FLIPE

LES MEMBRES DU PROJET :

- BENALI TAHA YASINE ISMAIL
- ILYES BOUAZIZ
- KANA ABDELHAK
- SAHRAOUI ABIR

HTTPS://WWW.CANVA.COM/DESIGN/DAFYJTVIQ9A/AC2ISYIRBTJPHYVJULA-KA/VIEW?UTM_CONTENT=DAFYJTVIQ9A&UTM_CAMPAIGN=DESIGNSHARE&UTM_MEDIUM=LINK&UTM_SOURCE=PUBLISHSHARELINK

FLIPE GAME

SOMMAIRE :

INTRODUCTION

- Au départ, nous avons commencé par le côté algorithmique en préparant un algorithme efficace pour que le jeu fonctionne à 100% en C avant de l'intégrer à SDL. Ensuite, nous avons travaillé sur SDL pour remplacer les entrées SCANF par des interactions à la souris et au clavier, ainsi que les sorties PRINTF par des affichages d'images et de textes (ce n'est pas exactement cela qui s'est passé, mais cela donne une idée claire) . Nous avons également ajouté des fonctionnalités supplémentaires telles que le timer, la sauvegarde, la pause et la possibilité d'enregistrer les neauvou scores. Ces points seront détaillés dans les pages suivantes.

- Enfin, nous avons travaillé sur les animations de transition entre les fenêtres, qui seront expliquées plus en détail ultérieurement. Nous avons également terminé en intégrant l'audio de tout le jeu, incluant les sons de clic, les animations et les sons de victoire...

REMARQUE :

- Dans ce document, nous allons seulement aborder les aspects les plus pertinents et importants du projet qui méritent d'être partagés.

CÔTÉ ALGORITHMIQUE

- Cette partie va être divisée en 3 sections :
- cas de la machine
- cas du joueur
- cas de confrontation entre joueur et machine.

CAS MACHINE :

- Au départ, notre principal souci était de comprendre comment étudier la machine à jouer tous seule. (Cela signifie que si vous lui donnez le "mot début", le "mot fin" et la "taille de mot" et le "nombre de lettres qu'elle peut inverser à chaque fois", elle vous donnera la combinaison du "mot début" vers le "mot fin"). Après réflexion, nous avons découvert qu'il fallait, à partir de ce mot début, créer tous les cas possibles. Cela nous permet, d'une part, de savoir si cette combinaison est possible et, d'autre part, de pouvoir extraire cette combinaison. Nous avons donc dû choisir un type de structure (liste chainée, tableau, arbre, graphe) et finalement nous avons utilisé les listes (voir figure 1).

```
typedef struct cel *liste;

typedef struct cel{

  int vect[10];

  liste svt;

}ne;
```

VECT 3 2 1

SVT

FIGURE 1

- Pour coder le mot, nous avons transformé celui-ci en un tableau d'entiers. Chaque lettre est codifiée par un entier (voir la figure 2).

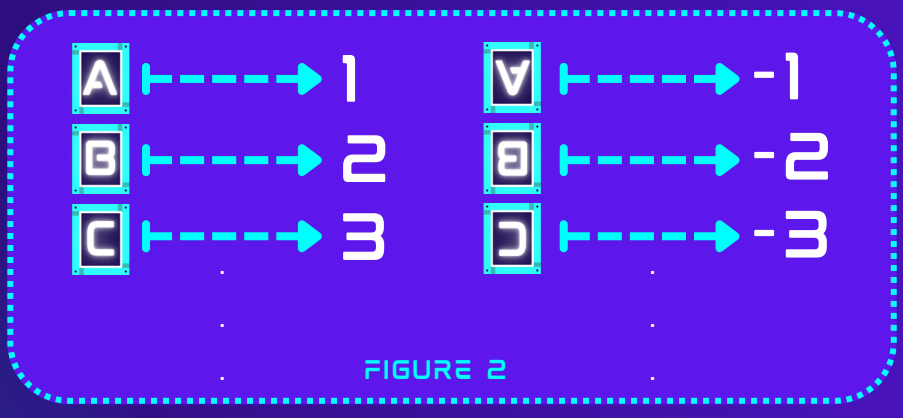

FIGURE 2

- Pour créer cette liste de tous les cas possibles, nous avons pris le mot début (par exemple "abc" avec le nombre de lettres inversées = 2, voir la figure 3). Nous avons inversé les deux premières lettres "ab" et nous avons vérifié si nous étions tombés sur un cas déjà trouvé. Si ce n'est pas le cas, nous l'avons ajouté à la liste. Nous avons fait la même chose avec les deux dernières lettres "bc",

- puis nous avons avancé le pointeur sur le deuxième élément et nous avons répété le processus: inversion des deux premières, des deux dernières, vérification, ajout. Nous avons continué jusqu'à ce que, à chaque fois que nous inversions et vérifiions, nous tombions sur un cas déjà trouvé. Nous avons alors avancé le pointeur jusqu'à ce qu'il arrive à la fin de la liste (pointeur = NULL), Si vous le souhaitez, vous pouvez essayer cette fonction par vous-même en exécutant le programme "Projet sans SDL v1.0" qui est inclus avec les fichiers du projet.

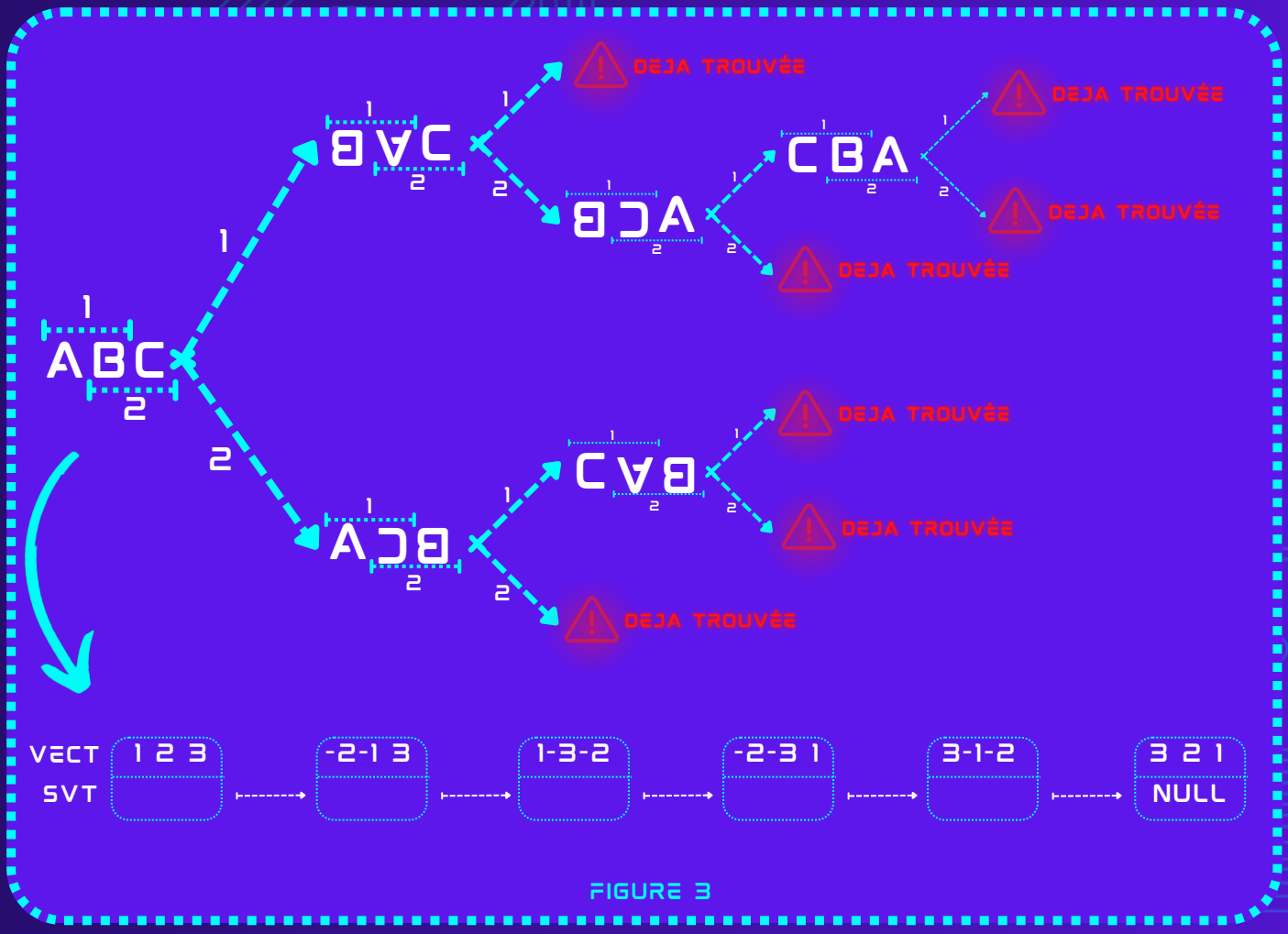

FIGURE 3

- Maintenant, si vous donnez à la machine le mot de début et de fin et la taille, elle peut créer tous les cas possibles mais elle ne peut pas trouver la combinaison du "mot début" vers le "mot fin" parce qu'elle ne peut pas simplement nous donner la liste du premier élément jusqu'au mot de fin. Il y aura des éléments en plus (par exemple, le mot de début est "abc" et le mot de fin est "bca" et le nombre de lettres inversées est 2, voir la figure 4).

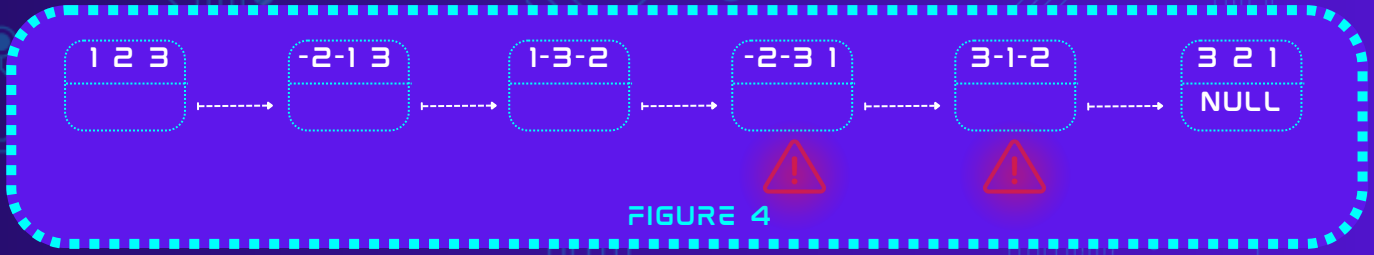

FIGURE 4

- Donc, ce que nous avons fait est d'ajouter un troisième champ "line" dans lequel nous enregistrons comment chaque élément de la liste a été trouvé. Par exemple, la line=121 signifie qu'il faut prendre le mot de départ, inverser les deux premières lettres, puis les deux secondes, puis les deux premières pour arriver à ce mot. En conséquence, l'enregistrement change (voir la figure 5) et ainsi la liste de tous les cas possible change également (voir la figure 6).

```
typedef struct cel *liste;

typedef struct cel{

  int vect[10];

  liste svt;

  int line;

}ne;
```

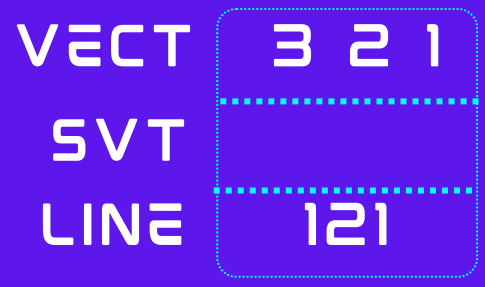

FIGURE 5

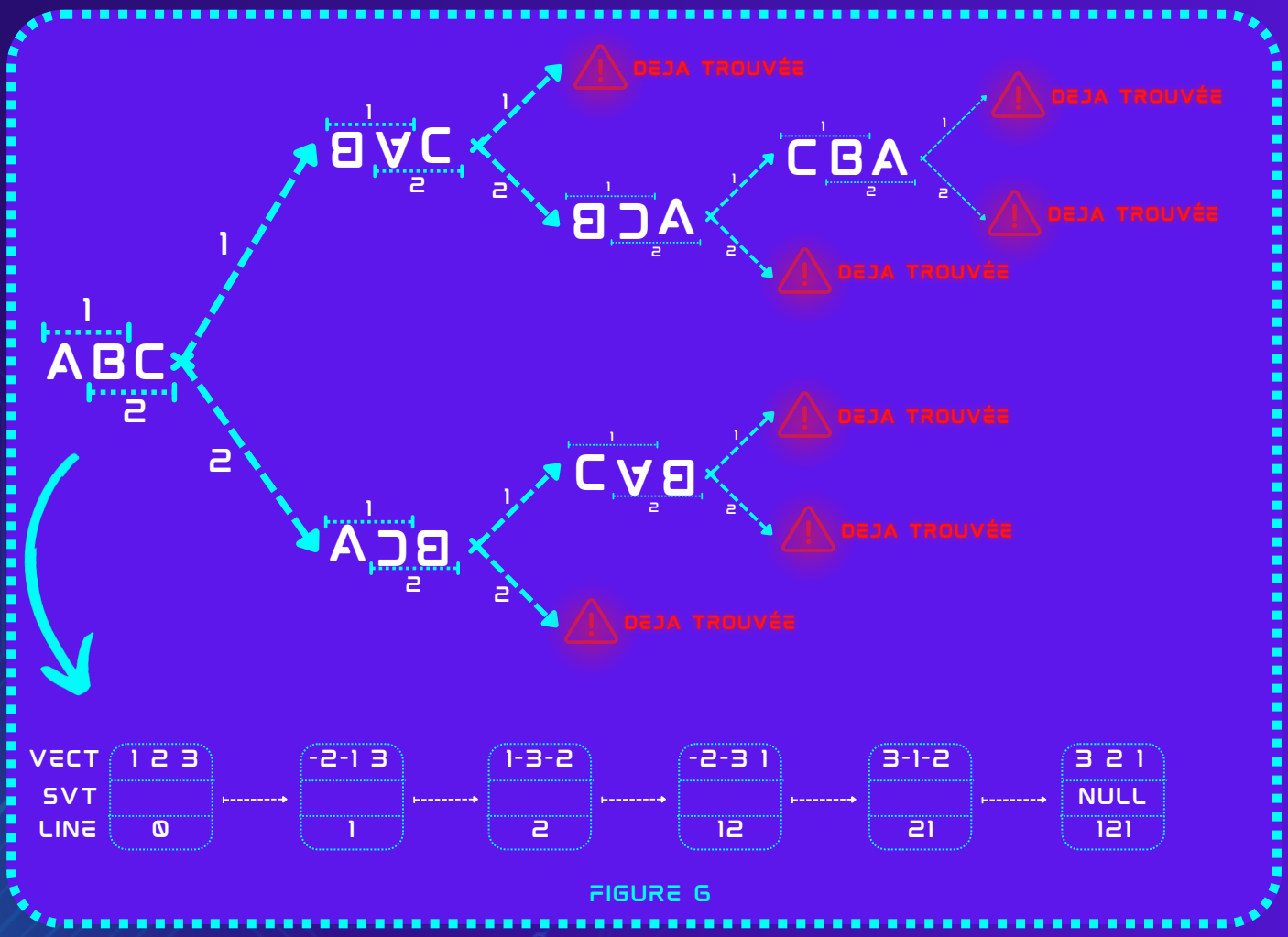

FIGURE 6

- Maintenant, en donnant à la machine un mot de départ et de fin ainsi que la taille, elle peut générer toutes les combinaisons possibles et trouver la séquence allant du "mot de départ" au "mot final" en utilisant la liste (comme dans l'exemple précédent, elle trouvera la séquence allant de "abc" à "cba" avec une line = 121, voir figure 7). Elle affichera ensuite cette séquence une par une avec une pause de 2 secondes entre chaque affichage. En ce qui concerne l'affichage de chaque tableau pour chaque étape, le caractère correspondant est affiché (voir figure 8). Vous pouvez également essayer cette fonction vous-même en exécutant le programme "Projet sans SDL v1.1" inclus avec les fichiers du projet.

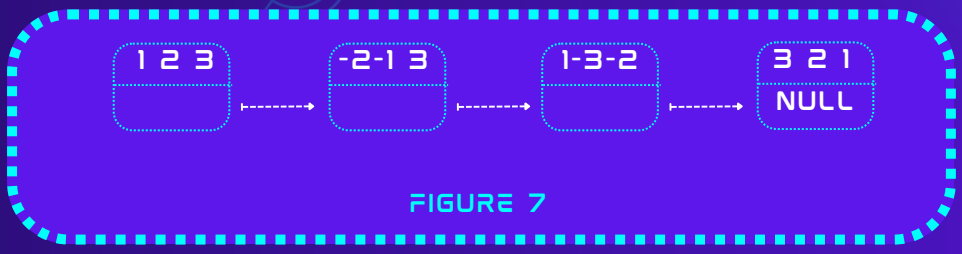

FIGURE 7

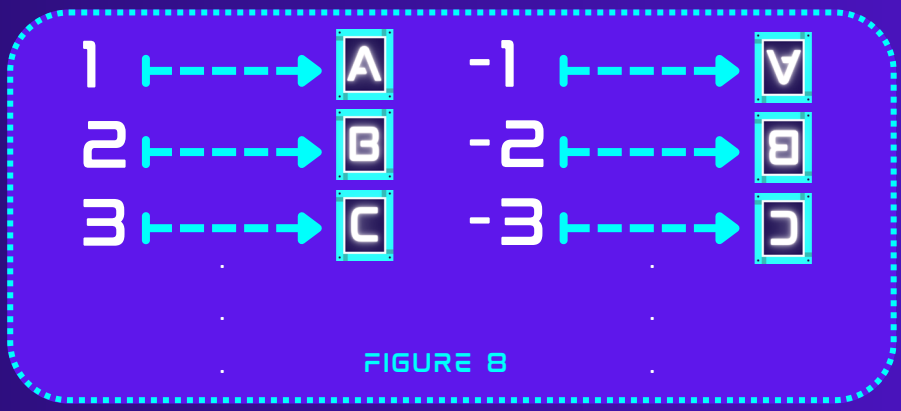

FIGURE 8

CAS PLAYER :

- Dans ce cas, le jeu sélectionne une taille de mot aléatoire entre 3 et 8, et un nombre de lettres inversées compris entre (taille / 2) et (taille - 1). Il génère ensuite un mot de départ aléatoire (composé de lettres normales et inversées sans répétition) et élabore la liste de tous les cas possibles à partir de ce mot en utilisant la méthode décrite précédemment. Il choisit ensuite un mot final aléatoire parmi cette liste. Lorsque le joueur inverse des lettres, le jeu vérifie s'il a atteint le mot final ou non, jusqu'à ce qu'un des deux résultats soit obtenu : soit il gagne, soit il atteint le nombre maximum d'inversions autorisé (qui est égal au nombre d'inversions minimal pour gagner multiplié par 2, ce qui permet au joueur de commettre des erreurs).

CAS PLAYER VS MACHINE :

- Dans ce cas, le jeu combine les deux premiers cas en choisissant une taille de mot aléatoire entre 3 et 8, ainsi qu'un nombre de lettres inversées entre (taille / 2) et (taille - 1). Il crée ensuite un mot de départ aléatoire (qui contient des lettres normales et inversées sans répétition) et génère la liste de tous les cas possibles de ce mot. D'une part, la machine affiche la combinaison du "mot début" vers le "mot fin" en utilisant la méthode expliquée précédemment, en affichant chaque élément de la combinaison avec un délai de 2 secondes entre chaque affichage. D'autre part, lorsque le joueur inverse une lettre, le jeu vérifie s'il est tombé sur le mot final ou non, jusqu'à ce qu'il gagne, ou bien atteigne le nombre maximum d'inversions autorisé, ou bien la machine gagne avant lui. Il est clair que si l'un des deux (machine ou joueur) termine (gagne ou perd), l'autre ne peut pas continuer à jouer et le temps sera arrêté.
- Vous avez la possibilité de tester le jeu fonctionnel à 100% en C avant de l'intégrer à SDL en exécutant le programme "projet sans sdl v1.6" inclus dans les fichiers de projet.

L'ANIMATION

- Au début de la conception du jeu, nous avons prévu des transitions entre chaque fenêtre de jeu, mais en étudiant SDL, nous avons constaté qu'il ne prend pas en charge l'affichage de fichiers vidéo (tels que mp4). Il ne peut afficher que des images et du texte, pas des vidéos. Nous avons donc utilisé Adobe Premiere Pro pour diviser les vidéos en images avec une fréquence de 30 images par seconde. En les affichant à un rythme approprié, nous pouvons réussi à simuler l'effet d'une vidéo. ,nous avons mis un délai de (1000 ms /30) entre chaque deux images (voir la figure 8). Dans le menu, le joueur peut activer et désactiver ces animations en utilisant la touche "espace" du clavier. L'activation et la désactivation des animations sont indiquées par un effet sonore.

```
for (int i = first_image; i <=last_image;i++){

affiche_image(i);SDL_Delay(1000/30);

        }
```

LE CODE A ÉTÉ SIMPLIFIÉ AFIN DE FACILITER LA COMPRÉHENSION DE L'ALGORITHME

FIGURE 8

LES CARACTÈRES DE MENU

- Le menu de jeu présente un effet visuel dynamique en affichant des caractères aléatoires de manière séquentielle. Ces caractères peuvent être normaux ou inversés, avec une taille et une position aléatoires sur l'écran. Ce processus est interrompu lorsque l'utilisateur clique sur l'écran avec la souris. Le but est de créer une expérience visuelle intéressante pour l'utilisateur avant de démarrer la partie.

LE DÉCALAGE

- Le tableau utilisé pour l'affichage des inversions a une limite de 13 lignes. Cependant, comme le nombre maximum d'inversions autorisé (calculé en multipliant le nombre d'inversions minimal pour gagner par 2) peut dépasser cette limite, nous avons implémenté un décalage pour résoudre ce problème. Lorsque le joueur atteint la ligne 13, toutes les lignes sont décalées d'une position vers le haut, permettant ainsi à la ligne 13 de devenir vide pour l'inversion suivante. Pour faciliter la correspondance entre les lignes et les mots inversés, nous avons ajouté des numéros de ligne à gauche du tableau et la position d'inversion à droite (voir la figure 9).

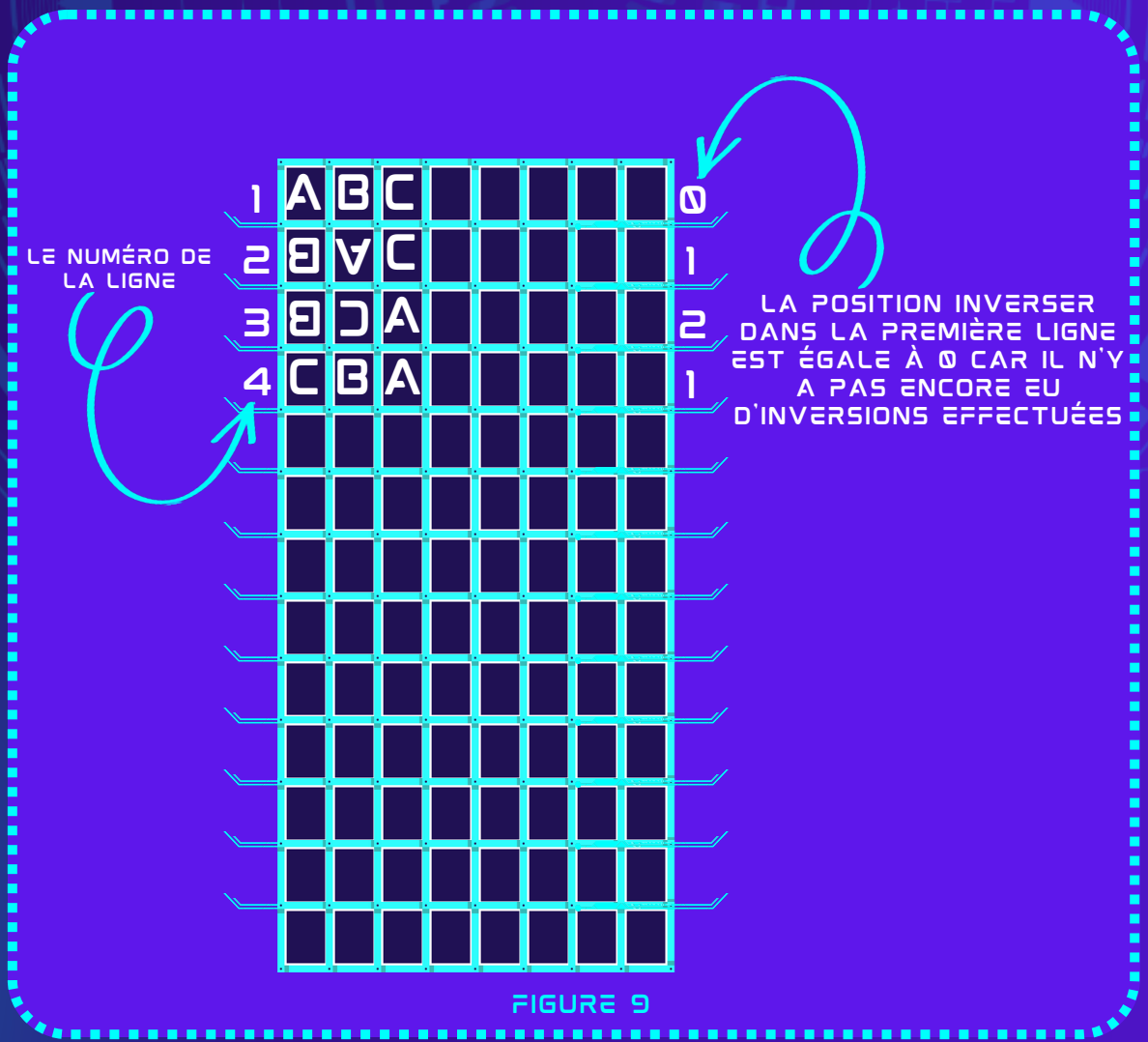

FIGURE 9

LOADING

- Nous avons constaté qu'à certaines occasions, la création de tous les cas prend beaucoup de temps, surtout si la taille du mot est grande. Pour résoudre ce problème, nous avons ajouté une fenêtre de chargement "loading" qui sera affichée pendant la création de tous les cas afin d'informer l'utilisateur que le processus est en cours.

LA SAUVEGARDE DE LA PARTIE

- Pour sauvegarder la partie en cours, nous avons créé un type "game" qui contient toutes les informations nécessaires à la reprise de la partie : la taille du mot, la taille d'inversion, le mot de départ, le mot final, le temps écoulé et d'autres variables nécessaires au fonctionnement du jeu (voir la figure 10).

```
typedef struct game{
    int N;
    int n;
    int Tinit[10];
    int Tresultat[10];
    float time_playing;
}game;
```

LE CODE A ÉTÉ SIMPLIFIÉ AFIN DE FACILITER LA COMPRÉHENSION DE L'ALGORITHME

FIGURE 10

- Ensuite, pour sauvegarder la partie en cours, on remplit tous les champs de la variable "game to save" avec les informations de la partie en cours, et on enregistre cette variable dans le fichier "game_file.bin" (voir la figure 11).

```
game game_to_saved;
//le remplisage d'enregistrement "game_to_saved"
game_to_saved.N=N;
game_to_saved.n=n;
for(i=0;i<10;i++){
    game_to_saved.Tinit[i]=Tinit[i];
    game_to_saved.Tresultat[i]=Tresultat[i];
}
game_to_saved.time_playing=time_playing;
//le sauvgadre dans le fichier
FILE *game_file_ptr;
game_file_ptr=fopen("Files/game_file.bin","wb");
fwrite(&player_game,sizeof(game),1,game_file_ptr);
fclose(game_file_ptr);
```

LE CODE A ÉTÉ SIMPLIFIÉ AFIN DE FACILITER LA COMPRÉHENSION DE L'ALGORITHME

FIGURE 11

- En outre, pour charger une partie sauvegardée, nous récupérons les informations du fichier "game_file.bin" en utilisant une variable "game_saved". Ensuite, nous passons les champs de cette variable à une fonction "resume" (voir la figure 12) ,qui affiche toutes les inversions effectuées par l'utilisateur avant qu'il n'ait sauvegardé la partie . Cette fonction permet également au joueur de continuer à inverser jusqu'à ce qu'il gagne ou perde. Cependant, si le joueur a sauvegardé la partie après avoir gagné ou perdu, il ne pourra pas effectuer d'autres inversions lorsqu'il chargera cette partie. Et le temps écoulé ne changera pas .

```
game game_saved;
//la recupération de la partie sauvgarder
FILE *game_file_ptr;
game_file_ptr = fopen("Files/game_file.bin","rb");
fread(&game_saved, sizeof(game),1,game_file_ptr);
fclose(game_file_ptr);
//le resumé de la partie
resume(game_saved.N,game_saved.n,game_saved.Tinit,game_saved.Tresultat,game_saved.time_playing);
```

LE CODE A ÉTÉ SIMPLIFIÉ AFIN DE FACILITER LA COMPRÉHENSION DE L'ALGORITHME

FIGURE 12

CLASSEMENT DES 5 MEILLEURS SCORES

- Lors de l'analyse de cette partie de jeu, nous avons constaté qu'il était injuste de classer les joueurs uniquement en fonction du temps mis pour gagner, Par exemple, si un joueur a gagné en 5 secondes avec un nombre d'inversions minimal (NUMBER OF FLIPS) de 2, et un autre joueur a gagné en 10 secondes avec un nombre d'inversions minimal (NUMBER OF FLIPS) de 6, il serait injuste que le premier joueur soit classé meilleur que le second dans la liste. Il fallait donc trouver une solution. Au départ, nous avions pensé à créer plusieurs listes pour les parties gagnées, une pour les mots de taille 3, une pour les mots de taille 4... jusqu'à 8. Cependant, nous avons réalisé que cette solution ne résoudrait pas le problème de la comparaison juste des performances des joueurs, car il est possible que deux combinaisons avec la même taille de mot (N) aient des nombres d'inversions minimal pour gagner (NUMBER OF FLIPS) très différents entre eux, le même problème pour "le nombre de lettres inversées à chaque tour (n)". Ainsi, après réflexion, nous avons décidé d'ajouter une variable "score" qui serait calculée de la manière suivante : SCORE = (TIME/NUMBER OF FLIPS) * 5; Cette méthode de calcul permet de pondérer le temps et le nombre d'inversions pour arriver à une valeur qui reflète la performance du joueur. Ainsi, lorsque nous comparons les parties gagnées, nous pouvons déterminer si un joueur a réalisé un score suffisamment élevé pour être classé dans les 5 meilleurs scores. Ainsi, le type de joueur devient comme ceci (voir figure 13).

```
typedef struct Player{
 int score;
 int time;
char name[10];
char date[11];
}Player;
```

FIGURE 13

LES PROBLÉMATIQUES

- Sans parler des problèmes que nous avons abordés plus tôt, tels que le score, le chargement et le décalage, nous avons rencontré de nombreux problèmes lors de notre parcours de développement. Un de ces problèmes était celui des caractères palindromes, c'est-à-dire des caractères qui ne changent pas de forme lorsqu'ils sont inversés, tels que "O", "S", "N", "Z", "I" et "H". Nous n'avions pas la possibilité de faire la différence entre eux, donc nous avons trouvé une solution pour résoudre ce problème. Il consistait à mettre un point rouge sur les caractères inversés pour les différencier des lettres non inversées (normales).
- Un autre problème que nous avons rencontré était celui de la machine. Lorsque la machine affichait une ligne, elle devait attendre 2 secondes pour afficher la ligne suivante. Nous avons utilisé la fonction sdl_delay pour cela. Cependant, cela posait un problème car, lorsque le jeu arrivait à l'instruction DELAY, les boutons de pause et de rechargement ne fonctionnaient pas car nous étions coincés dans l'instruction delay pendant 2 secondes. Par exemple, si vous cliquiez sur pause, il fallait 2 secondes pour afficher la fenêtre de pause (voir la figure 14).

L'INTRUCTION DELAY

```
while(RUNNING){
// les instruction de pause ;
// les instruction de reload ;
//les instruction de l'affichage de l'etape vuivant ;
DELAY(2000);
}
```

LE CODE A ÉTÉ SIMPLIFIÉ AFIN DE FACILITER LA COMPRÉHENSION DE L'ALGORITHME

FIGURE 12

- La solution de ce problème n'était pas si compliquée, nous avons lié l'affichage des étapes avec le temps (voir la figure 15).

```
int TIME_AFFICHE=0;
while(RUNNING){
// les instruction de pause ;
// les instruction de reload ;
IF(time > TIME_AFFICHE){
    //les instruction de l'affichage de l'etape vuivant;
    TIME_AFFICHE++;
}
```

LE CODE A ÉTÉ SIMPLIFIÉ AFIN DE FACILITER LA COMPRÉHENSION DE L'ALGORITHME

FIGURE 12

AMELIORATIONS POSSIBLES

- Il reste encore plusieurs améliorations à apporter à ce jeu. Par exemple, nous avons ajouté un décalage lorsqu'une partie arrive à la dernière ligne du tableau pour le cas "player". Il est également important de le faire pour les cas "machine" et "player vs machine". Ce problème n'a pas été résolu dans ces deux cas en raison du manque de temps.
- En suit, ce qui concerne la sauvegarde, il serait souhaitable de permettre aux joueurs de nommer leur partie pour la sauvegarder. Lorsqu'ils souhaitent charger une partie, une fenêtre s'afficherait contenant tous les noms des parties déjà enregistrées avec la date, permettant au joueur de choisir la partie qu'il souhaite continuer.
- La dernière amélioration concerne la fonction de création de tous les cas possibles mentionnée au début. Cette fonction vérifie si un élément est déjà présent dans la liste avant de l'ajouter. Pour cela, elle parcourt toute la liste. Il serait donc mieux de changer la structure de données en tableau et d'utiliser une table de hachage pour vérifier si un élément est déjà présent, ce qui permettrait de supprimer les fenêtres d'attente inutiles. De plus, la taille maximale des mots ne serait plus limitée à 8 caractères, elle pourrait être 9, 10, 15, sans aucun problème.

CONCLUSION :

- En conclusion, nous tenons à souligner que ce projet a été une expérience enrichissante pour nous, nous permettant d'améliorer nos compétences en travail d'équipe et de mieux comprendre le langage C, notamment les tableaux, les listes, les fichiers, les pointeurs, les arbres, etc. Cependant, il y a encore beaucoup de place pour des améliorations, telles que celles mentionnées précédemment, telles que l'optimisation des performances, la réduction de la consommation de mémoire, une meilleure conception du code et une plus grande fiabilité.
- Malgré ces défis, nous sommes fiers d'avoir réussi à développer ce jeu intégralement, avec ses trois parties et de nombreuses fonctionnalités supplémentaires. Nous sommes convaincus que ce projet a été bénéfique pour notre formation et pour notre développement en tant que développeurs de logiciels.
-

FIN.