Livrable 1 projet moteur de jeu

DUQUENNE Léo
BREVIERE Kilian
RENARD Houcine
VISEUX Emiliano

Toutes les informations détaillées sur l'utilisation du programme `diag` et 'standalone' sont disponibles sur GitHub, dans les fichiers README correspondants aux sections Standalone ou Diag". Vous pouvez consulter ces ressources à l'adresse suivante : https://github.com/LeoD-h/Avalam-Shallower-2023.git https://github.com/LeoD-h/Avalam-Shallower-2023/tree/ba2142fd689d2fae1bbc3a6d39f2414703fe5439/Avalam%20-%20Shalower
Veuillez vous référer à ces README pour des instructions spécifiques sur l'exécution du programme et d'autres détails pertinents.

Ce rapport sera divisé en quatre sections distinctes. La première partie sera consacrée aux explications du programme "standalone.c", tandis que la deuxième partie se penchera sur les explications du programme "diag.c". La troisième section présentera le bilan du travail réalisé, et enfin, la quatrième section détaillera les contributions individuelles de chaque élève dans le projet.

—-------- Standalone —---------

Objectif du programme :

L'objectif du fichier Standalone est assez simple en termes d'explications.
Ce fichier nous permet de jouer au jeu Avalam avec un plateau numérique déjà créé.
Grâce au programme, nous avons la possibilité de jouer suivant des règles de jeu différentes telles que :

- Communiquer des malus et bonus suivant les couleurs des équipes
- Connaître à chaque tour quelle équipe peut jouer
- Savoir quand le pion ne peut pas être placé pour telle ou telle raison

Explication :

Programme principal :

```
int main(int nombreArguments, char *arguments[])
{
   // Déclaration des variables de bonus et de malus pour les équipes jaune et rouge
   int bonusJaune;   // Variable du bonus jaune
   int malusJaune;   // Variable du malus jaune
   int bonusRouge;   // Variable du bonus rouge
   int malusRouge;   // Variable du malus rouge

   // Déclaration des variables pour l'origine et la destination du coup à jouer
   int origine;      // Origine du coup à jouer
   int destination;  // Destination du coup à jouer
```

```
    // Déclaration des structures de données pour la position, la liste de coups et le score
    T_Position position;
    T_ListeCoups listeCoups;
    T_Score score;

    // Initialisation de la position du jeu
    position = getPositionInitiale();

    // Génération du fichier JSON avec la position initiale et les scores, en utilisant le premier argument
    // Si la génération échoue, retourne 0 (false), sinon, continue l'exécution
    if (!genererJson(position, score, arguments[1]))
        return 0;
```

Ce programme commence par déclarer des variables pour les bonus, les malus, le premier joueur, la destination, la position du jeu, la liste des coups possibles et le score. Ensuite, il initialise la position du jeu en appelant la fonction `getPositionInitiale()`. Il génère un fichier JSON représentant l'état initial du jeu en utilisant la fonction `genererJson()`. Le programme vérifie également que la génération du fichier JSON ne renvoie pas d'erreur avant de continuer.

Determination des bonus et malus

```
  /* Saisie du bonus jaune */
do
{
    // Demande à l'utilisateur de saisir la case du bonus jaune
    printf("Saisissez la case du bonus jaune : ");
    scanf("%d", &bonusJaune);
    getchar();  // Capture le caractère de nouvelle ligne restant dans le tampon

    // Vérifie si la case sélectionnée n'a pas la couleur jaune
    if (position.cols[bonusJaune].couleur != 1)
    {
        printf("Case non conventionnelle\n");
    }
} while (position.cols[bonusJaune].couleur != 1);  // Répète tant que la case sélectionnée n'a pas la couleur jaune

// Assignation de la case sélectionnée comme bonus jaune dans la structure de position
position.evolution.bonusJ = bonusJaune;

// Génération du fichier JSON avec la nouvelle position et les scores, en utilisant le premier argument
// Si la génération échoue, retourne 0 (false), sinon, continue l'exécution
if (!genererJson(position, score, arguments[1]))
    return 0;

// Affiche la case sélectionnée comme bonus jaune
printf("Bonus Jaune : %d\n", bonusJaune);
```

Dans cette partie du code, l'utilisateur est invité à saisir la case du bonus jaune. La boucle do-while garantit que la saisie est effectuée au moins une fois et continue de demander à l'utilisateur tant que la case sélectionnée n'a pas la couleur jaune (1). Si la case n'a pas la couleur appropriée, un message est

affiché indiquant que c'est une "Case non conventionnelle". Une fois que l'utilisateur a saisi une case valide, la valeur de cette case est assignée à la variable bonusJaune et est également attribuée à l'attribut bonusJ de la structure position. Ensuite, la fonction genererJson() est appelée pour mettre à jour le fichier JSON avec la nouvelle position, le score et les coups possibles. Enfin, la valeur du bonus jaune est affichée à l'écran.

```
/* Saisie du bonus rouge */
do
{
   // Demande à l'utilisateur de saisir la case du bonus rouge
   printf("Saisissez la case du bonus rouge : ");
   scanf("%d", &bonusRouge);
   getchar();  // Capture le caractère de nouvelle ligne restant dans le tampon

   // Vérifie si la case sélectionnée n'a pas la couleur rouge
   if (position.cols[bonusRouge].couleur != 2)
   {
      printf("Case non conventionnelle\n");
   }
} while (position.cols[bonusRouge].couleur != 2);  // Répète tant que la case sélectionnée n'a pas la couleur rouge

// Assignation de la case sélectionnée comme bonus rouge dans la structure de position
position.evolution.bonusR = bonusRouge;

// Génération du fichier JSON avec la nouvelle position et les scores, en utilisant le premier argument
// Si la génération échoue, retourne 0 (false), sinon, continue l'exécution
if (!genererJson(position, score, arguments[1]))
   return 0;

// Affiche la case sélectionnée comme bonus rouge
printf("Bonus Rouge : %d\n", bonusRouge);
```

De manière similaire à la partie du bonus jaune, cette section gère le processus de saisie et de validation pour le bonus rouge. L'utilisateur est invité à saisir la case du bonus rouge, et la boucle do-while assure que la saisie est répétée tant que la case choisie n'a pas la couleur rouge (2). En cas d'erreur de saisie, un message approprié est affiché. Une fois une case valide choisie, la valeur est assignée à bonusRouge et également à l'attribut bonusR de la structure position. La fonction genererJson() est ensuite appelée pour mettre à jour le fichier JSON, et enfin, la valeur du bonus rouge est affichée à l'écran.

Ensuite, nous procéderons à l'identification des malus. Dans cette étape, l'utilisateur sera invité à spécifier les cases où il souhaite appliquer les malus, en veillant à ce que les cases choisies soient effectivement occupées par un pion de la couleur appropriée, comme indiqué dans la structure "pos". Les bonus ainsi déterminés seront ensuite affichés dans le fichier JSON.

Détermination des bonus et malus

```
/* Saisie du malus jaune */
do
{
   // Demande à l'utilisateur de saisir la case du malus jaune
   printf("Saisissez la case du malus jaune : ");
```

```
    scanf("%d", &malusJaune);
    getchar();  // Capture le caractère de nouvelle ligne restant dans le tampon

    // Vérifie si la case sélectionnée n'a pas la couleur jaune ou si elle est la même que le bonus jaune
    if (position.cols[malusJaune].couleur != 1 || (malusJaune == bonusJaune))
    {
        printf("Case non conventionnelle\n");
    }
} while (position.cols[malusJaune].couleur != 1 || (malusJaune == bonusJaune));  // Répète tant que la case sélectionnée n'a pas la couleur jaune ou est la même que le bonus jaune

// Assignation de la case sélectionnée comme malus jaune dans la structure de position
position.evolution.malusJ = malusJaune;

// Génération du fichier JSON avec la nouvelle position et les scores, en utilisant le premier argument
// Si la génération échoue, retourne 0 (false), sinon, continue l'exécution
if (!genererJson(position, score, arguments[1]))
    return 0;

// Affiche la case sélectionnée comme malus jaune
printf("Malus Jaune : %d\n", malusJaune);
```

Cette section du code gère la sélection du malus jaune en demandant à l'utilisateur de saisir la case correspondante, en veillant à sa conformité avec la couleur jaune et à son absence de concordance avec la case du bonus jaune. La boucle `do-while` assure que la saisie est répétée en cas de sélection incorrecte. Une fois une case valide choisie, la valeur est assignée à la variable `malusJaune` et à l'attribut correspondant de la structure `position`. Ensuite, la fonction `genererJson` est appelée pour mettre à jour le fichier JSON, et la valeur du malus jaune est affichée.

```
/* Saisie du malus rouge */
do
{
    // Demande à l'utilisateur de saisir la case du malus rouge
    printf("Saisissez la case du malus rouge : ");
    scanf("%d", &malusRouge);
    getchar();  // Capture le caractère de nouvelle ligne restant dans le tampon

    // Vérifie si la case sélectionnée n'a pas la couleur rouge ou si elle est la même que le bonus rouge
    if (position.cols[malusRouge].couleur != 2 || (malusRouge == bonusRouge))
    {
        printf("Case non conventionnelle\n");
    }
} while (position.cols[malusRouge].couleur != 2 || (malusRouge == bonusRouge));  // Répète tant que la case sélectionnée n'a pas la couleur rouge ou est la même que le bonus rouge

// Assignation de la case sélectionnée comme malus rouge dans la structure de position
position.evolution.malusR = malusRouge;

// Génération du fichier JSON avec la nouvelle position et les scores, en utilisant le premier argument
// Si la génération échoue, retourne 0 (false), sinon, continue l'exécution
if (!genererJson(position, score, arguments[1]))
    return 0;

// Affiche la case sélectionnée comme malus rouge
printf("Malus Rouge : %d\n", malusRouge);
```

Cette section du code gère la détermination du malus rouge en demandant à l'utilisateur de saisir la case correspondante, tout en vérifiant qu'elle est occupée par un pion rouge et qu'elle diffère de la case du bonus rouge. La boucle `do-while` garantit que la saisie est répétée en cas de sélection incorrecte. Une fois une case valide choisie, la valeur est assignée à la variable `malusRouge` et à l'attribut correspondant de la structure `position`. La fonction `genererJson` est ensuite appelée pour mettre à jour le fichier JSON, et la valeur du malus rouge est affichée à l'écran.

Boucle de Jeu: Saisie des Coups et Mise à Jour Continue

```
// Répète tant qu'il y a des coups légaux possibles
do
{
    // Affiche le joueur dont c'est le trait
    if (position.trait == 1)
        printf("Trait aux Jaunes\n");
    if (position.trait == 2)
        printf("Trait aux Rouges\n");

    // Affiche un séparateur
    printf("-------------------------------------------------\n");

    // Demande à l'utilisateur de saisir l'emplacement du pion à déplacer
    printf("Emplacement du pion à déplacer : ");
    scanf("%d", &origine);
    getchar();  // Capture le caractère de nouvelle ligne restant dans le tampon

    // Demande à l'utilisateur de saisir l'emplacement de la destination du pion
    printf("Emplacement de la destination : ");
    scanf("%d", &destination);
    getchar();  // Capture le caractère de nouvelle ligne restant dans le tampon

    // Affiche les détails du coup saisi par l'utilisateur
    printf("-------------------------------------------------\n");
    printf("Origine du coup : %d\nDestination du coup : %d\n", origine, destination);

    // Exécute la fonction qui effectue le coup et met à jour la structure de position
    position = jouerCoup(position, origine, destination);

    // Évalue le score de la nouvelle position
    score = evaluerScore(position);

    // Génère le fichier JSON avec la nouvelle position et les scores, en utilisant le premier argument
    // Si la génération échoue, retourne 0 (false), sinon, continue l'exécution
    if (!genererJson(position, score, arguments[1]))
        return 0;

    // Met à jour la liste des coups légaux possibles
    listeCoups = getCoupsLegaux(position);

} while (listeCoups.nb != 0);  // Répète tant qu'il y a des coups légaux possibles
```

Cette boucle `do-while` gère la saisie de l'origine et de la destination d'un coup, ainsi que l'exécution de la fonction `jouerCoup` qui effectue le mouvement. L'affichage indique le trait actuel, demande à l'utilisateur de spécifier les emplacements du pion à déplacer et de sa destination, puis imprime ces informations. Après chaque coup, la position du jeu est mise à jour, le score est évalué, et la fonction `genererJson` est appelée pour actualiser le fichier JSON. La boucle continue tant qu'il existe des coups légaux à jouer, déterminés par la liste des coups possibles.

Ensuite, l'utilisateur est invité à spécifier l'origine et la destination de son coup. La structure "pos" est mise à jour en considérant le coup, en appelant la fonction "jouerCoup". La fonction vérifie automatiquement la validité du coup, le cas échéant, elle modifie la structure "pos" en tenant compte du coup. Si le coup est invalide, aucune modification n'est effectuée, et l'utilisateur doit entrer un nouveau coup. Ensuite, le score est évalué après le coup, le fichier JSON est mis à jour, et de nouveaux coups légaux sont obtenus tant qu'ils existent. Si aucun coup légal n'est disponible, la boucle est quittée.

Evaluation Score Final

```
// Évalue le score final de la position actuelle
score = evaluerScore(position);

// Affiche le message indiquant le début de l'affichage du score final
printf("Score final : \n");

// Affiche le score final en détaillant les points des Jaunes et des Rouges
afficherScore(score);

// Vérifie les conditions de victoire ou d'égalité et affiche le résultat
if ((score.nbR > score.nbJ) || ((score.nbR > score.nbJ) && (score.nbR5 > score.nbJ5)))
    printf("Victoire des Rouges !!\n");
else if ((score.nbJ > score.nbR) || ((score.nbJ > score.nbR) && (score.nbJ5 > score.nbR5)))
    printf("Victoire des Jaunes !!\n");
else
    printf("Égalité !!\n");

// Affiche le message indiquant l'arrêt de l'application
printf("Arrêt de l'application\n");

// Retourne 1 pour indiquer la fin du programme
return 1;
```

En conclusion, le programme détermine le score final, l'affiche, et annonce le vainqueur ou une égalité éventuelle.

Fonction json

```
// Déclaration d'un pointeur de fichier pour le fichier JSON
FILE *fichierJson;

// Variable de boucle
```

```
int i;

// Nom du fichier JSON à générer
char nomFichier[100];

// Vérifie si un argument de nom de fichier est fourni
if (argument != NULL)
{
   // Formatage du nom du fichier avec l'argument fourni
   sprintf(nomFichier, "./%s.js", argument);

   // Ouverture du fichier JSON en écriture
   fichierJson = fopen(nomFichier, "w");
}
else
{
   // Ouverture du fichier JSON par défaut en écriture
   fichierJson = fopen("./web/data/avalam-refresh.js", "w");
}

// Vérifie si l'ouverture du fichier JSON est réussie
if (fichierJson == NULL)
{
   // Affiche un message d'erreur si l'ouverture échoue
   printf("Erreur lors de l'ouverture du fichier\n");

   // Retourne 0 pour indiquer un échec
   return 0;
}

// Ajout des éléments dans le fichier JSON
fprintf(fichierJson, "traiterJson({\n");
fprintf(fichierJson, "\"trait\" : %d,\n", position.trait);
fprintf(fichierJson, "\"scoreJ\" : %d,\n", score.nbJ);
fprintf(fichierJson, "\"scoreJ5\" : %d,\n", score.nbJ5);
fprintf(fichierJson, "\"scoreR\" : %d,\n", score.nbR);
fprintf(fichierJson, "\"scoreR5\" : %d,\n", score.nbR5);
fprintf(fichierJson, "\"bonusJ\" : %d,\n", position.evolution.bonusJ);
fprintf(fichierJson, "\"malusJ\" : %d,\n", position.evolution.malusJ);
fprintf(fichierJson, "\"bonusR\" : %d,\n", position.evolution.bonusR);
fprintf(fichierJson, "\"malusR\" : %d,\n", position.evolution.malusR);
fprintf(fichierJson, "\"cols\" : [\n");

// Parcourt les colonnes du plateau et ajoute les informations au fichier JSON
for (i = 0; i < NBCASES; i++)
{
   fprintf(fichierJson, "\t{\"nb\" : %d, \"couleur\" : %d},\n", position.cols[i].nb, position.cols[i].couleur);
}

// Ferme la section "cols" du fichier JSON
fprintf(fichierJson, "]\n");

// Termine le fichier JSON avec la fonction traiterJson
fprintf(fichierJson, "});");

// Ferme le fichier JSON
fclose(fichierJson);
```

```
// Retourne 1 pour indiquer le succès
return 1;
}
```

La fonction `genererJson` prend en paramètres la position du jeu (`T_Position`), le score (`T_Score`), et un argument pour spécifier le nom du fichier JSON à générer. Elle initialise des variables pour définir le chemin du fichier et le transforme en fonction de l'argument fourni. Si aucun argument n'est donné, le fichier est créé dans le répertoire "./web/data/" avec un nom par défaut.

Ensuite, la fonction vérifie que le fichier JSON peut être ouvert avec succès. Elle écrit ensuite dans ce fichier les informations cruciales telles que le trait, les scores, les bonus, les malus, et la position de chaque pion du jeu. Ces données sont formatées de manière à constituer du code JSON valide grâce à la fonction fprintf. Enfin, le fichier est fermé avec fclose, et la fonction renvoie 1 en cas de succès et 0 en cas d'échec.

Il est important de souligner que cette fonction est cruciale pour mettre à jour le fichier JSON qui représente l'état actuel du jeu à différentes étapes du programme principal.

Tests : Jeu d'essai

1. Placer le bonus et le malus sur le même pion

- Saisissez la case du bonus jaune : 1
- Saisissez la case du bonus rouge : 2
- Saisissez la case du malus jaune :
  - L'exécutable refuse la position et redemande la case souhaitée.

2. Placer le bonus jaune sur une case rouge

- Saisissez la case du bonus jaune : 2
  - Case non conventionnelle
  - L’exécutable refuse et demande la case souhaitée.

3. Faire le coup sur un pion qui n’est pas avoisinant

- Emplacement du pion à déplacer : 4
- Emplacement de la destination : 19

  - JouerCoup impossible : cases 4 et 19 inaccessibles!

- Emplacement du pion à déplacer :
  - L'exécutable refuse et demande la case souhaitée.

Difficultés rencontrées :

Une des principales difficultés a résidé dans la compréhension de l'ouverture et de l'écriture de fichiers. Pour surmonter ce défi, nous avons recherché des informations en ligne pour apprendre l'utilisation des fonctions fopen, fprintf, et fclose.

Un autre obstacle majeur était la compréhension des structures complexes telles que T_Position, T_Score et T_ListeCoups. Nous avons résolu cette problématique en pratiquant avec différents jeux d'essais, ce qui nous a permis de mieux maîtriser la manipulation de ces structures.

—--------- Diag —---------

Objectif du programme :

L'objectif principale du fichier diag permet la modélisation d'une section précise du jeu suivant deux arguments :

- Le numéro
- Le numéro FEN (correspondant à une suite de chiffres et de lettres indiquant une structure précise de la position des pions)

Le fichier nous permet également d'indiquer des informations types suivant un paragraphe comme indiquer les règles de jeux par exemple ou tout autre informations utiles.
Dans le livrable 2, le diag nous sera aussi d'une grande utilité pour afficher les différentes positions de pions suivant les techniques que l'on a pu trouver en jouer sur le plateau Avalam.

Explication :

Programme principal :

```
// Définition de la taille maximale d'une note
#define MAX_NOTE 1000

// Fonction principale du programme
int main(int argc, char *argv[])
{
   // Déclaration des variables
   T_Position plateau;  // Structure représentant la position du jeu
   char numDiag;        // Numéro du diagnostic
   char fen[NBCASES + 5];  // Tableau de caractères représentant la position du jeu
   char note[MAX_NOTE];  // Tableau de caractères pour stocker une note
   int i = 0, skip, bonus = 0, nbchar = 0, restant = 0;  // Variables de contrôle et compteur
   FILE *fichier;  // Pointeur de fichier pour la manipulation de fichiers
   char nomFichier[100] = "../web/data/diag.js";  // Chemin du fichier de sortie par défaut
   char caractere = 'N';  // Caractère initialisé à 'N'

   // Initialisation des malus et bonus dans la structure plateau
   plateau.evolution.malusJ = UNKNOWN;
   plateau.evolution.malusR = UNKNOWN;
   plateau.evolution.bonusJ = UNKNOWN;
   plateau.evolution.bonusR = UNKNOWN;
```

Ce programme en C génère un fichier JSON représentant l'état d'une partie de jeu. Il prend en compte les arguments de la ligne de commande pour spécifier le numéro du diagramme et la position FEN. Les données sont ensuite analysées et attribuées à une structure `plateau`. Le programme offre la possibilité de choisir un fichier de destination et prend en charge la saisie d'une note. Enfin, il écrit les informations dans un format JSON dans le fichier spécifié, avec des vérifications de validité tout au long du processus.

Vérification du nombre d'arguments

```
switch (argc)
{
case 3:
```

```
    // Si le nombre d'arguments est 3, cela signifie qu'il y a suffisamment d'informations pour traiter le diagnostic
    // Vérification de la validité de numDiag
    for (i = 0; argv[1][i] != '\0'; i++)
    {
        if (!isdigit(argv[1][i]))
        {
            printf("Le caractère '%c' à la position %d n'est pas un entier.\n", argv[1][i], i + 1);
            return 1;  // Arrêt du programme en cas d'erreur
        }
    }
    // Convertir numDiag en entier
    numDiag = atoi(argv[1]);
    // Stockage de la position du jeu (FEN)
    strcpy(fen, argv[2]);
    break;

default:
    // Si le nombre d'arguments est différent de 3, afficher un message d'erreur et quitter le programme
    printf("Usage : diag <numDiag> <fen>\n");
    return 1;  // Arrêt du programme en cas d'erreur
    break;
}
```

Cette partie du code gère les arguments de la ligne de commande. Si le programme est appelé avec trois arguments, il passe à la vérification et à la conversion du deuxième argument (`numDiag`). Un boucle parcourt chaque caractère de cet argument pour s'assurer qu'ils sont tous des chiffres. Si un caractère n'est pas un chiffre, le programme affiche un message d'erreur et se termine. Ensuite, la fonction `atoi` est utilisée pour convertir la chaîne de caractères en entier, et la valeur résultante est stockée dans la variable `numDiag`.

Le troisième argument (`argv[2]`) est ensuite copié dans la variable `fen`.

Si le nombre d'arguments n'est pas égal à trois, le programme affiche un message indiquant le format d'utilisation attendu (`diag <numDiag> <fen>`) et se termine avec un code d'erreur.

Fichier destination

```
// Demande à l'utilisateur s'il souhaite modifier le fichier de destination par défaut
printf("Le fichier de destination par défaut est : %s\nVoulez-vous le modifier ? (N,o): ", nomFichier);

// Obtient la réponse de l'utilisateur
caractere = getchar();

// Si la réponse est 'o' (oui), l'utilisateur souhaite modifier le fichier de destination
if (caractere == 'o')
{
    // Demande à l'utilisateur de renseigner le chemin vers le nouveau fichier de destination
    printf("Renseignez le chemin vers le fichier de destination : ");
    scanf("%s", nomFichier);
}
```

Cette section du code gère la sélection ou la modification du fichier de destination où les résultats seront sauvegardés.

Tout d'abord, le programme affiche le chemin du fichier de destination par défaut en utilisant la variable `nomFichier` dans le message. Ensuite, il demande à l'utilisateur s'il souhaite modifier le fichier de destination en entrant 'o' (pour oui) ou 'N' (pour non).

Si l'utilisateur choisit de modifier le fichier de destination en saisissant 'o', le programme affiche un message demandant à l'utilisateur de spécifier le nouveau chemin du fichier de destination. Ensuite, le programme attend que l'utilisateur entre le nouveau chemin, puis il le stocke dans la variable `nomFichier` à l'aide de la fonction `scanf`.

Initialisation des Colonnes du Plateau depuis la Chaîne FEN

```
// Initialise l'index à 0
i = 0;

// Tant que le caractère actuel de la chaîne 'fen' n'est pas un espace (' ')
while (fen[i] != ' ')
{
   // Utilise une instruction switch pour traiter le caractère actuel de 'fen'
   switch (fen[i])
   {
   // Cas pour les pièces de couleur jaune
   case 'u':
      plateau.cols[i - bonus + restant].couleur = JAU;
      plateau.cols[i - bonus + restant].nb = 1;
      break;
   case 'd':
      plateau.cols[i - bonus + restant].couleur = JAU;
      plateau.cols[i - bonus + restant].nb = 2;
      break;
   case 't':
      plateau.cols[i - bonus + restant].couleur = JAU;
      plateau.cols[i - bonus + restant].nb = 3;
      break;
   case 'q':
      plateau.cols[i - bonus + restant].couleur = JAU;
      plateau.cols[i - bonus + restant].nb = 4;
      break;
   case 'c':
      plateau.cols[i - bonus + restant].couleur = JAU;
      plateau.cols[i - bonus + restant].nb = 5;
      break;
   // Ajoutez d'autres cas si nécessaire pour d'autres caractères
   }
   // Incrémente l'index pour passer au caractère suivant dans 'fen'
   i++;
}
```

Partie pour les Pions Jaunes :

Cette section du code parcourt la chaîne de caractères `fen` jusqu'à rencontrer un espace. Pendant cette itération, elle examine chaque caractère de la FEN représentant la disposition du jeu. Si le caractère correspond à un pion jaune, il est traité en conséquence.

- Pour le pion "u", le programme attribue la couleur jaune (`JAU`) à la colonne et lui donne la valeur 1.
- Pour le pion "d", la couleur jaune et la valeur 2 sont attribuées à la colonne.
- De manière similaire, les caractères "t", "q", et "c" représentent différents pions jaunes avec des valeurs respectives de 3, 4, et 5.

Traitement des Pièces de Couleur Rouge dans la Chaîne FEN

```
// Cas pour les pièces de couleur rouge
case 'U':
   plateau.cols[i - bonus + restant].couleur = ROU;
   plateau.cols[i - bonus + restant].nb = 1;
   break;
case 'D':
   plateau.cols[i - bonus + restant].couleur = ROU;
   plateau.cols[i - bonus + restant].nb = 2;
   break;
case 'T':
   plateau.cols[i - bonus + restant].couleur = ROU;
   plateau.cols[i - bonus + restant].nb = 3;
   break;
case 'Q':
   plateau.cols[i - bonus + restant].couleur = ROU;
   plateau.cols[i - bonus + restant].nb = 4;
   break;
case 'C':
   plateau.cols[i - bonus + restant].couleur = ROU;
   plateau.cols[i - bonus + restant].nb = 5;
   break;
// Ajoutez d'autres cas si nécessaire pour d'autres caractères
```

Partie pour les Pions Rouges :

Dans le même bloc de code, la section destinée aux pions rouges est traitée. Elle utilise des caractères majuscules correspondant aux pions rouges dans la FEN.

- "U" représente un pion rouge avec une valeur de 1.
- "D" représente un pion rouge avec une valeur de 2.
- De manière similaire, les caractères "T", "Q", et "C" représentent différents pions rouges avec des valeurs respectives de 3, 4, et 5.

Ainsi, cette partie du code met à jour les propriétés de chaque colonne du plateau en fonction des pions jaunes et rouges rencontrés dans la FEN, tout en ajustant les indices en fonction des éventuels malus ou bonus.

```
// Cas pour les malus
case 'm':
   bonus++;
   // Vérifie si le malus jaune est déjà défini
   if (plateau.evolution.malusJ == UNKNOWN)
      plateau.evolution.malusJ = i - bonus;
   else
   {
      printf("Il y a trop de malus/bonus !\n");
      return 1;
   }
```

```
    break;
case 'M':
    bonus++;
    // Vérifie si le malus rouge est déjà défini
    if (plateau.evolution.malusR == UNKNOWN)
        plateau.evolution.malusR = i - bonus;
    else
    {
        printf("Il y a trop de malus/bonus !\n");
        return 1;
    }
    break;
```

Partie pour les Malus :
Cette section du code traite les caractères 'm' et 'M' dans la FEN, qui représentent respectivement les malus jaunes et rouges. Lorsqu'un malus est détecté, le compteur de bonus est incrémenté, et la position du malus est enregistrée dans la structure du plateau. Si la position du malus correspondant n'est pas encore définie (indiquée par `UNKNOWN`), elle est attribuée. Cependant, si une position est déjà attribuée, cela signifie qu'il y a trop de malus/bonus, et le programme affiche un message d'erreur avant de se terminer.

```
// Cas pour les bonus
case 'b':
    bonus++;
    // Vérifie si le bonus jaune est déjà défini
    if (plateau.evolution.bonusJ == UNKNOWN)
        plateau.evolution.bonusJ = i - bonus;
    else
    {
        printf("Il y a trop de malus/bonus !\n");
        return 1;
    }
    break;
case 'B':
    bonus++;
    // Vérifie si le bonus rouge est déjà défini
    if (plateau.evolution.bonusR == UNKNOWN)
        plateau.evolution.bonusR = i - bonus;
    else
    {
        printf("Il y a trop de malus/bonus !\n");
        return 1;
    }
    break;
```

Partie pour les Bonus :
De manière similaire, cette partie gère les caractères 'b' et 'B', représentant respectivement les bonus jaunes et rouges. Le processus est identique à celui des malus, avec l'incrémentation du compteur de bonus et l'enregistrement de la position du bonus dans la structure du plateau. Si une position de bonus correspondant est déjà définie, le programme affiche un message d'erreur et se termine. L'objectif de cette section est d'assurer qu'il n'y a pas de conflits dans la définition des malus et des bonus.

```
// Cases à sauter
default:
   if (isdigit(fen[i])) // Vérifie si le caractère est un chiffre
   {
      if (isdigit(fen[i + 1])) // Vérifie si le caractère suivant est un chiffre
      {
         // On saute le nombre de cases indiqué par les deux chiffres
         skip = (fen[i] - '0') * 10 + (fen[i + 1] - '0') - 2;
         i++;
      }
      else
         // Si le caractère suivant n'est pas un chiffre, saute une seule case
         skip = fen[i] - '0' - 1;

      // Ajoute le nombre de cases à 'restant'
      restant += skip;
   }
   else
   {
      // Si le caractère n'est ni un chiffre ni une pièce, affiche une erreur
      printf("Le caractère '%c' à la position %d n'est pas valide !\n", fen[i], i);
      return 1; // Retourne une erreur
   }
   break;
}

// Affiche le caractère et la valeur de 'restant' pour déboguer
printf("FEN[i] = %c\n", fen[i]);
printf("restant=%d\n", restant);
i++; // Passe au caractère suivant dans la FEN
```

Cette section du code traite la FEN, une représentation compacte de l'état d'un jeu d'Avalam. Le code analyse chaque caractère de la FEN pour déterminer la couleur et la quantité de pions dans chaque colonne du plateau de jeu. Les caractères minuscules représentent les pions jaunes, tandis que les majuscules représentent les pions rouges.

Lors de la lecture de la FEN, le code effectue différentes actions en fonction des caractères rencontrés :
1. Pour les pions jaunes ('u', 'd', 't', 'q', 'c') ou rouges ('U', 'D', 'T', 'Q', 'C'), il attribue la couleur et la quantité correspondantes à la colonne en cours de lecture.
2. En cas de malus ('m' ou 'M') ou de bonus ('b' ou 'B'), il ajuste la position du malus ou du bonus sur le plateau.
3. Si le caractère est un chiffre, cela signifie qu'il y a des cases à sauter. Le code détermine le nombre de cases à sauter en convertissant les chiffres en entiers et ajuste la position en conséquence.

La boucle se termine en déterminant le trait du joueur actuel ('r' pour rouge, 'j' pour jaune). Ce processus est essentiel pour initialiser correctement la position de jeu à partir de la FEN fournie, garantissant une représentation précise de l'état du jeu d'Avalam.

```
// Vérification longueur FEN
fen[strlen(fen) - 2] = '\0'; // Supprime les deux derniers caractères ('\r' et '\n' généralement)
```

```
if (strlen(fen) > NBCASES + bonus + restant)
{
   printf("\nLe FEN renseigné est trop grand !\n"); // Affiche un message d'erreur si la FEN est trop longue
   return 1; // Retourne une erreur
}
```

Cette partie du code vérifie la validité de la longueur de la chaîne de caractères FEN (Forsyth-Edwards Notation), qui représente l'état d'un jeu d'Avalam de manière compacte.

Tout d'abord, le code ajuste la longueur de la chaîne FEN en tronquant les deux derniers caractères. Cela est fait en plaçant un caractère nul ('\0') à la position avant les deux derniers caractères de la chaîne. Cette manipulation est réalisée pour ignorer les informations additionnelles potentielles après la FEN dans la chaîne.

Ensuite, le code compare la longueur ajustée de la FEN avec la taille attendue, qui dépend du nombre total de colonnes du plateau (NBCASES), du nombre de cases à sauter (bonus), et de la position restante. Si la FEN est plus longue que prévu, le code affiche un message d'erreur indiquant que la FEN renseignée est trop grande, et la fonction renvoie 1 pour signaler une erreur. Cette vérification est cruciale pour s'assurer que la FEN fournie respecte la structure attendue et éviter des problèmes potentiels liés à une longueur incorrecte.

```
// DESCRIPTION
printf("Renseignez une description (max. %d caractères) :\n", MAX_NOTE); // Affiche un message pour demander une description
while ((caractere = getchar()) != EOF)
{
   if (caractere != '\n')
      note[nbchar] = caractere; // Stocke chaque caractère dans la chaîne de caractères "note"
   else
      note[nbchar] = ' '; // Remplace les retours à la ligne par des espaces dans la chaîne de caractères "note"
   nbchar++;
}
note[nbchar] = '\0'; // Ajoute le caractère de fin de chaîne à "note"
if (nbchar >= MAX_NOTE)
{
   printf("\nCette description est trop grande !\n"); // Affiche un message d'erreur si la description est trop longue
   return 1; // Retourne une erreur
}
```

Cette section du code permet à l'utilisateur de saisir une description pour la position du jeu, avec une limite maximale de caractères définie par la constante MAX_NOTE. Le programme affiche un message invitant l'utilisateur à entrer une description et utilise une boucle while pour lire chaque caractère de l'entrée standard (clavier) jusqu'à ce que la fin de fichier (EOF) soit atteinte.

Pour chaque caractère lu, le code vérifie s'il s'agit d'un retour à la ligne ('\n'). Si ce n'est pas le cas, le caractère est ajouté à la chaîne de caractères "note". Si le caractère est un retour à la ligne, un espace

est ajouté à la place, permettant de conserver une structure lisible pour la description. Le compteur "nbchar" est incrémenté à chaque itération.

Une fois la saisie terminée, le code ajoute un caractère nul ('\0') à la fin de la chaîne "note" pour la terminer correctement. Ensuite, le code vérifie si le nombre de caractères saisis dépasse la limite définie par MAX_NOTE. Si c'est le cas, le programme affiche un message d'erreur signalant que la description est trop grande, et la fonction renvoie 1 pour indiquer une erreur. Cette procédure garantit que la description ne dépasse pas la taille maximale autorisée.

```
// LOGS
fichier = fopen(nomFichier, "w+"); // Ouvre le fichier de destination en mode écriture
if (fichier == NULL)
    perror("\nLe fichier de destination est introuvable !\n"); // Affiche un message d'erreur si le fichier n'est pas accessible

fprintf(fichier, "traiterJson({\n%s:%d,\n", STR_TURN, plateau.trait); // Écrit le trait dans le fichier JSON
fprintf(fichier, "%s:%d,\n", STR_NUMDIAG, numDiag); // Écrit le numéro de diagnostic dans le fichier JSON
fprintf(fichier, "%s:\"%s\",\n", STR_NOTES, note); // Écrit la description dans le fichier JSON
fprintf(fichier, "%s:\"%s\",\n", STR_FEN, fen); // Écrit la FEN dans le fichier JSON
fprintf(fichier, "%s:%d,\n", STR_BONUS_J, plateau.evolution.bonusJ); // Écrit le bonus Jaune dans le fichier JSON
fprintf(fichier, "%s:%d,\n", STR_MALUS_J, plateau.evolution.malusJ); // Écrit le malus Jaune dans le fichier JSON
fprintf(fichier, "%s:%d,\n", STR_BONUS_R, plateau.evolution.bonusR); // Écrit le bonus Rouge dans le fichier JSON
fprintf(fichier, "%s:%d,\n", STR_MALUS_R, plateau.evolution.malusR); // Écrit le malus Rouge dans le fichier JSON
fprintf(fichier, "%s:[\n", STR_COLS); // Débute l'écriture des colonnes dans le fichier JSON

for (i = 0; i < NBCASES; i++)
{
   if (plateau.cols[i].nb > 0 && plateau.cols[i].nb < 6 && plateau.cols[i].couleur == ROU || plateau.cols[i].couleur == JAU)
                fprintf(fichier, "\t{%s:%hhd, %s:%hhd},\n", STR_NB, plateau.cols[i].nb, STR_COULEUR, plateau.cols[i].couleur); // Écrit les détails de la colonne dans le fichier JSON
   else
       fprintf(fichier, "\t{%s:0, %s:0},\n", STR_NB, STR_COULEUR); // Écrit les détails d'une colonne vide dans le fichier JSON
}

fputs("]\n});", fichier); // Termine l'écriture du fichier JSON
fclose(fichier); // Ferme le fichier
printf("\nRéussite !\n"); // Affiche un message de réussite
return 0; // Retourne 0 pour indiquer le succès
}
```

Cette partie du code gère l'écriture des informations sur la position du jeu dans un fichier de destination au format JSON. Voici une description détaillée :

1. Ouverture du fichier : Le code utilise la fonction `fopen` pour ouvrir un fichier en mode écriture (`"w+"`). Si l'ouverture échoue (le fichier n'est pas accessible ou n'existe pas), le programme affiche un message d'erreur avec `perror` et renvoie 1 pour indiquer une erreur.

2. Écriture des données : Ensuite, le code utilise la fonction `fprintf` pour écrire les différentes informations formatées dans le fichier. Ces informations comprennent le trait, le numéro de diagramme, la description, la notation FEN, les bonus, malus et l'état de chaque colonne du jeu. Le formatage est effectué en respectant la syntaxe JSON.

3. Boucle sur les colonnes : Le code utilise une boucle `for` pour parcourir chaque colonne du jeu. Pour chaque colonne, il écrit les informations sur le nombre (`nb`) et la couleur dans le fichier. Si le nombre est entre 1 et 5 et la couleur est rouge ou jaune, les valeurs sont écrites. Sinon, des valeurs par défaut sont utilisées.

4. Fermeture du fichier : Une fois toutes les informations écrites, le fichier est fermé avec `fclose`.

5. Affichage de la réussite : Si tout se déroule correctement, le programme affiche un message de réussite et retourne 0 pour indiquer une exécution sans erreur.

Tests : Jeu d’essais

1re Test : Consiste à vérifier si le programme se lance correctement.

Exécutable :

./diag/diag.static 7 A1vC4qT3q
Réponses attendues de l'invite de commande :
Saisissez le nom du fichier à produire : test_programme1
Saisissez le message à produire (taille maximale de 10000 caractères et 10000) : Test du programme 1
Réponse du programme :
Le FEN contient des caractères non lisibles, le programme va les ignorer.
Dépassement du nombre de cases, FEN impossible, arrêt du programme.

2e Test : Placer le bonus et le malus sur le même pion

Exécutable :
./diag/diag.static 4 RmD3U2b
Réponses attendues de l'invite de commande :
Saisissez le nom du fichier à produire : test_programme2
Saisissez le message à produire (taille maximale de 10000 caractères et 10000) : Placer le bonus et le malus sur le même pion

Réponse du programme :
Bonus Rouge : 3
Malus Rouge : 2
Le FEN contient des caractères non lisibles, le programme va les ignorer.
Le FEN contient des caractères non lisibles, le programme va les ignorer.

Dépassement du nombre de cases, FEN impossible, arrêt du programme.

Problèmes rencontrés :
La principale difficulté rencontrée dans cette section réside dans la gestion de la description, permettant la saisie de plusieurs lignes. Nous avons surmonté cette difficulté en recherchant des solutions en ligne.

Rétrospective des réalisations :
Les deux programmes sont fonctionnels. Un problème est associé au fichier JSON du diagnostic : il apparaît dans le mauvais répertoire et ne subit pas les modifications du script, ni la création d'un fichier JS. Il devient ainsi plus complexe d'insérer l'adresse sur le site internet Diag-Avalam.

Travail par personnes :
BREVIERE Kilian Standalone : Conception du programme Standalone + Aide sur le diag sur le Fen + CR + EXCEL/ (8H30)
DUQUENNE Léo Diag : Conception du programme Diag + Aide sur le json + bonus + Prototype + CR + EXCEL + GIT / (9H)
RENARD HOUCINE : Recherche du Standalone : 3H
VISEUX Emiliano : Recherche du Diag : 3H

Conclusion :
Au terme de cette étape de développement, nous sommes parvenus à la réalisation fructueuse des différents programmes requis (Standalone / Diag), marquant ainsi une avancée significative dans notre apprentissage des fonctionnalités du langage C. Cette expérience nous a permis d'acquérir une maîtrise approfondie de diverses fonctions telles que fopen, fprintf, fclose, strcat, getline, et d'explorer efficacement l'utilisation des structures. Ce projet a constitué une opportunité inestimable pour approfondir nos compétences en programmation, nous confrontant à de nouveaux défis et élargissant notre compréhension du langage C. Nous sommes fiers des connaissances acquises et confiants dans notre capacité à relever des défis similaires à l'avenir grâce à cette expérience enrichissante.