BERESZYNSKI Ewen, PAZDZIEJ Dorian, CAULLET Pierre, DUTHOIT Thomas
TDB2, Groupe Avalpierre
Mars 2023

1er livrable CMJ : Compte rendu

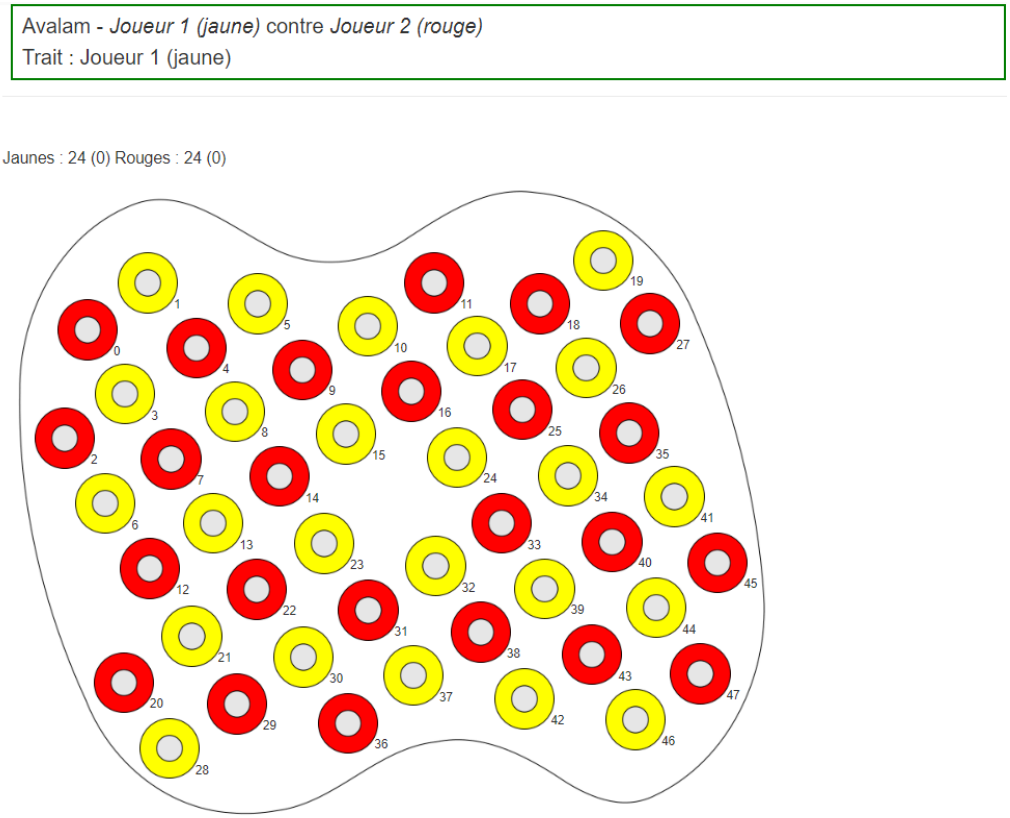
Introduction :

Contexte :

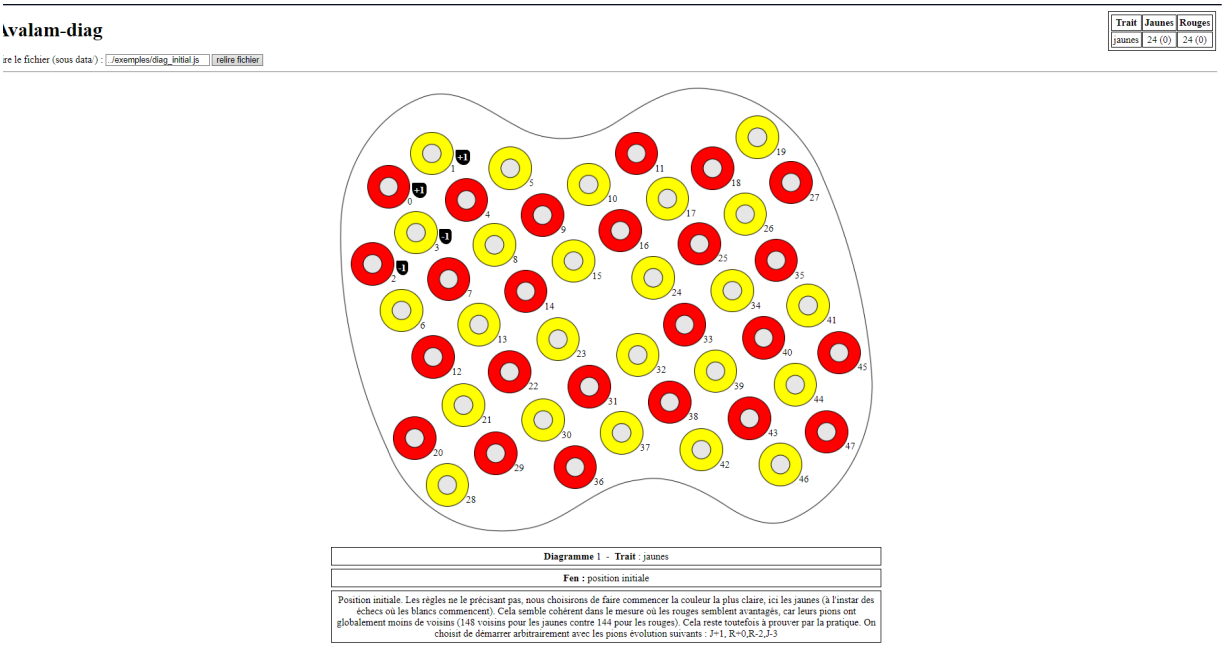
Dans le cadre de la matière "Challenge Moteur de Jeux", M. Bourdeaud'huy nous propose de réaliser, pour le premier livrable, deux programmes (standalone.c et diag.c) qui nous serviront de base pour les prochains livrables.

Objectifs :

Le cahier des charges de standalone nous impose de réussir à pouvoir jouer, en ligne de commande, au jeu Avalam en créant un fichier JS (contenant les scores et l'état du jeu) lu par avalam-refresh. Par ailleurs, le programme devra afficher le gagnant et doit permettre de passer en paramètre un chemin répertoriant où le fichier JS doit être créé. Il est aussi possible de faire un mode debug.



Le cahier des charges de diag nous impose de passer en ligne de commande un numéro de diagramme et une position de type "FEN" qui servira à décrire l'état du jeu. Au lancement de l'exécutable, il devra permettre de saisir un chemin permettant de référencer l'endroit où le fichier JS correspondant sera créé. Il devra aussi permettre, en ligne de commande, de pouvoir ajouter des notes. Finalement, le fichier créé devra pouvoir être lu par avalam-diag. Il doit être aussi possible de faire des redirections ainsi qu'un mode debug.



Sommaire :

Phase de Réflexion :

Afin de pouvoir créer le programme standalone, il faut réussir à comprendre et assimiler les règles du jeu.

Un plateau de jeu d'avalam est constitué de 48 emplacements, dont 24 contennant un pion de couleur rouge et 24 contenant un pion de couleur jaune. Les jaunes commencent puis chaque joueur joue chacun son tour. Lors d'un tour un joueur peut bouger des pions de couleur rouge ou jaune. Il peut déplacer sur une case voisine en prenant en compte que la tour ne doit pas dépasser 5 pions. Le jeu se termine quand il n'y a plus aucun coup légal. Le gagnant est celui avec le plus de tours de sa couleur (où le sommet correspond à la couleur du joueur). Si il y a égalité, celui avec le plus de tours de hauteur 5 gagne. A la fin du jeu, chaque pion bonus fait gagner un point supplémentaire au joueur qui possède la colonne où il se trouve
Contrairement aux pions malus, qui eux, font perdre un point.

Une règle supplémentaire est l'évolution. On ajoute 4 pions différents des autres qu'on appellera "pion évolution", un malus et un bonus par couleur, qui devront être placés au début de la partie par les joueurs en remplaçant 2 pions de sa couleur

Ensuite, la création passe par une phase de compréhension des librairies fournies, à savoir avalam.h et topologie.h.

Dans la première librairie, il y est référencé plusieurs structures indispensables à la lisibilité et au développement du programme. Dans l'autre librairie, il y est précisé une topologie du jeu qui servira par la suite dans standalone, notamment la position initiale du jeu.

Par ailleurs, il a été indispensable de se renseigner sur différentes fonctions afin de pouvoir créer les fichiers JS, tel que fopen, fclose et fprintf qui permet d'écrire dans le fichier souhaité.

Partie réalisation des Programmes :

Standalone.c:

Standalone a été la première étape qui nous a permit de réfléchir sur tout le fonctionnement de nos deux programmes. De mieux comprendre ce qu'on attendait de nous.

Dans un premier temps, l'objectif était très simple, réussir à jouer un coup, la base du programme et pouvoir créer le fichier JS. On pourra demander aussi les pièces évolutions.
Ensuite il fallait réussir à passer facultativement en paramètre un chemin où l'on voulait créer le fichier JS.
Enfin, il fallait implémenter la fin de partie lorsqu'il ne reste aucun coups légaux.

Jouer un coup:

Pour cela, on a utilisé les fonctions présentes dans avalam.h tel que getCoupsLegaux afin de savoir s'il reste des coups légaux à jouer et jouerCoup afin de pouvoir jouer un coup en updatant l'état des colonnes.

Une fois que la fonction pour jouer un coup serait assez satisfaisante, nous aurions plus qu'à la conditionner dans un while qui vérifie la condition de s' il y a encore des coups légaux à jouer ou non. Si c'est le cas, alors il sera toujours possible au joueur de jouer. C'est le main qui s'occupe de savoir cela via le return de la fonction.

Nous sommes passés par diverses améliorations de la fonction qui va jouer le coup jusqu'à arriver à une version finale satisfaisante.

```
int coupV3(T_Position *p) {  // function to make moves
    int depart, fin;  // decalring variables for the origin and end
    printf("\tcaseO ? : ");  // asking the player which pile to move
    scanf("%d", &depart);  // getting the value with scanf
     printf("\tcaseD ? : ");  // asking the player where to put that
pile
    scanf("%d", &fin);  // getting the value with scanf
     if (estValide(*p, depart, fin)) *p = jouerCoup(*p, depart, fin);
// if the move is legal, we play it and we update the position variable
    else return 1;  // else, we return 1 so the condition of the while
in the main loop is true
    printf("\tOn joue %d -> %d", depart, fin);  // printing which move
was done
    return 0;  // returning 0 (=the move is valid)
}
```

Cette fonction prend en paramètre l'état du plateau avant de demander la case départ et la case d'arrivée de la pile que l'on veut jouer. Il vérifie bien entendu que le coup est valide avant de le jouer.
Bien entendu, on demande à l'utilisateur au début du programme le placement des jetons évolutions. Voici un exemple avec le jeton bonus des jaunes :

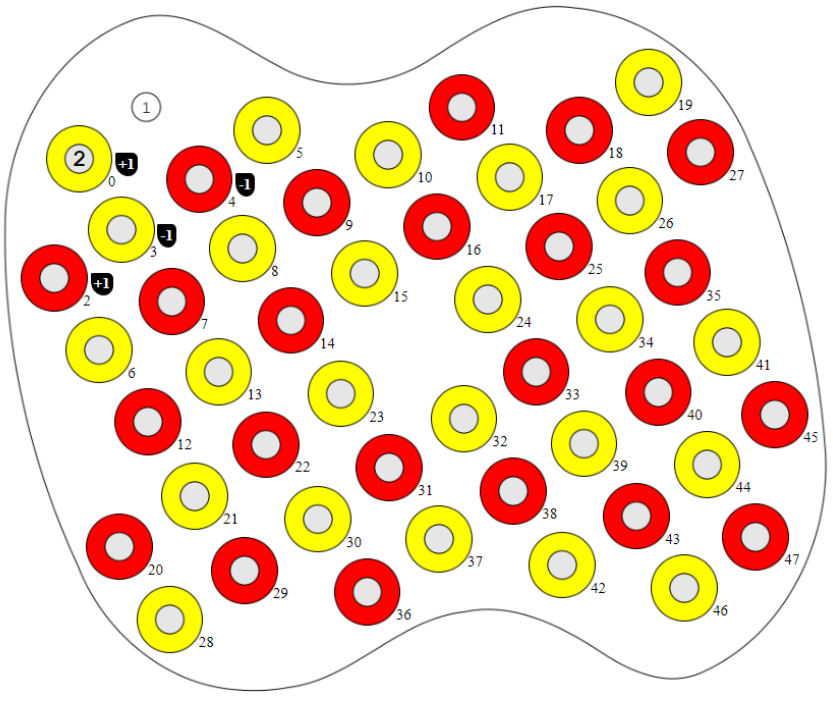
```
while (((0 <= p.evolution.bonusJ || p.evolution.bonusJ > NBCASES) &&
(p.evolution.bonusJ)%2 == 0) || p.evolution.bonusJ == UNKNOWN)
    {
        printf("\tbonusJ ? : ");
        scanf("%hhd",  &(p.evolution.bonusJ));    //  not  %d  but  %hhd
because we need bytes (warning and solution given by gcc)
    }
```

La condition du while permet de faire face à, normalement, n'importe quel cas de jetons évolutions mal placé. Le programme ne fait que lire l'indice de la case où l'utilisateur souhaite poser son jeton évolution et boucle si jamais l'utilisateur met un input non valide d'après les règles du jeu.

Voici un jeu d'essai témoignant de la robustesse des fonctionnalités précédentes.

```
shiro@The-new-one: /mnt/c/Users/ewenb/Documents/GitHub/CMJ$./standalone/standalone.exe
Utilisation du fichier ./web/data/refresh-data.js
Continuer ?
        bonusJ ? : -1
        bonusJ ? : 50
        bonusJ ? : 43
        bonusJ ? : 1
        bonusR ? : 2
        malusJ ? : 1
        malusJ ? : 3
        malusR ? : 4
J: 24 (0 piles de 5) - R : 24 (0 piles de 5)
Trait aux jaunes :
        caseO ? : 1
        caseD ? : 0
On joue 1 -> 0
nb coups possibles : 292
J: 24 (0 piles de 5) - R : 23 (0 piles de 5)
```

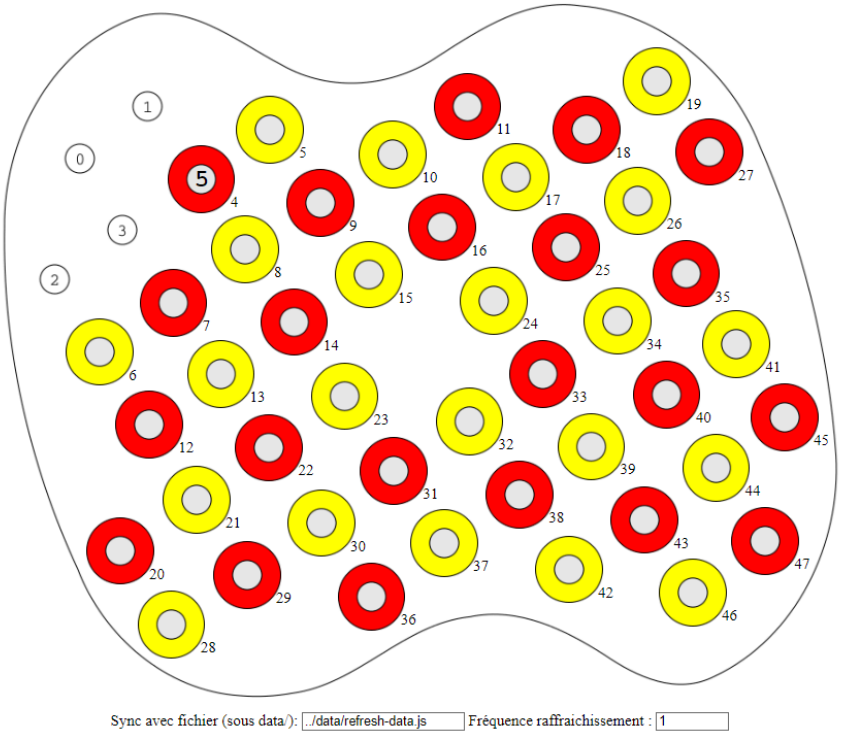
Trait	Jaunes	Rouges
rouge	24 (0)	23 (0)



```
shiro@The-new-one: /mnt/c/Users/ewenb/Documents/GitHub/CMJ$./standalone/standalone.exe
Utilisation du fichier ./web/data/refresh-data.js
Continuer ?
    bonusJ ? : 1
    bonusR ? : 2
    malusJ ? : 3
    malusR ? : 4
J: 24 (0 piles de 5) - R : 24 (0 piles de 5)
Trait aux jaunes :
    caseO ? : 0
    caseD ? : -1
jouerCoup impossible : la colonne 255 est vide !
J: 24 (0 piles de 5) - R : 24 (0 piles de 5)
Trait aux jaunes :
    caseO ? : 47
    caseD ? : 48
jouerCoup impossible : cases 47 et 48 inaccessibles!
J: 24 (0 piles de 5) - R : 24 (0 piles de 5)
Trait aux jaunes :
    caseO ? : 0
    caseD ? : 1
On joue 0 -> 1
nb coups possibles : 292
J: 22 (0 piles de 5) - R : 25 (0 piles de 5)
Trait aux rouges :
    caseO ? : 1
    caseD ? : 0
jouerCoup impossible : la colonne 0 est vide !
J: 22 (0 piles de 5) - R : 25 (0 piles de 5)
Trait aux rouges :
    caseO ? : 1
    caseD ? : 4
On joue 1 -> 4
nb coups possibles : 284
J: 22 (0 piles de 5) - R : 24 (0 piles de 5)
Trait aux jaunes :
    caseO ? : 2
    caseD ? : 3
On joue 2 ->
3nb coups possibles : 278
J: 22 (0 piles de 5) - R : 23 (0 piles de 5)
Trait aux rouges :
    caseO ? : 3
    caseD ? : 4
On joue 3 -> 4
nb coups possibles : 272
J: 22 (0 piles de 5) - R : 22 (1 piles de 5)
Trait aux jaunes :
    caseO ? : 5
    caseD ? : 4
jouerCoup impossible : trop de jetons entre 5 et 4 !
J: 22 (0 piles de 5) - R : 22 (1 piles de 5)
```

Trait aux jaunes :
caseO ? : 34
caseD ? : 6
jouerCoup impossible : cases 34 et 6 inaccessibles!

Trait	Jaunes	Rouges
jaune	22 (0)	22 (1)



Le premier essai sert tout d'abord à vérifier le placement des jetons évolutions. Il arrive à vérifier si le jeton est placé à une case qui n'existe pas ou bien à une case non valide (comme le cas où le jeton est placé sur un jeton de l'autre couleur ou bien si un bonus et un malus sont placé au même endroit, ce qui n'est pas possible). Il vérifie aussi si un coup légal peut correctement être joué, ce qui est le cas. Le deuxième montre la robustesse du programme face à tous les types de coups impossibles, tel qu'un déplacement sur une case qui n'existe pas, vers une case vide ou pleine, ou alors deux cases qui ne sont pas à côté l'une de l'autre.

Afin d'écrire le fichier JS, on utilise une fonction nommé ecrireJSON qui prendra en paramètre l'état du jeu ainsi que le chemin dans lequel il doit écrire le fichier.

```
int ecrireJSON(T_Position p, char *chemin) {   // function to write the
game state into a json file
    FILE *fichier;  // declaring variables
      T_Score  s  =  evaluerScore(p);    //  getting  the  score  from  the
position
    int i;
```

```
    fichier = fopen(chemin, "w");  // opening the file
    CHECK_IF(fichier, NULL, chemin);  // security check

     fprintf(fichier, "traiterJson({\n");   // write the start of the
json file
     fprintf(fichier, "\t"STR_TURN":%d,\n", p.trait); // write which
player needs to play

       fprintf(fichier, "\t"STR_SCORE_J":%d,\n", s.nbJ);    // write
yellow's score
     fprintf(fichier, "\t"STR_SCORE_J5":%d,\n", s.nbJ5);   // write the
number of pile of 5 of yellow
     fprintf(fichier, "\t"STR_SCORE_R":%d,\n", s.nbR);   // write red's
score
     fprintf(fichier, "\t"STR_SCORE_R5":%d,\n", s.nbR5);   // write the
number of pile of 5 of red

     fprintf(fichier, "\t"STR_BONUS_J":%d,\n", p.evolution.bonusJ);   //
write the position of the yellow bonus piece
     fprintf(fichier, "\t"STR_MALUS_J":%d,\n", p.evolution.malusJ);   //
write the position of the yellow malus piece
     fprintf(fichier, "\t"STR_BONUS_R":%d,\n", p.evolution.bonusR);   //
write the position of the red bonus piece
     fprintf(fichier, "\t"STR_MALUS_R":%d,\n", p.evolution.malusR);   //
write the position of the red malus piece

    fprintf(fichier, "\t"STR_COLS":[\n");  // write the columns state

for (i = 0; i < NBCASES; i++)
    {
                fprintf(fichier,  "\t\t{"STR_NB":%d,  "STR_COULEUR":%d}",
p.cols[i].nb, p.cols[i].couleur);
        if (i < NBCASES-1) fprintf(fichier, ",\n");
    }
    fprintf(fichier, "\n\t]\n");


    fprintf(fichier, "});");  // write the end of the json file

    // closing the file
    fclose(fichier);
    return 1;
```

La fonction permet une indentation claire et écrit chacun des éléments nécessaires au bon fonctionnement d’avalam-refresh dans le fichier JS.

Chemin en paramètre:

Le cahier des charges nous imposait de pouvoir passer en paramètre un chemin où le fichier JS sera créé à l'exécution du programme. Pour cela, on utilise argc et argv[] où argc correspond au nombre d'argument et argv[] correspond aux arguments.

```
// updating the JSON file after each move
if (argc == 1) ecrireJSON(p, CHEMIN_PAR_DEFAUT);    // if no path
specified, we use the default one
else ecrireJSON(p, argv[1]);  // else, we use the specified path
```

Ce simple if permet de vérifier si il y a un argument, si c'est le cas, c'est qu'aucun chemin n'a été spécifié en ligne de commande, on utilisera donc le chemin par défaut. Sinon, on utilisera le chemin spécifié. Le chemin utilisé est d'ailleurs spécifié au début du lancement du programme en reprenant la même condition de cette manière :

```
if (argc == 1) printf("Utilisation du fichier %s\n",
CHEMIN_PAR_DEFAUT);  // if no path is specified, we use the default one
else printf("Utilisation du fichier %s\n", argv[1]);  // else, we use
the path specified as an argument
```

Voici un jeu d'essai montrant dans un cas où aucun chemin est spécifié, et dans l'autre où un chemin est spécifié, nous tenterons aussi de créer un fichier JS à l'aide d'un chemin absolu. On tentera par ailleurs de créer un fichier JS dans un répertoire qui n'existe pas pour voir si le programme nous renvoie une erreur. On coupera le programme après avoir placé les jetons évolutions, nous ne souhaitons pas jouer une partie mais simplement voir où le fichier a été créé.

```
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$ ./standalone/standalone.exe
Utilisation du fichier ./web/data/refresh-data.js
Continuer ?
        bonusJ ? : 1
        bonusR ? : 2
        malusJ ? : 3
        malusR ? : 4
J: 24 (0 piles de 5) - R : 24 (0 piles de 5)
Trait aux jaunes :
        caseO ? : ^C
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$ ls -l web/data/
total 8
-rwxrwxrwx 1 shiro shiro 1345 Mar  4 09:36 diag.js
-rwxrwxrwx 1 shiro shiro 1349 Mar  4  2024 refresh-data.js
```

```
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$./standalone/standalone.exe
./web/refresh.js
Utilisation du fichier ./web/refresh.js
Continuer ?
        bonusJ ? : 1
        bonusR ? : 2
        malusJ ? : 3
        malusR ? : 4
J: 24 (0 piles de 5) - R : 24 (0 piles de 5)
Trait aux jaunes :
        caseO ? : ^C
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$ ls -l web
total 36
-rwxrwxrwx 1 shiro shiro 11542 Feb 20 10:31 avalam-diag.html
-rwxrwxrwx 1 shiro shiro  5610 Feb 20 10:31 avalam-refresh.html
-rwxrwxrwx 1 shiro shiro 11044 Feb 20 10:31 avalam-standalone.html
drwxrwxrwx 1 shiro shiro   512 Mar  3 19:45 data
drwxrwxrwx 1 shiro shiro   512 Mar  3 23:03 exemples
drwxrwxrwx 1 shiro shiro   512 Feb 20 10:31 js
-rwxrwxrwx 1 shiro shiro  1349 Mar  4  2024 refresh.js
drwxrwxrwx 1 shiro shiro   512 Feb 20 10:31 ressources
```

```
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$./standalone/standalone.exe
/mnt/c/Users/ewenb/Documents/GitHub/CMJ/web/data/refresh.js
Utilisation du fichier /mnt/c/Users/ewenb/Documents/GitHub/CMJ/web/data/refresh.js
Continuer ?
        bonusJ ? : 1
        bonusR ? : 2
        malusJ ? : 3
        malusR ? : 4
J: 24 (0 piles de 5) - R : 24 (0 piles de 5)
Trait aux jaunes :
        caseO ? : ^C
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$
ls -l /mnt/c/Users/ewenb/Documents/GitHub/CMJ/web/data
total 12
-rwxrwxrwx 1 shiro shiro 1345 Mar  4 09:36 diag.js
-rwxrwxrwx 1 shiro shiro 1349 Mar  4  2024 refresh-data.js
-rwxrwxrwx 1 shiro shiro 1349 Mar  4  2024 refresh.js
```

```
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$./standalone/standalone.exe
./web/test/refresh.js
Utilisation du fichier ./web/test/refresh.js
Continuer ?
        bonusJ ? : 1
        bonusR ? : 2
        malusJ ? : 3
        malusR ? : 4
erreur appel systeme
./web/test/refresh.js: No such file or directory
```

Le programme réagit bien comme escompté, il est capable de créer un fichier, que ce soit avec un chemin relatif ou absolu. Par contre, il n'en crée pas si le répertoire n'existe pas (ce qui est logique)

Une piste d'amélioration de notre programme serait justement de séparer le chemin et le nom du fichier, afin de ne pas devoir taper un chemin complet avec le fichier à créer à l'intérieur. Car pour l'instant, le programme nous renvoie cela :

```
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$./standalone/standalone.exe ./web
Utilisation du fichier ./web
Continuer ?
        bonusJ ? : 1
        bonusR ? : 2
        malusJ ? : 3
        malusR ? : 4
erreur appel systeme
./web: Is a directory
```

En soit, ce n'est pas un bug car cela est tout à fait logique, il faudrait cependant pouvoir réussir à créer un fichier avec un nom par défaut dans le dossier web dans le cas ci-dessus. Ce n'était néanmoins pas demandé mais nous avons jugé intéressant de le signaler.

Fin de partie :

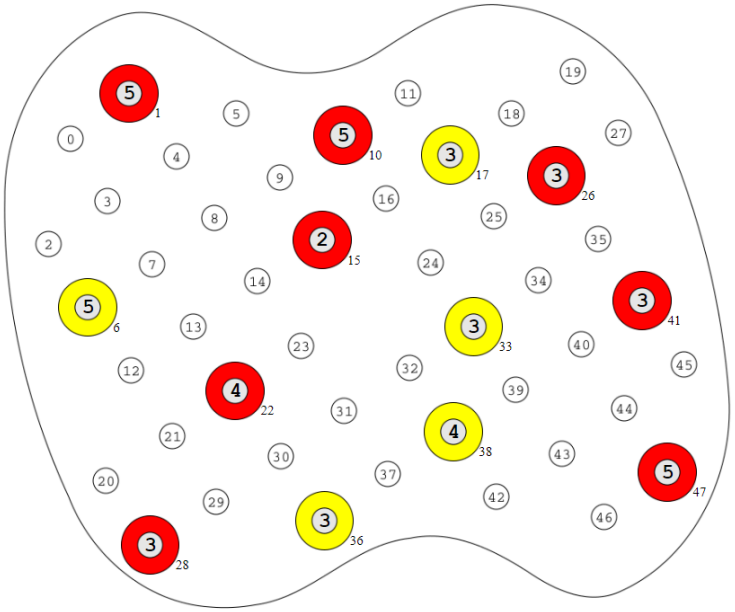
Il ne reste plus qu'à implémenter la fin de partie dès que l'on sort du while qui faire dérouler le jeu.

```
    // the game is finished:
    printf("Partie finie !\nScore: ");
    afficherScore(s);  // displaying score
     if (s.nbJ > s.nbR) printf(" %s gagnent\n", joueur1.pseudo);   //
player 1 wins
     else if (s.nbJ < s.nbR) printf(" %s gagnent\n", joueur2.pseudo);
// player 2 wins
    else if (s.nbJ == s.nbR) {
          if (s.nbJ5 > s.nbR5) printf(" %s gagnent\n", joueur1.pseudo);
// player 1 wins because he has more piles of 5
               else if (s.nbJ5 < s.nbR5) printf(" %s gagnent\n",
joueur2.pseudo);  // player 2 wins because he has more piles of 5
        else printf(" Egalité\n");  // exact same scores
    }
```

Voici comment s'affiche une fin de partie (pour des raisons de longueur du jeu d'essaie, il ne contiendra que le début du programme ainsi que la fin de partie, pas les déplacements des pièces, ils seront représentés par "…")

```
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$ ./standalone/standalone.exe
Utilisation du fichier ./web/data/refresh-data.js
Continuer ?
    bonusJ ? : 1
    bonusR ? : 2
    malusJ ? : 3
    malusR ? : 4
…
Partie finie !
Score: J: 6 (1 piles de 5) - R : 8 (3 piles de 5)
```

Trait	Jaunes	Rouges
rouge	5 (1)	8 (3)



Sync avec fichier (sous data/): ./data/refresh-data.js Fréquence raffraichissement : 1

Debug :

Il était aussi demandé de faire un mode debug au programme. Dans notre cas, nous avons parsemé notre programme de différents print (défini dans avalam.h) qui ne s'active que si une constante symbolique est défini à la compilation. Voici un exemple du programme en mode debug (ici on ne va pas jusqu'à la fin de partie car rien n'a été ajouté à la fin pour le mode debug) :

```
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$ ./standalone/standalone.exe
…
Utilisation du fichier ./web/data/refresh-data.js
Continuer ?
Demande aux joueurs où ils veulent placer leurs pièces d'évolution.
Demande au joueur jaune de placer le jeton bonus.
    bonusJ ? : 1
Le jeton bonus jaune a été placé à la case 1
Demande au joueur rouge de placer le jeton bonus.
    bonusR ? : 2
Le jeton bonus rouge a été placé à la case 2
Demande au joueur jaune de placer le jeton malus.
    malusJ ? : 3
Le jeton malus jaune a été placé à la case 3
Demande au joueur rouge de placer le jeton malus.
    malusR ? : 4
Le jeton malus rouge a été placé à la case 4
J: 24 (0 piles de 5) - R : 24 (0 piles de 5)
Trait aux jaunes :
Demande au joueur quelle pile il veut déplacer.
    caseO ? : 0
Demande au joueur où il veut déplacer la pile.
    caseD ? : 1
```

N.B : Le mode débug active un print de tous les coups possibles dès que l'on récupère les coups légaux grâce à la fonction, soit au début du programme et entre temps. Il est défini dans la fonction.

Diag.c:

Afin de réussir à réaliser diag au complet, notre approche a été de découper en plusieurs parties le processus de développement :

- Vérification du nombre d'arguments et récupération
- Ecriture du JSON
- Possibilité d'indiquer un chemin
- Inclusion des notes
- Interprétation de la chaîne FEN

Nombre d'argument :

La première étape était la plus simple : Pouvoir vérifier si le nombre d'arguments passé en paramètre était au nombre de 3. Afin de le vérifier, on utilise les paramètres argc et argv[]. argc permet justement de savoir le nombre d'arguments passés en paramètre. Il faut donc bien vérifier qu'il soit bien au nombre de 3 comme ceci :

```
    if(argc != 3)  // If arg's number are Insufficient
    {
        printf("diag <numDiag> <fen>\n");
        return -1;
    }
```

Voici donc ce qu'il renvoie si jamais le nombre d'argument est insuffisant :

```
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$ ./diag/diag.exe 3
diag <numDiag> <fen>
```

Le programme précise donc que le nombre d'arguments est insuffisant et qu'il doit être lancé en définissant des paramètres précis. Si le nombre d'argument est correct, alors le programme pourra continuer son exécution.

JSON :

Ensuite vint l'écriture du fichier JSON. La fonction est à peu près la même que standalone mise à part que l'on rajoute cette fois certaines lignes propre à diag, tel que le numéro de diagramme, les notes ainsi que la chaîne FEN utilisée.

Afin de faciliter l'implémentation des variables dans le fichier produit, nous avons fait le choix de définir une nouvelle structure :

```
typedef struct
{
    int num_diag;  // Diag's number
    char* notes;  // Diag's notes
    char* fen;  // Diag's FEN string.
} T_Diag;
```

T_Diag référence le numéro de diagramme, les notes de l'utilisateur, ainsi que la chaîne FEN. Le numéro de diagramme ainsi que la chaine FEN sont récupérés préalablement grâce à argv[].

Ainsi le prototype de la fonction EcrireJSON s'en retrouve modifié ainsi que l'ajoute de l'écriture du numéro de diagramme, du trait, des notes, et de la chaine FEN :

Prototype de la fonction :

```
int ecrireJSON(T_Position p, char *chemin, T_Diag d)
```

Ajout dans la fonction :

```
fprintf(fichier, "\t"STR_TURN":%d,\n", p.trait); // Write the "trait"
fprintf(fichier, "\t"STR_NUMDIAG":%d,\n", d.num_diag);  // Write diag's
number
fprintf(fichier, "\t"STR_NOTES":\"%s\",\n", d.notes);  // Write notes
fprintf(fichier, "\t"STR_FEN":\"%s\",\n", d.fen); // Write the FEN
```

L'une des choses qui peut être assez troublante est que d.num_diag est considéré ici comme un entier. Or, lorsque l'on place quelque chose en paramètre lors de l'exécution d'un programme est considéré comme un string.

Après l'étude de l'exécutable de diag fournit en réalisant plusieurs tests pour comprendre son fonctionnement, on comprend assez vite que la chaîne de caractère représentant le numéro de diagramme est convertie en entier. On utilise donc une fonctionnalité présente dans la librairie standard nommée atoi().

Cette fonction prend en paramètre une chaîne de caractère qu'elle convertit alors en un entier. Cependant, si la chaîne de caractère ne contient pas que des digits (0 à 9) alors elle renverra 0 comme en témoigne les jeux d'essai suivants (qui sont donc exactement les mêmes jeux d'essai que notre programme, nul besoin de le refaire pour notre programme exclusivement pour cela).

```
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$./build/liv1.x86.diag.static
33 "dtu20U5u r"
Diagramme 33
Fen : dtu20U5u r
```

```
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$./build/liv1.x86.diag.static
test "dtu20U5u r"
Diagramme 0
Fen : dtu20U5u r
```

Choix du chemin :

La troisième étape était de pouvoir référencer un chemin où il fallait créer le fichier JS. Sinon, on utilisait un chemin par défaut et ceci, de cette manière :

```
printf("Fichier (sera créé sous la forme %s si possible) ?",
CHEMIN_PAR_DEFAUT);
fgets(chemin, MAX_K, stdin); // Path's input
format(chemin);
```

MAX_K est une constante définie par nous-mêmes, référençant le nombre de caractère maximum que peut accepter le tableau de char chemin. Dans notre cas, elle est définie à 1000 mais peut être modifiée.

fgets() est très pratique pour récupérer les inputs de l'utilisateur. Elle comporte un petit défaut qui est l'ajout de '\n' (retour à la ligne) à la fin. D'où la fonction format qui permet de le retirer.

```
void format(char ch[])  // Remove '\n' due to fgets
{
    int i = 0;
    while(ch[i] != '\0') i++;
    if(ch[i-1] == '\n') ch[i-1] = '\0';
}
```

On remplace le '\n' par un caractère fin de chaîne car le programme ne lit pas ce qui a après un caractère '\0'.

Il fallait alors aussi définir un chemin par défaut grâce à un define au cas où l'utilisateur ne référence aucun chemin :

```
#define CHEMIN_PAR_DEFAUT "./web/data/diag.js"
```

Le fichier est créé grâce à la fonction ecrireJSON() où l'on donne le chemin en paramètre. Il ne requiert donc que l'utilisation d'un if pour vérifier si chemin[] est vide ou non. Si il est vide, alors on utilise le chemin par défaut.

```
if(chemin[0] == '\0')   // Creating and writing the file depending of
the user's input.
    ecrireJSON(p, CHEMIN_PAR_DEFAUT, d);
else
    ecrireJSON(p, chemin, d);
```

Une potentielle piste d'amélioration de notre programme serait de séparer le nom du fichier par défaut ainsi que le chemin. Car actuellement, afin de créer le fichier JS, le programme écrit dans le chemin référencé. En clair, si on ne référencie qu'un répertoire déjà existant, alors, le programme renvoie une erreur. Il faut aussi écrire l'extension du fichier.

```
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$
./diag/diag.exe 33 "dtu20U5u r"
Diagramme 33
Fen : dtu20U5u r
Fichier (sera créé sous la forme ./web/data/diag.js si possible) ?
Description (vous pouvez saisir du HTML, 1000 caractères max, Ctrl+D pour terminer) ? []

Description : “”
Enregistrement de ./web/data/diag.js
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$ cd web/data/
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ/web/data$ ls -l
total 12
-rwxrwxrwx 1 shiro shiro 1349 Mar  3 19:31 diag.js
-rwxrwxrwx 1 shiro shiro 1349 Mar  3 17:18 refresh-data.js
-rwxrwxrwx 1 shiro shiro 1409 Feb 22 14:51 test.js
```

```
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$ ./diag/diag.exe 33 "dtu20U5u r"
Diagramme 33
Fen : dtu20U5u r
Fichier (sera créé sous la forme ./web/data/diag.js si possible) ?./web/exemples
Description (vous pouvez saisir du HTML, 1000 caractères max, Ctrl+D pour terminer) ? []
Description : ""
erreur appel systeme
./web/exemples: Is a directory
```

```
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$ ./diag/diag.exe 33 "dtu20U5u r"
Diagramme 33
Fen : dtu20U5u r
Fichier (sera créé sous la forme ./web/data/diag.js si possible) ?./web/exemples/fic1.js
Description (vous pouvez saisir du HTML, 1000 caractères max, Ctrl+D pour terminer) ? []

Description : ""

Enregistrement de ./web/exemples/fic1.js
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$ cd web/exemples/
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ/web/exemples$ ls -l
total 20
-rwxrwxrwx 1 shiro shiro   76 Feb 20 10:31 description
-rwxrwxrwx 1 shiro shiro 1548 Feb 20 10:31 diag_another.js
-rwxrwxrwx 1 shiro shiro 1855 Feb 20 10:31 diag_initial.js
-rwxrwxrwx 1 shiro shiro 1548 Feb 20 10:31 diag_test.js
-rwxrwxrwx 1 shiro shiro 1350 Mar  3 19:42 fic1.js
-rwxrwxrwx 1 shiro shiro 1352 Feb 20 10:31 refresh-data.js
```

Dans le premier cas, le fichier JS est correctement enregistré dans le chemin par défaut. Dans le second cas, l'objectif était de créer un fichier js dans le dossier exemples dans web mais cela n'a pas marché car exemples est un dossier. Si l'on voulait créer un fichier js dans exemples avec notre programme, la bonne syntaxe serait "./web/exemples/fic1.js" comme montré dans le 3ème jeu d'essai.

Notes :

Ensuite, il fallait réussir à pouvoir inclure des notes de façon intéractive. Il devait être possible d'effectuer des retours à la ligne qui seraient ensuite traduit en HTML par une balise
.

```
#define BREAK "<br />"
```

fgets() remplace cependant l'ancien contenu d'une chaîne de caractère par un nouveau dés que l'on confirme l'input et ce n'est pas ce qu'on veut car il faut enregistrer la chaîne complète une fois que l'on appuie sur CTRL+D sur une ligne vide

Après beaucoup de recherches de documentations sur stackoverflow principalement, il a été choisi de passer par une chaîne intermédiaire que l'on concatène à la chaîne des notes tant qu'elle n'est pas vide.

```
    while(fgets(temp, MAX_K, stdin) != NULL)  // Description's input
    {
        format(temp);
        if(description[0] != '\0')
            concat(description, BREAK);
        concat(description, temp);
    }
```

```
void concat(char dest[], char src[])  // Concatenate 2 strings (useful
for notes)
{
    int i = 0, j = 0;
    while(dest[i] != '\0') i++;
    while(src[j] != '\0')
    {
        dest[i] = src[j];
        i++;
        j++;
    }
    dest[i] = '\0';
}
```

La fonction concat permet de concaténer deux chaînes ensemble. On va en premier lieu à la fin de la chaine où l'on veut concaténer avant de recopier l'autre chaine entièrement.

Le if présent dans le while de l'input permet de savoir si la chaîne de notes est vide ou non, histoire de ne pas placer de balise
 avant le texte. Voici une liste de jeu d'essai témoignant de la solidité du programme (que ce soit vide ou multilignes) :

```
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$ ./diag/diag.exe 33 "dtu20U5u r"
Diagramme 33
Fen : dtu20U5u r
Fichier (sera créé sous la forme ./web/data/diag.js si possible) ?
Description (vous pouvez saisir du HTML, 1000 caractères max, Ctrl+D pour terminer) ? []

Description : ""

Enregistrement de ./web/data/diag.js
```

```
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$ ./diag/diag.exe 33 "dtu20U5u r"
Diagramme 33
Fen : dtu20U5u r
Fichier (sera créé sous la forme ./web/data/diag.js si possible) ?
Description (vous pouvez saisir du HTML, 1000 caractères max, Ctrl+D pour terminer) ? []
ceci est un test

Description : "ceci est un test"

Enregistrement de ./web/data/diag.js
```

```
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$ ./diag/diag.exe 33 "dtu20U5u r"
Diagramme 33
Fen : dtu20U5u r
Fichier (sera créé sous la forme ./web/data/diag.js si possible) ?
Description (vous pouvez saisir du HTML, 1000 caractères max, Ctrl+D pour terminer) ? []
ceci est un test
et un autre test
et encore un autre test

Description : "ceci est un test<br />et un autre test<br />et encore un autre test"

Enregistrement de ./web/data/diag.js
```

Interprétation de la chaîne FEN:

Enfin, la dernière étape était l'interprétation de la chaîne FEN. Après réflexion, la logique utilisée a été d'étudier chacun des caractères qui comporte la chaîne FEN et réagir en conséquence grâce à un switch case en respectant les cases suivants précisée dans le cahier des charges.

A l'aide de différents tests sur l'exécutable exemple de diag fournie et après réflexion, une liste de standards arbitraires a été défini comme suit :

- Si aucun caractère ne définit le trait : Le trait sera égal à 0.
- Le trait peut être défini dans la partie de la chaîne qui référencie l'état du plateau.
- Si il existe plusieurs caractères qui définissent le trait, alors on prend le dernier.
- Si la chaîne FEN possède un caractère inconnu, alors le programme ignore le dit caractère.

Un exemple ci-dessous d'une des case :

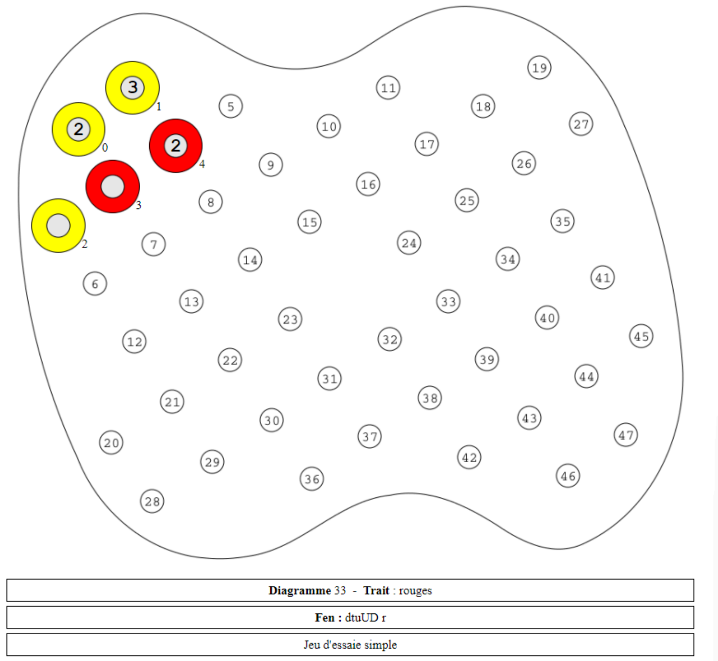
```
while(d.fen[i] != '\0' && case_prise < NBCASES) // All possibles cases
for the FEN interpretation
    {
        switch(d.fen[i])
        {
        case 'u':
            p.cols[case_prise].couleur = 1;
            p.cols[case_prise].nb = 1;
            case_prise++;
            break;
      …
      i++;
```

Voici donc un jeu d'essaie qui témoigne du fonctionnement d'une chaîne FEN en considérant qu'il n'y a pas de case vide intermédiaire ainsi qu'un essai pour voir si le programme est assez robuste, s'il y a trop de caractère dans la chaîne FEN mais aussi trop peu.

shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$ ./diag/diag.exe 33 "dtuUD r"
Diagramme 33
Fen : dtuUD r
Fichier (sera créé sous la forme ./web/data/diag.js si possible) ?
Description (vous pouvez saisir du HTML, 1000 caractères max, Ctrl+D pour terminer) ? []
Jeu d'essaie simple

Description : "Jeu d'essaie simple"

Enregistrement de ./web/data/diag.js

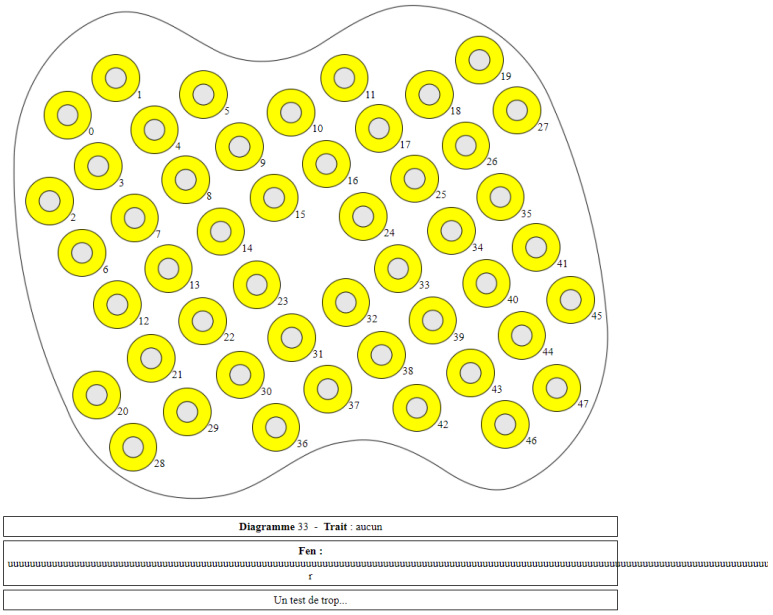


---

shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$./diag/diag.exe 33 "uu r"
Diagramme 33
Fen : uuu r
Fichier (sera créé sous la forme ./web/data/diag.js si possible) ?
Description (vous pouvez saisir du HTML, 1000 caractères max, Ctrl+D pour terminer) ? []
Un test de trop...

Description : "Un test de trop..."

Enregistrement de ./web/data/diag.js



Dans ces deux jeux d'essai, on peut voir que l'affichage de la part d'avalam-diag marche comme prévu. De plus, il est robuste dans les deux cas grâce à la condition du while mais aussi d'une initialisation à 0 des valeurs des colonnes.

```
//  Initializing before FEN interpretation
    p.trait = 0;
    p.evolution.bonusJ = UNKNOWN;
    p.evolution.bonusR = UNKNOWN;
    p.evolution.malusJ = UNKNOWN;
    p.evolution.malusR = UNKNOWN;
    for(i = 0; i < NBCASES; i++)
    {
        p.cols[i].couleur = 0;
        p.cols[i].nb = 0;
    }
    i = 0;
```

Avant l'ajout des nombres pour indiquer des cases vides, la méthode utilisée était de dire que le numéro de la case ainsi que l'indice "i" étaient les mêmes car il y avait correspondance entre les deux. Cependant dès que l'on ajoute un nombre de cases vides intermédiaires grâce aux nombres ou bien un caractère inconnu, l'indice i ne correspond plus forcément à l'indice de la case qui va être remplie. Le choix le plus logique a donc été de créer une variable case_prise afin de pouvoir savoir quel est la prochaine case qui va être prise.

Pour la partie gérant le fonctionnement des nombres, la voici :

```
        default:
            if('0' <= d.fen[i] && d.fen[i] <= '9')
            {
                while('0' <= d.fen[i] && d.fen[i] <= '9')
                {
                    num[j] = d.fen[i];
                    j++;
                    i++;
                }
                num[j] = '\0';
                j = 0;
                case_prise = case_prise + atoi(num);
                i--;
            }
```

De base, le case default est un case qui ne correspond à aucun de ceux précédents. Cependant, le programme va donc vérifier si le caractère est un digit. Si ça l'est, alors le programme va lire les chiffres indice par indice jusqu'à ce que ce ne soit plus un chiffre en le recopiant à un tableau temporaire. num est convertie en int par atoi() et on ajoute à case_prise ce que nous retourne atoi(), car grâce à l'initialisation à 0 précédente, il n'y a pas besoin de dire que chaque case intermédiaire à nb = 0 et couleur = 0 mais à la place, directement aller à l'indice de la prochaine case qui va être remplie.

Voici donc les jeux d’essai finaux qui témoignent de la robustesse du programme :

```
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$ ./diag/diag.exe 33 "dtu20U5u r"
Diagramme 33
Fen : dtu20U5u r
Fichier (sera créé sous la forme ./web/data/diag.js si possible) ?
Description (vous pouvez saisir du HTML, 1000 caractères max, Ctrl+D pour terminer) ? []
Test final
nVide intermédiaire

Description : "Test final<br />nVide intermédiaire"

Enregistrement de ./web/data/diag.js
```

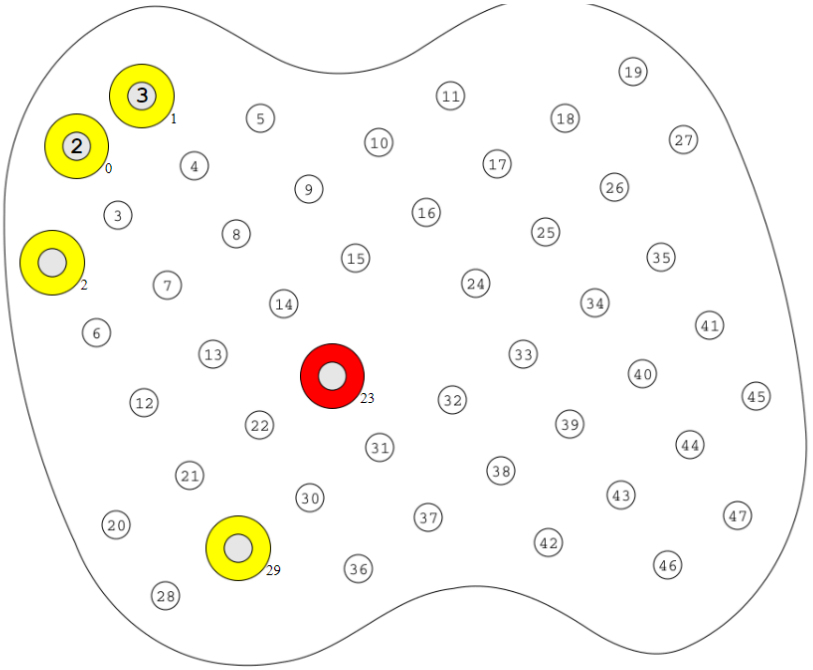


Diagramme 33 - **Trait** : rouges
Fen : dtu20U5u r
Test final nVide intermédiaire

```
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$ ./diag/diag.exe 33 "dtu;b;D50u j"
Diagramme 33
Fen : dtu;b;D50u j
Fichier (sera créé sous la forme ./web/data/diag.js si possible) ?
Description (vous pouvez saisir du HTML, 1000 caractères max, Ctrl+D pour terminer) ? []
Test final
Vide
Caractère inconnu

Description : "Test final<br />Vide<br />Caractère inconnu"

Enregistrement de ./web/data/diag.js
```

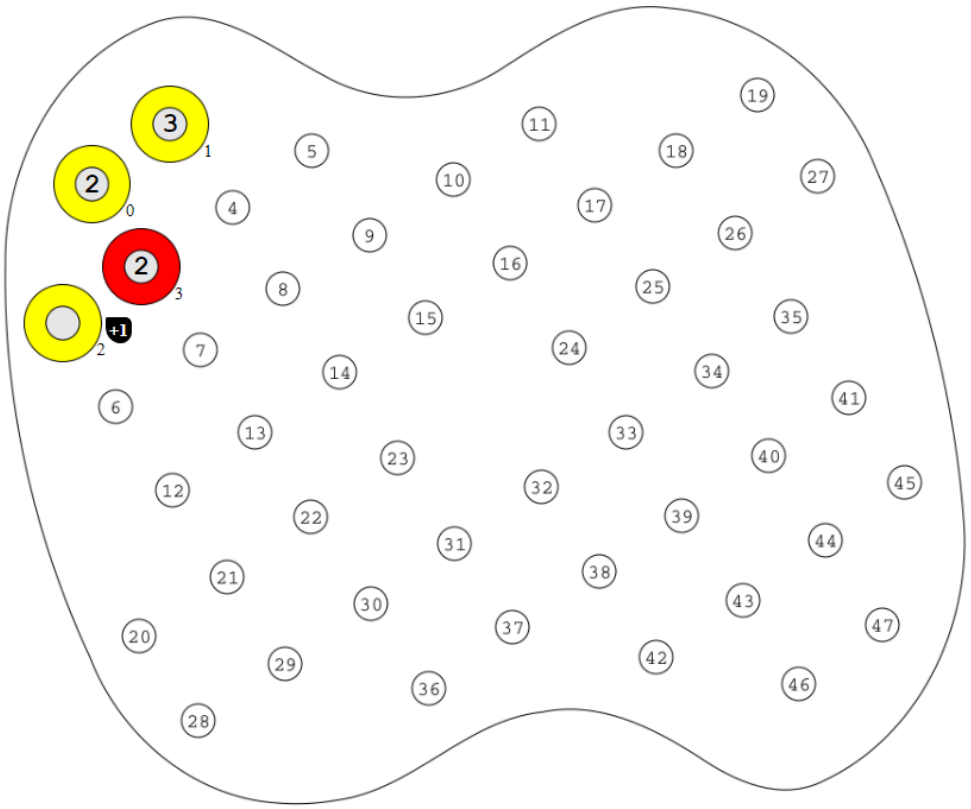


Diagramme 33 - **Trait** : jaunes

Fen : dtu;b;D50u j

Le premier test est témoin du fait que le programme tolère donc les nombres, même à deux chiffres. Le deuxième est témoin du fait que le programme est robuste et ne fait aucune segmentation fault au-delà de NB_CASES mais aussi qu'il ignore correctement les caractères non reconnus et qu'il permet l'inclusion des pièces évolutions.

Une petite remarque concernant avalam-diag.html, celui-ci est censé afficher les scores en haut à droite. Cependant, il n'affiche aucun score, et ce, même si les scores sont précisés dans le fichier JS.

Redirection et Debug :

Diag devrait être aussi capable de supporter une redirection afin de mettre le chemin du fichier à créer ainsi que les notes qui l'accompagnent.

Cependant, notre groupe a rencontré un blocage, en effet, le fichier ne se crée pas tout à faire correctement, du moins, son nom. Son contenu, quant à lui, se crée correctement.

Démonstration :

```
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$ cat redir
fic1.js
Une description sur
deux lignes
ou plus
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$ ./diag/diag.exe 33 "dtu;b;D50u" < redir
Diagramme 33
Fen : dtu;b;D50u
Fichier (sera créé sous la forme ./web/data/diag.js si possible) ?Description (vous pouvez saisir du HTML, 1000 caractères max, Ctrl+D pour terminer) ? []

<br />ou plus"nese description sur

Enregistrement de fic1.js
```

En dehors du fait que l'affichage soit assez étrange, le nom du fichier n'est pas le bon : fic1.js* où * représente un caractère quelconque qui ne s'affiche pas. Cependant, ce caractère ne se crée pas s'il n'y avait aucune description.
Une tentative pour fixer le problème était dans la fonction de format si jamais le caractère encore avant était un retour à la ligne, mais cela n'a rien changé.

Concernant le mode debug, il permet de faire une version légèrement plus verbeuse du programme. Voici un exemple en utilisant un des tests finaux :

```
shiro@The-new-one:/mnt/c/Users/ewenb/Documents/GitHub/CMJ$ ./diag/diag.exe 33 "dtu;b;D50u j"
Diagramme 33
Fen : dtu;b;D50u j
Fichier (sera créé sous la forme ./web/data/diag.js si possible) ?
Description (vous pouvez saisir du HTML, 1000 caractères max, Ctrl+D pour terminer) ? []

Description : ""
Début de l'interprétation de la FEN
Trait non défini, recherche dans la FEN
Le trait est aux jaunes
Chemin non défini
Enregistrement de ./web/data/diag.js
```

Conclusion et Problèmes externes:

En conclusion, les deux programmes possèdent les fonctionnalités attendues dans le cahier des charges.

Ce travail nous a permis de développer notre sens logique, à se documenter notamment sur la création de fichier en C. Mais aussi à se documenter sur des problèmes que l'on peut se poser sur différents sites spécialisés (stackoverflow principalement). Il nous a aussi appris de force à créer une méthode de travail différente de ce que l'on a l'habitude de faire au vu de la taille conséquente du projet. Il nous a aussi créé un esprit de travail en groupe comme jamais fait auparavant, les fichiers nous ont aussi montré leurs limites.

Parfois la simple documentation ne suffit pas, comme par exemple avec le problème de la redirection avec diag où cela dépasse nos capacités. En clair, les projets nous apprennent parfois à devoir contourner les différents problèmes rencontrés, à réagir face à eux.