COMPTE RENDU : Challenge moteur de jeu LIVRABLE n° 1

Nom de l'équipe : AVALAM FLEX
Groupe TP : B2
Numéro de groupe : 14

Nom et prénom des membres :

- BOURNONVILLE Charles
- GOUDAL Louis
- GOURDON Thomas
- LEROUX Léo

Introduction :

Notre projet se base sur le jeu de plateau : AVALAM

Le but final de ce projet et de créer un algorithme pouvant gagner contre un humain, nous sommes dans la première partie de ce projet, le livrable 1, dans ce livrable nous avons pour but de pouvoir jouer des parties d'Avalam entre deux joueurs au clavier et de pouvoir générer des plateaux de jeux en fonctions d'une chaîne de caractère

Pour mener à bien ce livrable n°1 nous avons donc dû réaliser ces grandes étapes :

standalone.exe : Jeu à deux joueurs, saisie des coups au clavier, affichage par avalam-refresh

- Permet de jouer à deux joueurs
- Saisie des coups chacun son tour au clavier
- A chaque coup, produit un fichier json avec la position et le score
- Ce fichier sera utilisé par avalam-refresh.html; Le nom du fichier facultatif peut être passé en ligne de commandes ; Un nom par défaut est prévu dans le programme
- Détecte la fin de la partie et affiche le score à l'écran

diag.exe : Saisie de positions sur la ligne de commande, affichage par avalam-diag

- Permet de passer, en ligne de commandes : un numéro du diagramme ; une position, type "FEN"
- Une fois l'exécutable lancé, saisies au clavier : du nom du fichier à produire [un nom par défaut est prévu] ; d'une chaîne de description [vide par défaut]
- Produit un fichier json avec la position et la description, pouvant être utilisé par avalam-diag

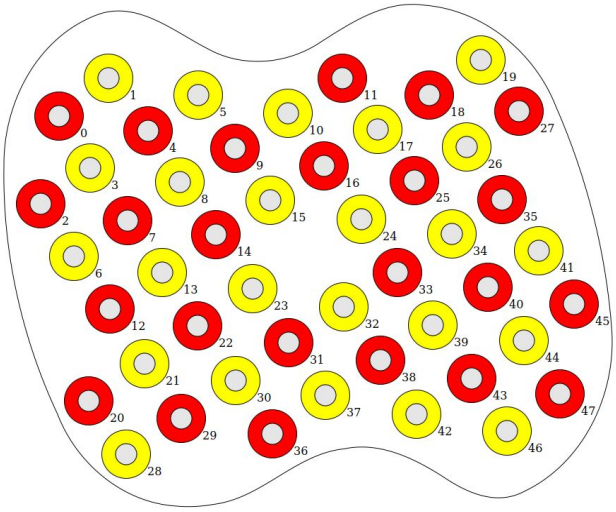
---

standalone.exe :

Avec standalone.exe, le/les utilisateurs doivent pouvoir lancer le programme en indiquant ou non le fichier dans lequel ils veulent sauvegarder leur partie.
Il faut aussi que les coups puissent être entrés au clavier et que chaque coup mette à jour la page web.

Exemple de fonctionnement :

Trait	Jaunes	Rouges
jaune	24 (0)	24 (0)

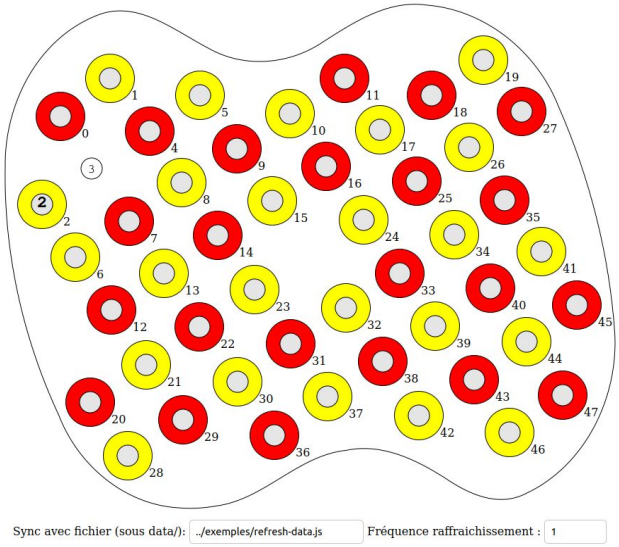


ces données permettent de faire ce coup :

A jaune de jouer
Entrer origine
3

Entrer destination
2
J : 24 (0 piles de 5) - R : 23 (0 piles de 5)
A rouge de jouer

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



Pour ce faire nous avons programmé 3 différentes fonctions que nous allons vous expliquer juste en dessous.

1 - main

On initialise l'adresse d'écriture du fichier json par défaut
char fic[] = "../build/web/exemples/refresh-data.js";

Ensuite, si lors de l'exécution du programme il n'y a qu'un paramètre, on exécute standalone() qui écrira le json à l'adresse par défaut
```
if(argc == 1) {
    standalone(fic); }
```

Si il y a deux paramètres d'entrée, on exécute standalone() qui écrira le fichier json à l'adresse indiquée(si celle ci se trouve être fausse, cela peut créer une erreur core dumped)
```
else if(argc == 2) {
     standalone(argv[1]);}
```

Enfin, si il y a plus de 2 paramètres d'entrée, on indique à l'utilisateur qu'il y a trop de paramètres.
```
 else {
    printf("erreur trop d'arguments"); }
```

2 - standalone

Pour récupérer les positions que l'utilisateur rentres et détecter la fin de partie, il nous fallait une fonction , nous l'avons donc appelée standalone, voici son prototype : *standalone(char *fic);*

On initialise les variables plateau,CoupLeg,coup,score
On effectue un refresh() expliqué ci dessous, du plateau,
et on initialise le trait au joueur Jaune par défaut
```
  T_Position plateau = getPositionInitiale();
  T_ListeCoups CoupLeg;
  T_Coup coup;
  T_Score score;
  Refresh(fic,plateau);
  plateau.trait = JAU; //jaune commence
```

On demande ensuite au joueur le coup qu'il souhaite jouer et on récupère les valeurs indiquées.
On donne à plateau la valeur *jouercoup(jouerCoup(plateau,coup.origine,coup.destination)*
) qui nous renvoie la nouvelle position et on vérifie ensuite si le coup est possible à l'aide de la fonction *getCoupLegaux()*

On refresh() ensuite le plateau, et on met à jour le score à l'aide de la fonction *evaluerScore()*

```
printf("A %s de jouer\n",COLNAME(plateau.trait));
printf("Entrer origine\n");
scanf("%hhd",&coup.origine);
printf("\nEntrer destination\n");
scanf("%hhd",&coup.destination);
plateau = jouerCoup(plateau,coup.origine,coup.destination);
CoupLeg =  getCoupsLegaux(plateau);
Refresh(fic,plateau);
score = evaluerScore(plateau);
afficherScore(score);
```

En version debug le programme indiquera aussi :

- le point d'origine
- la destination
- le nombre de coup légaux
- le trait
- le score des jaunes
- le nombre de tour jaune à 5
- le score des rouges
- le nombre de tour rouge à 5

```
#ifdef __DEBUG__
        printf("\n__DEBUG__ ");
        printf("\ncoup origine saisit : %hhd\n",coup.origine);
        printf("coup destination saisit : %hhd\n",coup.destination);
        printf("Nb coup légaux <%d>\n",CoupLeg.nb);
        printf("trait: <%d>\n", plateau.trait);
        printf("Nb points Jaune: <%d>\n", score.nbJ);
        printf("Nb tour à 5 Jaune: <%d>\n", score.nbJ5);
        printf("Nb points Rouge: <%d>\n", score.nbR);
        printf("Nb tour à 5 Rouge: <%d>\n", score.nbR5);
        printf("FIN __DEBUG__");
#endif
```

On recommencera le procédé expliqué ci-dessus jusqu'à qu'il n'y ait plus de coup possible ou alors que le coup indiqué ne soit pas valide.

```
while(CoupLeg.nb != 0 ||!estValide(plateau,coup.origine,coup.destination));
```

On indique ensuite la fin de partie au deux joueurs.
printf("fin de la partie");

3 - Refresh

Pour actualiser le json et ainsi modifier l'affichage web, nous avons décidé de faire une autre fonction que nous avons appelé Refresh, voici son prototype : *Refresh(const char *fic,const T_Position plateau);*

Tout d'abord, dans ce programme, on initialise la variable qui nous sera utile pour une boucle utilisée après.
int i=0;

Ensuite on récupère la valeur du score avec la fonction *evaluerScore().*
T_Score score = evaluerScore(plateau);

On ouvre ensuite le fichier json qui va nous permettre d'écrire les valeurs utile pour la page web
*FILE *fp = fopen (fic, "w+");*

On réécrit les premières lignes en y mettant à jours les valeurs, on utilise pour cela la fonction *fprintf()*

```
fprintf(fp,"%s","traiterJson({\n");  //1ère ligne
fprintf(fp,"\n  %s:%d,\n",STR_TURN,plateau.trait);
fprintf(fp,"    %s:%d,\n",STR_SCORE_J,score.nbJ);
fprintf(fp,"    %s:%d,\n",STR_SCORE_J5,score.nbJ5);
fprintf(fp,"    %s:%d,\n",STR_SCORE_R,score.nbR);
fprintf(fp,"    %s:%d,\n",STR_SCORE_R5,score.nbR5);
fprintf(fp,"    %s:[\n",STR_COLS);
```

La boucle sert à réécrire le "tableau" contenant les positions, couleurs et tailles de piles des pions du plateau

```
  for(i=0;i<NBCASES;i++) {
     fprintf(fp,"
{%s:%d,%s:%d},\n",STR_NB,plateau.cols[i].nb,STR_COULEUR,plateau.cols[i].couleur);//motif répété sur
NBCASES-1
  }
```

On indique enfin la fin du fichier et on referme le fichier

```
fprintf(fp,"%s","]\n});");
 fclose(fp);
```

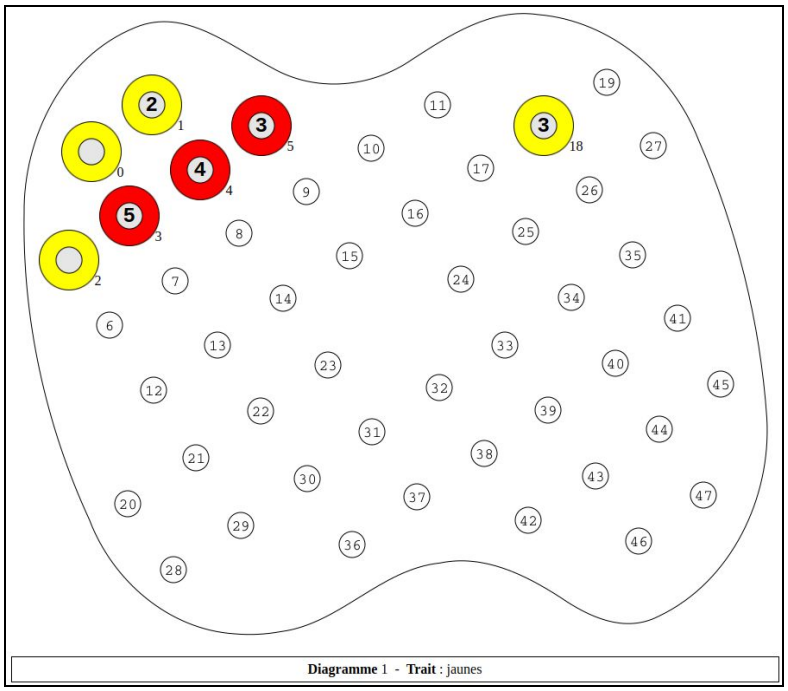
diag.exe :

1- Formatage : l'utilisateur entre une chaine de caractère, le programme doit la formater au type FEN

Le formatage se base sur plusieurs critères :

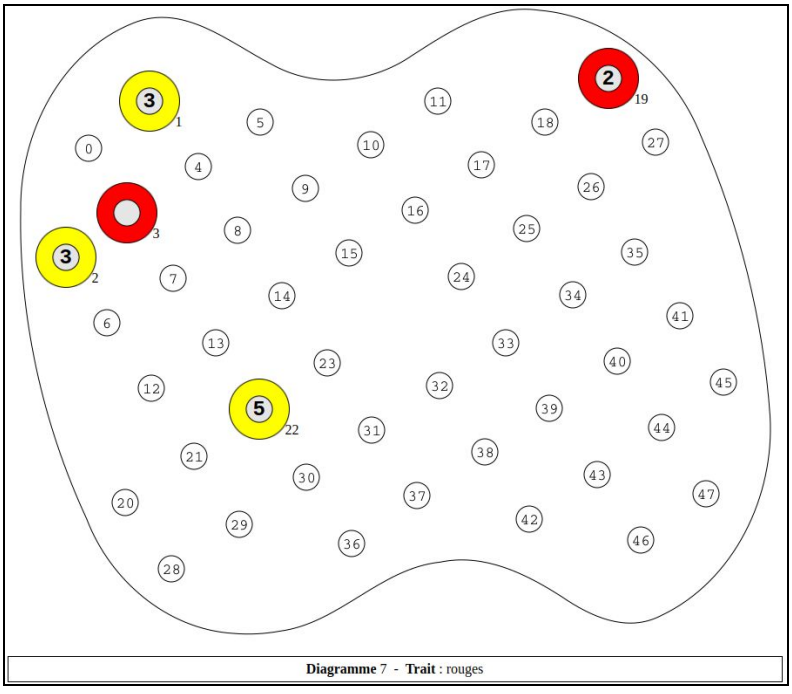
- u d t q c pour ajouter des points pour les jaunes
- U D T Q C pour les rouges
- 1 ... n pour indiquer un nombre de colonnes vides
- en suivant l'ordre des indices des cases
- Le dernier "j" pour les jaunes ou "r" pour les rouges symbolisera le trait (tour de jeu)

ex : ***uduCQT12tj*** et ***1totoU123D2rc*** donneront respectivement ces plateaux :



commande : ***./exe 1 uduCQT12tj***
Numéro du diagramme : **1**

u: 1 pion jaune
d: 2 pions jaunes
u: 1 pion jaune
C: 5 pions rouges
Q: 4 pions rouges
T: 3 pions rouges
12: 12 espaces
t: 3 pions jaunes
uduCQT12tj: Trait : jaunes



commande : ***./exe 7 1totoU123D2rc***
Numéro du diagramme : **7**

1: 1 espace
t: 3 pions jaunes
o: On ignore le "o"
t: 3 pions jaunes
o: On ignore le "o"
U: 1 pion rouge
1: 12 espaces
3: 3 espaces
D: 2 pions rouges
2: 2 espaces
c: 5 pions jaunes
1totoU123D2rc: Trait : rouges

Description de notre programme de formatage :

Nom du fichier : **diag.c**

En ouvrant ce fichier, vous pouvez déjà apercevoir des commentaires (//commentaires) expliquant les actions de notre programme. Ici nous expliquons dans les grandes lignes ce que fait notre programme.

On commence par déclarer nos variables

3 chaînes de caractères :

fen[2][50] : pour stocker ce que l'utilisateur entre en commande

FEN2[50] : pour le type FEN finale

trait[50] : pour garder le dernier "j" ou "r" en mémoire

(50 car pour nous la chaîne la plus grande fera 51 caractères avec '\0' à la fin, soit 49 caractères purs.

Notre programme fonctionne comme un d'entonnoir, il filtre de cette manière :

- On regarde si c'est un caractère valide ou un trait
- On regarde si c'est un nombre
- Sinon on ne le garde pas
- A la fin on écrit le dernier "j" ou "r" que l'on a trouvé en dernier, si rien est trouvé, la valeur par défaut est "j"

On filtre jusqu'à ce que la chaîne de l'utilisateur soit finie.

1er filtre : On regarde si c'est un caractère valide (u,d,t,q,c,U,D,T,Q,C) ou c'est un trait (j ou r)

Si c'est un caractère valide :

On passe au caractère suivant et on repasse ce test

Si c'est un trait :

On l'écrit dans une chaîne à part pour garder sa valeur et on passe au caractère suivant en repassant ce test.

Sinon :

On passe au 2nd filtre

En entrant ici on sait que la caractère n'est pas valide et n'est pas un trait.

2nd filtre : On regarde si c'est un nombre (1 ... 99)

Si c'est un chiffre (1 ... 9)

On l'écrit dans la chaîne FEN, on passe au prochain caractère et on revient au 1er filtre

Sinon

On n'écrit pas dans la chaîne FEN mais on passe aussi au prochain caractère et on revient aussi au 1er filtre

Enfin, nous finissons par entrer le trait dans la chaîne FEN

C'est-à-dire que si la chaîne trait n'est pas vide, on insère son caractère à la dernière place de la chaîne FEN, sinon on insère un "j".

Interprétation de la chaîne FEN :

Maintenant que nous avons une chaîne de caractères valide, il faut l'interpréter.

Il faut qu'un "u" soit considéré comme l'ajout d'un pion jaune, "d" comme deux pions jaunes, etc ...

Pour faire cela, on utilise une fonction : "*T_Position interpreteur(const char *FEN2);*"
Et nous associons à chaque caractère valide une action grâce à leur code ASCII

Caractère valides	**u**	**d**	**t**	**q**	**c**	**U**	**D**	**T**	**Q**	**C**	**j**	**r**
Code ASCII	117	100	116	113	99	85	68	84	81	67	106	114

Chiffre	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Code ASCII	48	49	50	51	52	53	54	55	56	57

L'interprétation passe par 3 phases :

- **L'interprétation des chiffres**
- **L'interprétation des caractères**
- **L'interprétation du trait**

Interprétation des chiffres :

Voici notre programme :

```
if((FEN2[k]-48)>=0 && (FEN2[k]-48)<10)          vérifie si chiffre: (code ASCII - code ASCII de 0)
{
    if ((FEN2[k]-48)>=0 && (FEN2[k+1]-48)<10)        vérifie si nombre à 2 chiffres
        {
            i = (((FEN2[k]-48)*10) + (FEN2[k+1]-48) + i);    1 chiffre * 10 + 2e chiffre (ex : 12 = 1 *10 + 2)
            k = k+2;                                 On regarde le caractère après le nombre
        }

        else {                                       Si ce n'est pas un nombre
            k++;                                     On regarde la prochain caractère à interpréter
            i = (FEN2[k]-48) + i;                    On place les espaces
        }
}
```

Interprétation des caractères :

Voici notre façon de faire :

```
        case 67:                                 Si c'est un 'C'
            plateau.cols[i].nb = 5;              On associe 5 au nombres de pions pour le colonne
            plateau.cols[i].couleur = ROU;       On donne la couleur rouge au pion
            break;                               On sort du case 67

        case 68://D                              Si c'est un 'D'
            plateau.cols[i].nb = 2;              2 pions pour la colonne
            plateau.cols[i].couleur = ROU;       couleur rouge
            break;                               On sort du case 68
```

```
case 81://Q                          Si c'est un 'Q'
   plateau.cols[i].nb = 4;           4 pions pour la colonne
   plateau.cols[i].couleur = ROU;    couleur rouge
   break;                            On sort du case 81
 .
 .
 .
 .
```

Nous pouvons faire ceci pour toutes les lettres valides

Interprétation du trait :

```
if(FEN2[k+1] == 'j'){                Si le dernier cara est j
    plateau.trait = 1;               alors trait jaune
  }
  else{                              Sinon, il est rouge
    plateau.trait = 2;               alors trait rouge
  }
```

Si c'est un "j" ou un "r" alors on associe à la valeur du trait 1 pour "j" et 2 pour "r"

Rafraîchir la page pour afficher le plateau :

Désormais, maintenant que nous avons formater la chaîne de l'utilisateur et que ton savons interpréter le chaîne FEN, il ne nous reste plus qu'à afficher le plateau de jeu

Pour ce faire, nous devons faire une fonction *void Refresh_diag(const char *fic,const T_Position plateau,const char *FEN,const char *num,const char *notes);* qui nous permet d'écrire dans un fichier
Cette fonction a but but de rafraîchir le tableau de jeu une fois l'interprétation faite afin de simplement voir le plateau avec ces pions

voici le prototype :

```
void Refresh_diag(const char *fic,const T_Position plateau,const char *FEN,const char *num,const char *notes);
```

```
FILE *fp = fopen (fic, "w+");          On commence l'écriture
```

Ensuite, on écrit tout le nécessaire dans le fichier d'affichage

```
fprintf(fp,"%s","traiterJson({\n");
fprintf(fp,"\n        %s:%d,\n",STR_TURN,plateau.trait);        trait
fprintf(fp,"   %s:\"%s\",\n",STR_NUMDIAG,num);                  numéro de diagramme
fprintf(fp,"   %s:\"%s\",\n",STR_NOTES,notes);                  description
fprintf(fp,"   %s:\"%s\",\n",STR_FEN,FEN);                      chaîne FEN
fprintf(fp,"   %s:[\n",STR_COLS);
      for(i=0;i<NBCASES;i++)
      {
```

```
            fprintf(fp,"
{%s:%d,%s:%d},\n",STR_NB,plateau.cols[i].nb,STR_COULEUR,plateau.cols[i].couleur);
        }
        fprintf(fp,"%s","]\n});");
        fclose(fp);
}
```

motif répété sur NBCASES-1

fin de mon fichier

On ferme

BILAN DE CETTE PREMIÈRE ÉTAPE:

Notre groupe a réussi toutes les tâches demandées, nos codes sont fonctionnels, certainement pas parfaits, mais aucun problème n'est à signaler.

Au départ nous ne pensions pas que la charge de travail était aussi grande, malgré une certaine anticipation nous avons failli être débordés vers la fin, cependant le dernier TNE nous a permis de fixer énormément de problèmes.

Grâce à ce premier livrable nous avons appris énormément de choses :

Nous avons appris à faire un fichier au format JSON (standalone.c et diag.c), nous savons désormais effectuer des redirections depuis un autre fichier (dans ce dernier nous pouvons écrire sur plusieurs lignes).

Nous savons désormais utiliser des fonctions comme fprintf et fgetc. Nous nous sommes encore plus familiariser avec les chaînes de caractères.

Enfin, nous avons appris à demander des arguments dans des lancements de fonctions.