Année 2022-2023

RAPPORT DE PROJET – LE1 A1

PROJET : Challenge Moteur de Jeux

(1er livrable)

Projet réalisé par

DEVIGNES Bastien

DUVAL Marie

RABAULT Tim

BOUKHIZZOU Ines

centralelille IG2I

1. INTRODUCTION

Avalam Evolution se joue sur un plateau hexagonal de 121 cases, avec des pions de deux couleurs différentes. L'objectif du jeu est de capturer les pions adverses en les écrasant sous d'autres pions, ou de les bloquer pour les empêcher de se déplacer. Le joueur qui capture tous les pions adverses ou qui parvient à bloquer tous les mouvements de son adversaire remporte la partie.

L'objectif assigné est de créer le jeu Avalam Evolution, pour deux joueurs, jouable au clavier en C avant la date butoir du 02/03/2022. Nous disposons d'une série de fichiers d'en-tête, notamment "avalam.h", qui contient les déclarations des types, les constantes, et les fonctions nécessaires, ainsi que "topologie.h", qui décrit la forme du plateau de jeu. Nous avons également accès au fichier "libavalam.c", qui contient des fonctions intégrées utiles pour le développement du jeu. En outre, il y a un dossier contenant des fichiers HTML/JS pour afficher le plateau de jeu sur une page web, qui est déjà fonctionnel. Pour atteindre notre objectif, nous devons écrire les fichiers "standalone.c" et "diag.c", qui permettront de jouer à Avalam Evolution sur un ordinateur. Nous commencerons par écrire "standalone.c", suivi de "diag.c".

2. Fichier « Standalone.c »

Nous avons entamé ce premier livrable en réalisant le programme "standalone.c".
Ce programme vise à permettre aux deux joueurs de placer leurs pions bonus et malus tour à tour en suivant la règle de priorité, où les bonus jaunes et rouges sont prioritaires, suivis des malus rouges et jaunes. Le programme doit garantir qu'un pion rouge ne remplace pas un pion jaune et inversement, et empêcher l'utilisateur de placer un pion malus ou bonus au-dessus d'un pion malus ou bonus déjà posé. Ensuite, le programme doit demander à chaque joueur la position vers laquelle il souhaite déplacer son pion. Un fichier JavaScript sera créé à chaque tour pour enregistrer les positions des pions. Enfin, le programme doit détecter la fin de la partie lorsqu'il n'y a plus de coup légal possible et afficher le score à l'écran.

a. Explication du code

```c
#include <stdio.h>
#include <stdlib.h>
#include <avalam.h>
#include <topologie.h>
#include <string.h>
```

Le programme fait appel, dans un premier temps, aux bibliothèques standards du langage C et des fichiers d'entêtes `<avalam.h>` et `<topologie.h>` où expliquer quelles sont ces bibliothèques.
La fonction main() est le point d'entrée du programme, elle prend deux arguments : un entier `argc` qui représente le nombre d'arguments de ligne de commande et un tableau de pointeurs de caractères `argv` qui contient ces arguments.
On déclare ensuite les variables suivantes :

- `pos` pour stocker la position du jeu
- `l` pour stocker une liste de coups légaux pour la position actuelle
- `deplacement` pour stocker un coup à jouer

- `score` pour stocker l'état du score du jeu
- `chgt` pour stocker une variable de changement.

Ainsi que deux fonctions :

- `ecritureJS()` écrit la position actuelle du jeu en JavaScript.

La fonction permet d'écrire les données de la partie en cours dans un fichier au format JSON. Elle prend en entrée la position actuelle du jeu, le score et le nom du fichier dans lequel écrire les données.

- La fonction ouvre un fichier en mode écriture et utilise la fonction `fprintf` pour écrire les données dans le fichier. Les données écrites comprennent :
- La couleur du joueur qui doit jouer (jaune ou rouge).
- Les scores des deux joueurs, ainsi que leur score en bonus (nbJ5 et nbR5) s'ils ont atteint la ligne adverse.
- Les bonus/malus de chaque joueur.
- Les informations sur chaque colonne, y compris le nombre de pions et la couleur du pion supérieur.

Enfin, la fonction ferme le fichier. Les données écrites dans ce fichier peuvent être utilisées par un programme externe pour afficher une représentation graphique de la partie en cours.

- `menu()` fonction qui affiche le menu principal du jeu.

Lorsque l'utilisateur choisit une option, une action différente est effectuée selon l'option choisie :

- Si l'utilisateur choisit l'option 1, une description du jeu Avalam (affichage en bleu).
- Si l'utilisateur choisit l'option 2, les règles du jeu (affichage en vert).
- Si l'utilisateur choisit l'option 3, des astuces pour le jeu (affichage en jaune).

- Si l'utilisateur choisit l'option 4, le lancement du jeu est annoncé (en bleu) et le programme se termine.
- Si l'utilisateur entre un choix invalide, un message d'erreur est affiché (en rouge) et il est invité à réessayer.

Le code vérifie également si l'utilisateur a fourni un nom de fichier JavaScript en argument de ligne de commande ou non. Si c'est le cas, il stocke le nom du fichier dans la chaîne `chaine_JS.` Sinon, il utilise un fichier JavaScript par défaut.
Ensuite, le programme crée la position initiale du jeu, récupère les coups légaux pour cette position et affiche le nombre de coups possibles ainsi que le trait. Il appelle la fonction `ecritureJS()` pour les écrire au format JSON afin de les transmettre à une application JavaScript.

Puis le programme demande à l'utilisateur de placer des bonus et des malus sur le plateau de jeu. Le joueur doit choisir les positions où placer les bonus et les malus.

La boucle `do while` est utilisée pour s'assurer que le joueur choisi une position valide pour chaque bonus et malus. Si le choix n'est pas valide, le programme affiche un message d'erreur et demande au joueur de choisir une autre position.
Pour chaque bonus et malus, le programme vérifie si la case choisie est de la bonne couleur et pour le malus jaune ou rouge, il s'assure que leur position n'est pas la même que celle des bonus rouge ou jaune.
Une fois que le joueur a choisi des positions valides pour tous les bonus et malus, le programme utilise la fonction `ecritureJS()` pour écrire les positions des bonus et des malus dans un fichier JavaScript.

Ensuite, la fonction passe à la boucle principale du jeu qui permet au joueur de jouer des coups jusqu'à ce qu'il n'y ait plus de coups possibles. La boucle `while` s'exécute tant qu'il y a des coups possibles, c'est-à-dire tant que le nombre de coups légaux n'est pas égal à zéro. Dans la boucle, il y a d'abord un affichage du nombre de coups possibles et

de la couleur qui doit jouer. Ensuite, le joueur doit entrer la position d'origine de la pièce à déplacer, suivie de la position de destination. Ces informations sont utilisées pour jouer le coup à l'aide de la fonction `jouerCoup()`.

Après chaque coup, la liste des coups légaux est mise à jour à l'aide de la fonction `getCoupsLegaux()` et le score est évalué à l'aide de la fonction `evaluerScore()`. Enfin, la position actuelle du jeu, le score et les informations sur les bonus et les malus sont écrites dans un fichier JavaScript à l'aide de la fonction `ecritureJS()`.
La boucle `while` continue à s'exécuter tant qu'il y a des coups possibles, et elle s'arrête lorsque le nombre de coups légaux est égal à zéro. Cela signifie que le jeu est terminé, et le score final est affiché à l'utilisateur.

```
    //affichage du score
    if(score.nbJ = score.nbR)  //isn't it == ?
    {
        if(score.nbJ5 > score.nbR5) printf("C'est le joueur jaune qui a gagné \n");
        else printf("C'est le joueur rouge qui a gagné \n");
    }
```

Enfin, le programme affiche le score final et annonce qui est le vainqueur de la partie. Pour cela, il utilise les données contenues dans la structure `score`.

- La première condition `if(score.nbJ = score.nbR)` vérifie si le nombre de jetons jaunes est égal au nombre de jetons rouges.
    - Si c'est le cas, on regarde alors le nombre de jetons de chaque couleur qui sont en jeu sur les cases centrales, avec la condition `if(score.nbJ5 > score.nbR5)` .
        - Si le joueur jaune en a plus, on affiche `"C'est le joueur jaune qui a gagné",` sinon c'est le joueur rouge qui a gagné.
- Si le nombre de jetons jaunes n'est pas égal au nombre de jetons rouges, on compare simplement les deux nombres de jetons pour déterminer le gagnant.

Enfin, le code affiche le score à l'aide de la fonction `afficherScore(score),` et renvoie la valeur 0 pour signaler que tout s'est bien déroulé.

b. Jeu de tests

Affichage du menu :

```
=== MENU ===
1. Description
2. Afficher les règles
3. Afficher les astuces
4. Jouer au jeu
Entrez votre choix : 1
```

Affichage de l'option 1 :

```
Description :

Jeu abstrait pour deux, Avalam a remporté de nombreuses récompenses.
A la fois rapide et stratégique, Avalam est un jeu de réflexion qui séduit autant par sa
mécanique et ses règles simples que par son matériel original et de qualité.
```

Affichage de l'option 2 :

```
Voici les règles du jeu :

Le principe du jeu est simple : au début de la partie, les joueurs choisissent une couleur
de pions. Cependant, chaque joueur à le droit de jouer avec les pions de son adversaire !
À leur tour, ils déplacent tous les pions placés sur une case vers une case voisine
occupée. Les pions s'empilent, formant des tours de 1 à 5 pions.
Le joueur qui parvient à placer le plus de pions de sa couleur au sommet des piles de
pions constituées au cours de la partie l emporte.
A savoir qu'une pile ne peut dépasser cinq pions, et qu'un pion ne se déplace jamais sur
un emplacement vide... Les tours ont donc des hauteurs variables, au gré de la stratégie.
Dans cette septième édition « Evolution », une variante a été ajoutée : chaque joueur
dispose de 2 pions bonus et malus qui augmentent ou diminuent la valeur de la tour. Ces
2 x 2 pions sont placés sur le plateau en début de partie.
```

Affichage de l'option 3 :

```
Voici quelques astuces pour le jeu :

-Essayez d'isoler les tours des que possible sans chercher les piles de 5 forcements,
ainsi commencer par isoler les pions au bord qui sont plus simple.
-Mettre les bonus dans le coin pour pouvoir a son tour l'isoler et marquer plus de point.s
-Pensez que les tours de 3 qui vous appartiennent sont très utiles car soit elle reste a
vous tout le long ou alors si l'adversaire décide de mettre un pion dessu, vous pouvez en
rajouter un de votre couleurs juste après pour former une tour de 5 .
-malus au milieu si l'adversaire joue les jaunes.
-mettre les pions adverse sur les malus.
```

Overview du jeu sur console :

Placement des bonus et malus :

Cas où le bonus/ malus est mal placé

```
          Veuillez choisir la position du **bonus** jaune
4


----Vous ne pouvez pas mettre ce bonus ici ----
```

Cas où le bonus/ malus est bien placé

```
          Veuillez choisir la position du **bonus** rouge
4
```

c. Conclusion

Nous avons rencontré plusieurs problèmes lors de la rédaction du fichier « standalone.c ».

Lorsqu'on exécutait dans le terminal la commande suivante « ./build/standalone.exe »

Le code ne voulait pas s'exécuter, il a fallu mettre à sa place « ./build/standalone.static »

De plus, lorsque le joueur saisissait la position des pions bonus et malus (la position étant une chaîne de caractères), le fichier JSON la traduisait en code ASCI et ne renvoyait pas le résultat voulu. On était alors contraint de tout refaire…

3. Fichier « diag.c »

Pour conclure ce premier livrable, nous avons réalisé le programme « diag.c » qui doit remplir certains critères. Tout d'abord, il doit être capable de prendre une ligne de commande en entrée afin de placer des pions à l'aide du code "FEN". Ensuite, il doit permettre à l'utilisateur de saisir le nom du fichier qu'il souhaite modifier une fois que le programme est lancé. Enfin, le programme doit générer un fichier JS qui sera utilisé pour le fichier "avalam-diag.html".

a. Explication du code

On commence d'abord par déclarer les bibliothèques suivantes :

```c
#include <ctype.h>
#include <stdio.h>
#include "../include/avalam.h"
#include <stdlib.h>
#include <string.h>
```

Elles nous servent à utiliser les fonctions intégrées du langage C, notamment avec la bibliothèque `<ctype.h>` qui fournit des fonctions pour déterminer le type de caractère d'un caractère donné.

```c
int main(int argc, char **argv)
```

Dans la fonction principale de notre programme, qui prend comme argument `int argc` e et t `char **argv`

La première partie du code vérifie si le nombre d'arguments est bien égal à 3. Si le nombre est différent, un message d'erreur s'affiche et la fonction quitte avec "return 0".
Ensuite, la longueur de la chaîne de caractères dans le deuxième argument (`argv[2]`) est stockée dans "fenlen". Cette valeur doit être entre 0 et 54 inclus, ce qui correspond à une fenêtre de 48 cases, un espace et le trait, plus 4 caractères pour les bonus. Si `"fenlen"` ne respecte pas cette contrainte, une erreur s'affiche et la fonction s'arrête. Enfin, la

variable "trait" est initialisée en fonction du dernier caractère de la chaîne de caractères dans le deuxième argument. Si le dernier caractère est "j" ou "J", "trait" prend la valeur 1. Si c'est "r" ou "R", "trait" prend la valeur 2. Si ce n'est ni "j" ni "r", une erreur s'affiche et la fonction s'arrête.

```
char nomFichier[80];
```

Cette partie de code permet à l'utilisateur de saisir le nom d'un fichier pour stocker le résultat de l'exécution du programme en format JSON.
Tout d'abord, une variable nomFichier de type char est déclarée avec une taille maximale de 80 caractères. L'utilisateur saisit ensuite le nom du fichier en utilisant la fonction fgets. La boucle `for(i=0;nomFichier[i]!='$' && nomFichier[i]!='\n' ;i++);` est utilisée pour supprimer les caractères de fin de ligne \n et de fin de chaîne $ qui peuvent être inclus dans le nom du fichier saisi par l'utilisateur.
Si l'utilisateur n'entre aucun nom de fichier, le programme utilise le nom de fichier par défaut diag.js (dans les deux cas, l'extension .js est ajoutée à la fin du nom de fichier). Le programme essaie ensuite d'ouvrir le fichier en mode écriture (w+).
Si l'ouverture échoue, un message d'erreur s'affiche et le programme se termine. Une fois le fichier ouvert, les caractéristiques connues, telles que le trait (trait), le numéro de diagnostic (numDiag) et une liste vide de notes (notes), sont ajoutées au fichier en utilisant la fonction fprintf.

```
    //Ajout de la note, nombre de lignes infini, mais 1000 caracteres par ligne car
c'est suffisant
    char ligne[1000];
    printf("Entrez votre note.\nPour arreter d'écrire, entrez une ligne vide.\n");
    do
    {
        strcpy(ligne,"");           //Utile pour l'entrée par redirection de fichier,
evite un 'echo' pour la deniere ligne
        fgets(ligne, 1000, stdin);
        if (strcmp(ligne,"\n")!=0)  //On ajoute la ligne uniquement si elle est
interessante, donc non vide.
        {
            ligne[strlen(ligne)-1]='\0';
```

```
            fprintf(fichier,"   \"%s\",\n",ligne);

        }
    }while (strcmp(ligne,"\n")!=0 && strlen(ligne)>=1); // On arrete lorsque la ligne
est "vide" ou vide.
    fprintf(fichier,"   \"\"\n],\n");//Fin du tableau, ajout d'une derniere ligne vide
pour finir sans virgule (Eviter "texte",])
    fprintf(fichier,"\"fen\":\"%s\",\n",argv[2]);//Ajout du FEN
```

Cette partie du code permet d'ajouter des notes à un fichier créé précédemment. L'utilisateur est invité à entrer des notes qui sont ajoutées dans le fichier sous forme d'une ligne, jusqu'à ce que l'utilisateur entre une ligne vide.

La boucle `do-while` est utilisée pour permettre à l'utilisateur d'entrer autant de notes qu'il le souhaite. La fonction fgets est utilisée pour lire l'entrée de l'utilisateur depuis le clavier et la condition `strcmp(ligne,"\n")!=0` permet de vérifier que l'utilisateur a bien entré une note avant de l'ajouter dans le fichier. Si ligne est vide, la boucle s'arrête. La fonction fprintf est ensuite utilisée pour ajouter la note au fichier en utilisant le format approprié pour un tableau JSON.

Ensuite, la boucle ajoute également les 4 bonus (ou malus) dans le fichier JSON en utilisant une instruction switch-case pour chaque caractère de la chaîne FEN.

```
fprintf(fichier, "\"cols\":[\n");
    int espace, k, compteur = 0, ajout;
    // espace represente le nombre de cases vides a ajouter au cas ou le programme
rencontre un nombre.
    // compteur represente le nombre de cases ajoutées. Il ne doit pas depasser 48,
sinon on arrete l'ajout de pions.
    for (i = 0; i <= fenlen; i = i + 1)
    {
        ajout = 1;
        switch (argv[2][i])
        {
        // Pour chaque lettre ayant un sens, on ajoute son equivalent. Les caracteres
n'ayant rien a faire ici seront passés.
        case 'u':
            fprintf(fichier, "  {\"nb\":1, \"couleur\":1}");
            compteur++;
            break;

        case 'd':
            fprintf(fichier, "  {\"nb\":2, \"couleur\":1}");
            compteur++;
            break;
```

```
        case 't':
            fprintf(fichier, "  {\"nb\":3, \"couleur\":1}");
            compteur++;
            break;

        case 'q':
            fprintf(fichier, "  {\"nb\":4, \"couleur\":1}");
            compteur++;
            break;

        case 'c':
            fprintf(fichier, "  {\"nb\":5, \"couleur\":1}");
            compteur++;
            break;

        case 'U':
            fprintf(fichier, "  {\"nb\":1, \"couleur\":1}");
            compteur++;
            break;

        case 'D':
            fprintf(fichier, "  {\"nb\":2, \"couleur\":1}");
            compteur++;
            break;

        case 'T':
            fprintf(fichier, "  {\"nb\":3, \"couleur\":1}");
            compteur++;
            break;

        case 'Q':
            fprintf(fichier, "  {\"nb\":4, \"couleur\":1}");
            compteur++;
            break;

        case 'C':
            fprintf(fichier, "  {\"nb\":5, \"couleur\":1}");
            compteur++;
            break;
        }
    }
```

La partie suivante du code est responsable de l'ajout des pions dans la notation FEN. Le programme parcourt la notation FEN caractère par caractère en utilisant une boucle for

et ajoute l'équivalent JSON pour chaque caractère ayant un sens en utilisant la fonction fprintf(). Un compteur est utilisé pour s'assurer que le programme n'ajoute pas plus de 48 pions et l'ajout de cases vides est géré par la variable espace.

b. Conclusion

Il n'y a pas vraiment eu de problèmes rencontrés lors de la rédaction du code, hormis quelques améliorations qui étaient nécessaires afin de d'obtenir un code plus performant.

4. Conclusion générale

En conclusion, ce projet nous a permis de mettre en pratique nos compétences en programmation et en manipulation de fichiers. Nous avons développé un programme capable de générer des fichiers au format JSON contenant des informations relatives à une partie d'échecs, à partir d'une notation FEN ou d'une liste de coups. Nous avons également implémenté des fonctionnalités permettant d'ajouter des notes et des commentaires à ces fichiers, ainsi que des bonus/malus liés aux cases de l'échiquier.

Ce projet a été très enrichissant car il nous a permis de travailler sur un sujet intéressant et complexe, nécessitant une bonne connaissance des règles du jeu d'échecs et une grande rigueur dans la gestion des données. Nous avons appris à utiliser différentes fonctions en C pour lire et écrire des fichiers, ainsi que des structures de contrôle telles que les boucles et les conditions.

Enfin, ce projet nous a également permis de travailler en équipe, de nous répartir les tâches et de communiquer efficacement pour aboutir à un résultat final cohérent et fonctionnel. Nous sommes fiers du travail accompli et nous espérons que ce programme pourra être utile aux amateurs d'échecs et aux développeurs intéressés par ce domaine.