les modifications du programme. Ce qui en résulte est un bug dans l'affichage, les objets paraissent figés. Pour régler ce problème, il suffit de reload la page en faisant Ctrl + R par exemple.

- Le Makefile … ????

Jeu d’essais :

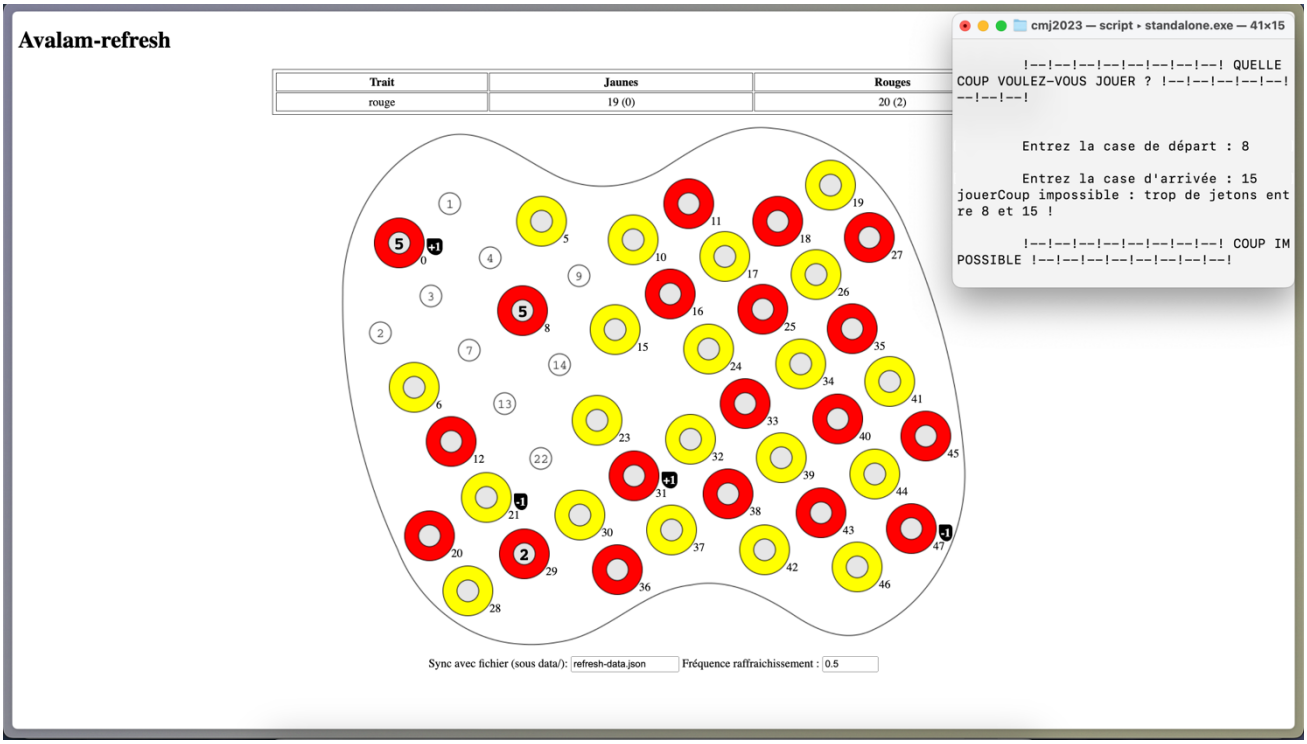
Pour tester le programme, nous allons jouer une partie en testant les limites du programme. Pour cela, tous les positions et actions jouées pourront être retrouvées dans la trace écrite qui correspond à un miroir de la console au moment de l’exécution du code. Vous pourrez retrouver ce fichier sous le nom : ‘jeu_essais_standalone.txt’. Pour accompagner cela, vous pourrait retrouver ici des captures d’écrans des différentes avancées de la partie.

Première partie :

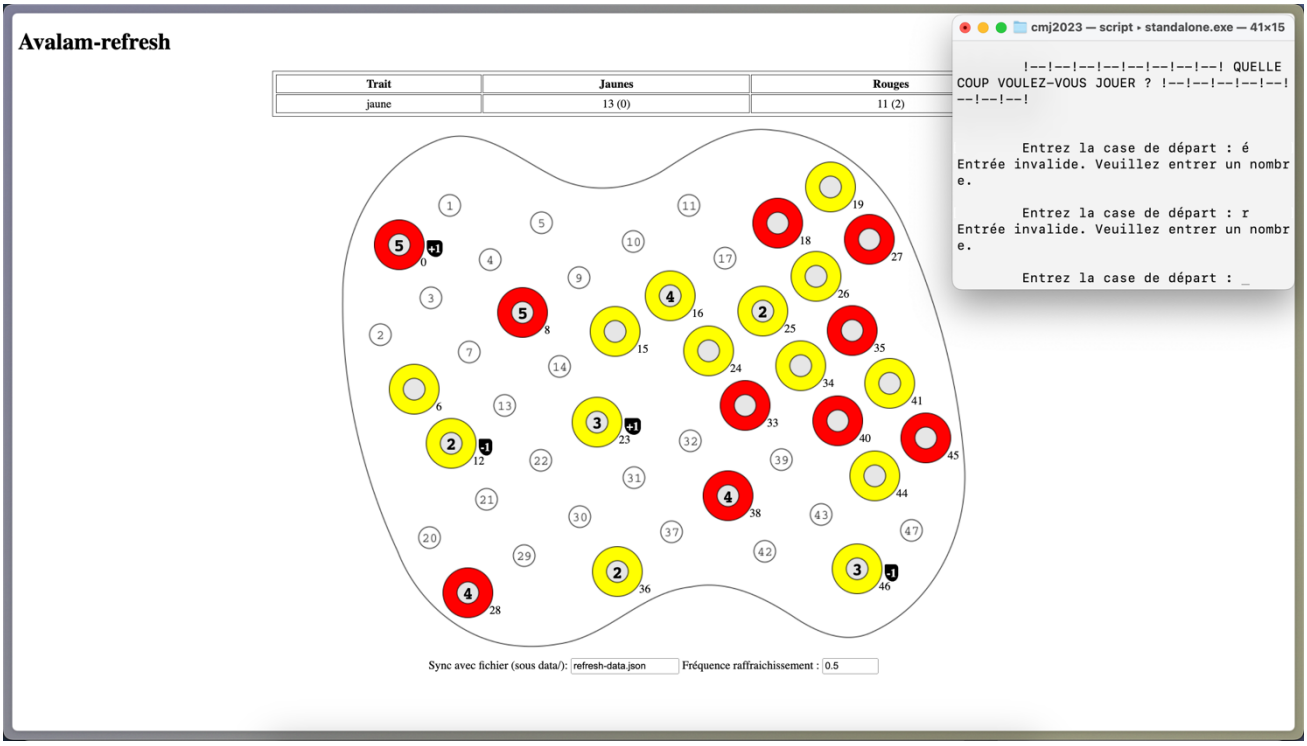
Pour cette première partie, nous choisissons de positionner nos bonus et nos malus selon cette description :

Ici, on test bien les déplacements impossibles (le fait d’aller plus loin que possible). De ce que l’on constate ci-dessous, cela fonctionne également.

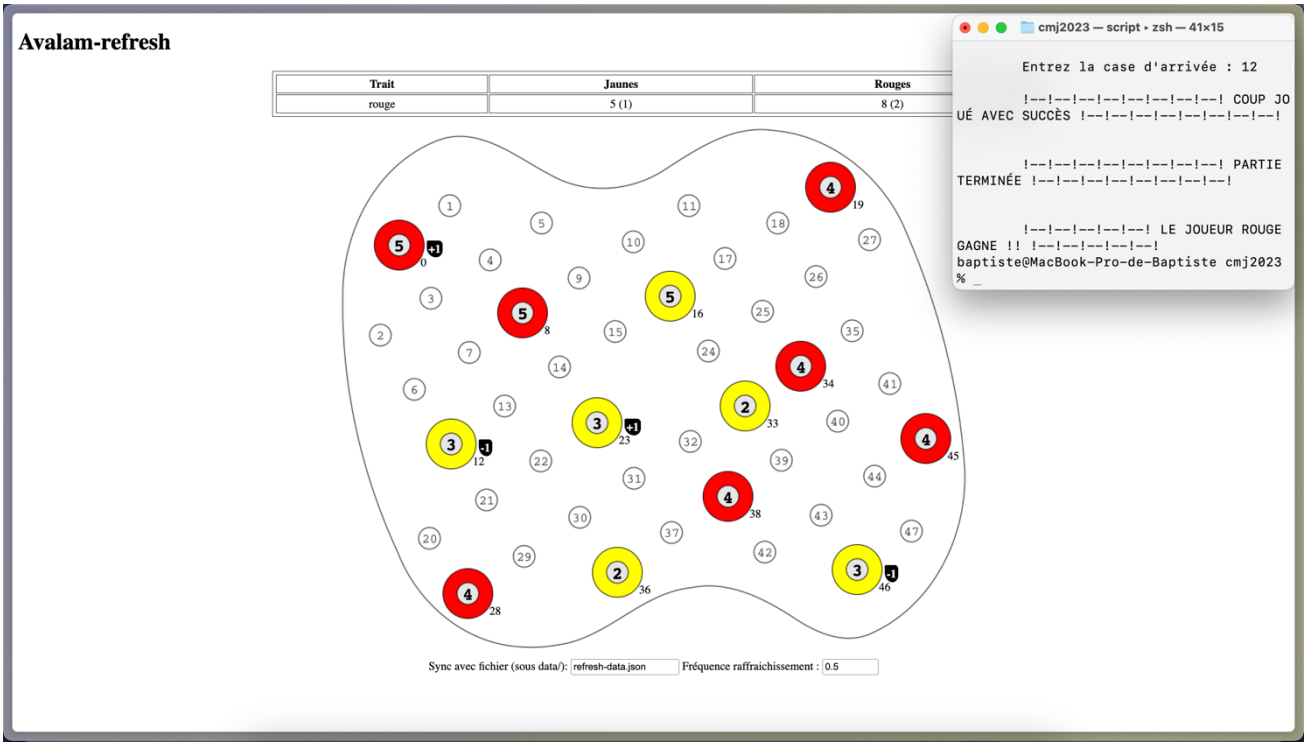
Dans cette figure-la, nous testons si un pion peut se rajouter sur une tour déjà remplie de 5 pions. Comme on le constate ci-dessous, cela fonctionne bien !



Maintenons, avançons dans la partie et testons si le programme est robuste à une entrée qui n'est pas un chiffre et une position valable. Comme on le constate ci-dessous, c'est un succès. Cela est réalisé comme dit plus tot aux do while.



Testons maintenant la fin de partie avec la position suivante. Nous constatons bien que le gagnant est le bon (dans ce cas ici ce sont les rouges qui gagnent).



Développement du fichier diag.exe : début 15/02/2024

Notre équipe a ensuite poursuivi son travail de programmation par le développement du fichier diag.exe. Nous avons abordé la programmation de cet exécutable par une phase d'analyse poussée du fonctionnement de cet exécutable afin de comprendre le réel objectif et fonctionnement de celui-ci. Nous avons dans un premier abord rencontré des difficultés de modélisation et de compréhension de son fonctionnement et avons pu alors enclencher la phase de programmation de celui-ci.

Description technique :

Selon les cahiers des charges, notre programme diag.exe doit pouvoir entrer une chaine FEN (notation permettant de représenter n'importe quelle position d'un plateau du jeu Avalam Evolution) et créer un fichier JSON qui sera lu par une page HTML. Ainsi, notre idée de départ était de reprendre le standalone.exe que nous avons tout d'abord développé pour lui ajouter des fonctions supplémentaires pour que le programme puisse lire un numéro de diagramme et de le représenter sur une page Web. Cependant, nous avons constaté après avoir développée une première version de ce diag.exe que le programme était trop limité et que son développement allait être trop alambiqué. Nous avons donc pris la décision de repartir sur une base neutre et nouvelle pour développer la fonction.

Ainsi, nous définissons les différents includes au début du programme avant le main. En parlant du main, il prendra en paramètre le numéro du diagramme (commençant par 1), et la chaine FEN qui représente le diagramme. Après cela, nous vérifions tout d'abord le nombre d'arguments passés à la fonction et sinon, on rappelle la syntaxe à utiliser pour passer les arguments à la fonction. Le code suivant le démontre :

```
    if(argc != 3)   //Condition d'erreur pour le nombre d'arguments passé en
ligne de commandes
    {
        fprintf(stderr, "!--!--!--!--!--! ERREUR CRITIQUE : NOMBRE
D'ARGUMENTS TROP IMPORTANT. SYNTAXE AUTORISÉE: EXE NUMDIAG FEN !--!--!--!--!-
-!\n");
        exit(EXIT_FAILURE);
    }
```

Dès lors, nous pouvons commencer en prélude à légèrement traiter la chaine FEN. Pour des raisons pratiques, nous commençons par récupérer le trait de la chaine FEN. Pour cela, il suffit de récupérer la longueur de la chaine, de décrémenter de 1 cette longueur pour avoir le trait. Avec un if, nous comparons si le trait est 'r' ou 'j', pour qu'il indique ainsi au programme le trait. Si le trait n'est pas renseigné (ce qui est préjudiciable), nous stoppons net le programme. Voici le code le mettant en place :

```
longueur=strlen(argv[2]);
    if(argv[2][longueur-1]!='r' && argv[2][longueur-1]!='j')
    {
        printf("!--!--!--!--!--! ERREUR CRITIQUE : LE TRAIT N'EST PAS
RENSEIGNÉ !--!--!--!--!--!\n");
        exit(EXIT_FAILURE);
    }
    if(argv[2][longueur-1]=='r')
    {
        position.trait=ROU;
    }
    else
    {
        position.trait=JAU;
    }
```

Maintenant, nous demandons à l'utilisateur de rentrer la description. Si l'utilisateur n'a pas entré de description, on l'informe via la console qu'il n'a fourni de description. Nous implémentons aussi une option d'entrer la description à l'aide d'une redirection. Pour cela, nous créons un fichier 'description' qui est dans le même répertoire à qui l'on pourra donner la description à l'aide d'une redirection. Dans le programme diag.exe, nous lisons le contenu du fichier et nous l'écrivons dans le fichier qui sera donné à la page HTML selon le code suivant :

```
    FILE *ficdesc;


    ficdesc=fopen("description.txt","r");
    fread(description,1,MAXCHAR,ficdesc);
    fclose(ficdesc);

    printf("!--!--!--!--!--! VEUILLEZ ENTREZ UNE DESCRIPTION POUR LE
DIAGRAMME !--!--!--!--!--!\n");
```

```
    fgets(desc, MAXCHAR, stdin); //on demande à l'utilisateur de rentrers la
description du fichier
    strtok(desc, "\n"); //on supprime le retour à la ligne du au fgets

    if(strcmp(desc,"")==0)
    {
        printf("!--!--!--!--!--! DESCRIPTION NON RENSEIGNÉE !--!--!--!--!--
!\n");
    }
```

Après cela, nous renseignons les caractéristiques globales du diagramme comme on peut le voir avec le code ci-dessous. Nous avons repris le style d'affichage du standalone.exe pour garder une certaine cohérence à travers le projet et le livrable.

```
    printf("!--!--!--!--!--! CARACTÉRISTIQUES DU FICHIER !--!--!--!--!--
!\n");
    printf("!--! DIAGRAMME NUMÉRO : %s !--!\n", argv[1]);
    printf("!--! CHAINE FEN : %s !--!\n",chaineF);
    if(strcmp(desc,"")==0)
        printf("!--! DESCRIPTION : %s !--!\n",description);
    else
        printf("!--! DESCRIPTION : %s !--!\n",desc);
    printf("!--!--!--!--!--! FIN DES CARACTÉRISTIQUES !--!--!--!--!--!--
!\n");
```

Par la suite, comme demandé dans le cahier des charges, nous demandons ici à l'utilisateur s'il veut modifier le chemin d'écriture du fichier, sinon le fichier JSON à produire prendra un nom par défaut : 'diag_ressources.js'.

```
    printf("!--!--!--!--!--! VOULEZ-VOUS CHANGER LE CHEMIN D'ACCES DU FICHIER
??? 1 POUR OUI, 0 POUR NON !--!--!--!--!--!\n");
    scanf("%d",&rep);

    if(rep == 1)
    {
        printf("!--!--!--!--!--! VEUILLEZ RENSEIGNER LE NOUVEAU CHEMIN
D'ACCES !--!--!--!--!--!\n");
        while ((c = getchar()) != '\n' && c != EOF) { }
        fgets(chemin, MAXNOM, stdin);
        strtok(chemin, "\n");
    }
```

Nous commençons à écrire donc dans le fichier JSON pour qu'il soit lu par la page HTML. C'est ici que nous allons interpréter la chaine FEN. Cette partie du code a dû être réécrite trois fois à cause d'un problème d'interprétation des chiffres dans la chaine FEN (chiffres correspondant aux nombres de trous dans le plateau Avalam). En effet, nous avions adopté une approche de coder une fonction propre qui réalise l'interprétation de la chaine FEN à part. Cependant, après de multiples tentatives (à retrouver dans les anciens commit du github), nous avons donc décider de l'interpréter à l'écriture du fichier. Ainsi, nous utilisons un while qui prend en condition l'arrêt un indice i qui s'arrête jusqu'au nombre de cases maximum du plateau. Nous décidons de prendre une variable stock qui sera utilisée pour tester s'il y a un trou dans la chaine FEN. Stock sera positif lorsqu'il y aura un trou (soit un nombre dans la chaine FEN). Ainsi, on pourra vérifier s'il est positif. Si c'est le cas, on écrit dans le fichier JSON un nombre de pièces sur une case nulle et une couleur nulle également. Voici le code qui réalise ce que l'on vient de présenter :

```
if(stock>0)
        {
            if(i!=NBCASES && chaineF[j]!='B' && chaineF[j]!='b' &&
chaineF[j]!='M' && chaineF[j]!='m')
                fprintf(fic,"\t{%s:%d, %s:%d},\n",STR_NB,
position.cols[i].nb=0,STR_COULEUR, position.cols[i].couleur=0);
            else if (chaineF[j]!='B' && chaineF[j]!='b' && chaineF[j]!='M' &&
chaineF[j]!='m')
                fprintf(fic,"\t{%s:%d, %s:%d}\n",STR_NB,
position.cols[i].nb=0,STR_COULEUR, position.cols[i].couleur=0);

            stock--;
        }
```

Ici, nous affectons à stock la valeur du nombre dans la chaine FEN pour qu'il soit positif. On affecte à une variable 'stockage' la valeur de stock qui nous servira plus tard pour positionner les bonus et les malus.

```
        else if (chaineF[j]>='0' && chaineF[j]<='9')
        {
            if(chaineF[j+1]>='0' && chaineF[j+1]<='9')
            {
                stock = (chaineF[j+1]-'0')+10*(chaineF[j]-'0');
```

```
            j++;
        }
        else
        {
            stock = (chaineF[j]-'0');
        }

        stockage=stock;
    }
```

Dans la suite du code, si l'indice rencontre une lettre, on utilise un switch pour déterminer de quel cas il s'agit. En fin de l'écriture, nous écrivons la position du plateau à condition qu'il ne détecte aucune lettre de bonus ou malus. Cela permet de simplifier l'affichage. A la fin de l'écriture du fichier JSON, nous écrivons l'emplacement des bonus et des malus à la fin par nécessité dont induit la façon dont nous avons développé le diag.exe. Nous distinguons ici un problème étrange. En effet, lors des tests que nous avons réalisés pour voir si le programme correspondait à nos attentes, nous nous sommes aperçus que les bonus et malus des jaunes n'étaient pas affectés correctement, alors qu'ils ont été développés de la même façon que ceux des rouges, les rouges fonctionnants parfaitement. En complément du programme, nous avons développé une fonction qui permet de transformer une position en chaine FEN, cas qui pourrait être utile si l'on veut récupérer une position intéressante et la partager.

Jeu d'essais :

Dans cette partie, nous testons le bon fonctionnement de notre programme. Pour cela, nous utiliserons l'exemple de chaine FEN suivant : t2dMtb10UBDm10U20 r avec le numéro de diagramme 1. Vous pourrait retrouver le jeu d'essais dans le fichier nommé 'jeu_essais_diag.txt'.

Conclusion :

En définitive, le délai de livraison du livrable a pu être respecté grâce à une planification des taches d'études et de programmations efficaces permettant l'obtention d'un résultat fonctionnel et utilisable dans de bonnes conditions. Le groupe a rencontré des difficultés de compréhension des programmes rapidement corrigés par une phase d'étude conséquente ainsi qu'une excellente communication interne (entre membres) mais également externe grâce à une sollicitation profitable du référent de projet.