ÉCOLE D'INGÉNIEUR DU LITTORAL CÔTE D'OPALE

Projet d'étude : Jeu Mastermind
Octobre 2024 - Janvier 2025

Projet réalisé par :
GUILLIN Stylian, ALLAIRE Thomas et VAZ E SILVA Angel

Tutrice : BEL MOUDDEN Oumaima

TABLE DES MATIÈRES

1 - Introduction

1.1 Présentation du projet

Le projet consiste à développer une version numérique du jeu Mastermind en langage C. Ce jeu, basé sur la réflexion et la logique oppose un joueur a une intelligence artificielle avec pour but de deviner une combinaison secrète en un nombre limité d'essais, en s'appuyant sur les indices fournis a chaque tentative.

1.2 Objectif

L'objectif principal est de reproduire les règles officielles du jeu tout en intégrant des fonctionnalités supplémentaire pour enrichir l'expérience utilisateur :

- sauvegarde et chargement des parties permettent de reprendre une partie à tout moment
- visualisation des parties jouées pour analyser le déroulement des parties précédente
- consultation des statistiques incluant les temps de réflexion et le nombre de tentatives des différentes parties

Le programme devra également garantir une interface simple et intuitive.

2 - Cahier des charges

2.1 Règles du jeu Mastermind

Mastermind est un jeu de réflexion ou le but est de deviner une combinaison secrète de couleurs. Les différentes couleurs du Mastermind sont :
le bleu, le rouge, le jaune, le vert, le orange, le violet et le blanc.
Chaque essai vous donne des indices pour vous rapprocher de la solution. Selon le niveau de difficulté, il y a aura plus ou moins d'indices.

2.2 Fonctionnalités attendues

Pour enrichir l'expérience utilisateur, plusieurs fonctionnalités doivent être implémentées :

- Sauvegarde et chargement des parties :
  - Permettre au joueur de sauvegarder une partie en cours et de la reprendre ultérieurement.
  - Les données sauvegardées incluront : la combinaison secrète, le nombre de tentatives restantes, les propositions précédentes et les indices donnés.

- Revisualisation des parties jouées
  - Fournir la possibilité de revoir les parties terminées pour analyser les différentes tentatives et les retours correspondants.
- Statistiques des parties :
  - Calculer et afficher des statistiques détaillées pour chaque joueur
  - Temps moyen de résolution d'une partie.
  - Nombre moyen de tentatives pour trouver la solution.
  - Historique des victoires et des défaites.

- Intelligence du joueur contrôlé par l'ordinateur
  - Implémenter un adversaire contrôlé par l'ordinateur capable de générer des combinaisons aléatoires
  - Permettre à l'ordinateur de jouer le rôle du joueur et de résoudre une combinaison secrète

- Interface utilisateur
  - Proposer une interface simple et intuitive, permettant une prise en main rapide.
  - Utiliser des menus textuels pour guider l'utilisateur dans les différentes option du jeu

- Gestion graphique
    - Intégrer une interface graphique pour une meilleure immersion (par exemple, une représentation visuelle des couleurs et des retours).

2.3 Contraintes et exigences techniques

Le projet doit respecter plusieurs contraintes technique et fonctionnelles:

1) Langage et environnement :
    - le jeu doit être entièrement développé en langage C
    - Les outils de développement incluront un éditeur de code standard et un compilateur compatible

2) Structure du programme
    - Le programme doit être structuré de manière modulaire avec des fonctions dédiées pour chaque fonctionnalité

3) Gestion des données :
    - Les sauvegardes devront être stockées dans des fichiers pour permettre leur réutilisation.
    - Les données statistiques devront être mises à jour dynamiquement après chaque partie

4) Performance :
    - Le programme doit fonctionner de manière fluide

5) Robustesse :
    - Le programme devra inclure des mécanismes pour gérer les erreurs, telles que des entrées utilisateur invalides.

3 - Gestion du projet

3.1 Planification des étapes du projet

Mois 1 : Préparation et Conception

- Semaine 42 : Définition des objectifs et des spécifications
  - Clarification des règles du jeu et des fonctionnalités à implémenter.
  - Rédaction du cahier des charges.
- Semaine 43-44 : Conception de l'architecture du jeu
  - Planification de l'interface utilisateur (console).
  - Définition des structures de données (par exemple, le code secret, les tentatives).
- Semaine 45 : Préparation de l'environnement de développement
  - Configuration du projet sur l'IDE (environnement de développement intégré) et du gestionnaire de version (Git).
  - Planification des premières étapes de développement et affectation des tâches.

Mois 2 : Développement de la logique du jeu

- Semaine 46-47 : Implémentation de la génération du code secret
  - Création de la logique pour générer un code secret aléatoire.
  - Implémentation de la vérification des tentatives (comparaison du code secret avec les propositions).
  - Développement des fonctions de contrôle de jeu (répétition des tentatives, fin du jeu).
- Semaine 48 : Développement des règles du jeu et du mécanisme de scoring
  - Mise en place des règles du jeu (nombre d'essais, critères de victoire).
  - Calcul et affichage du score

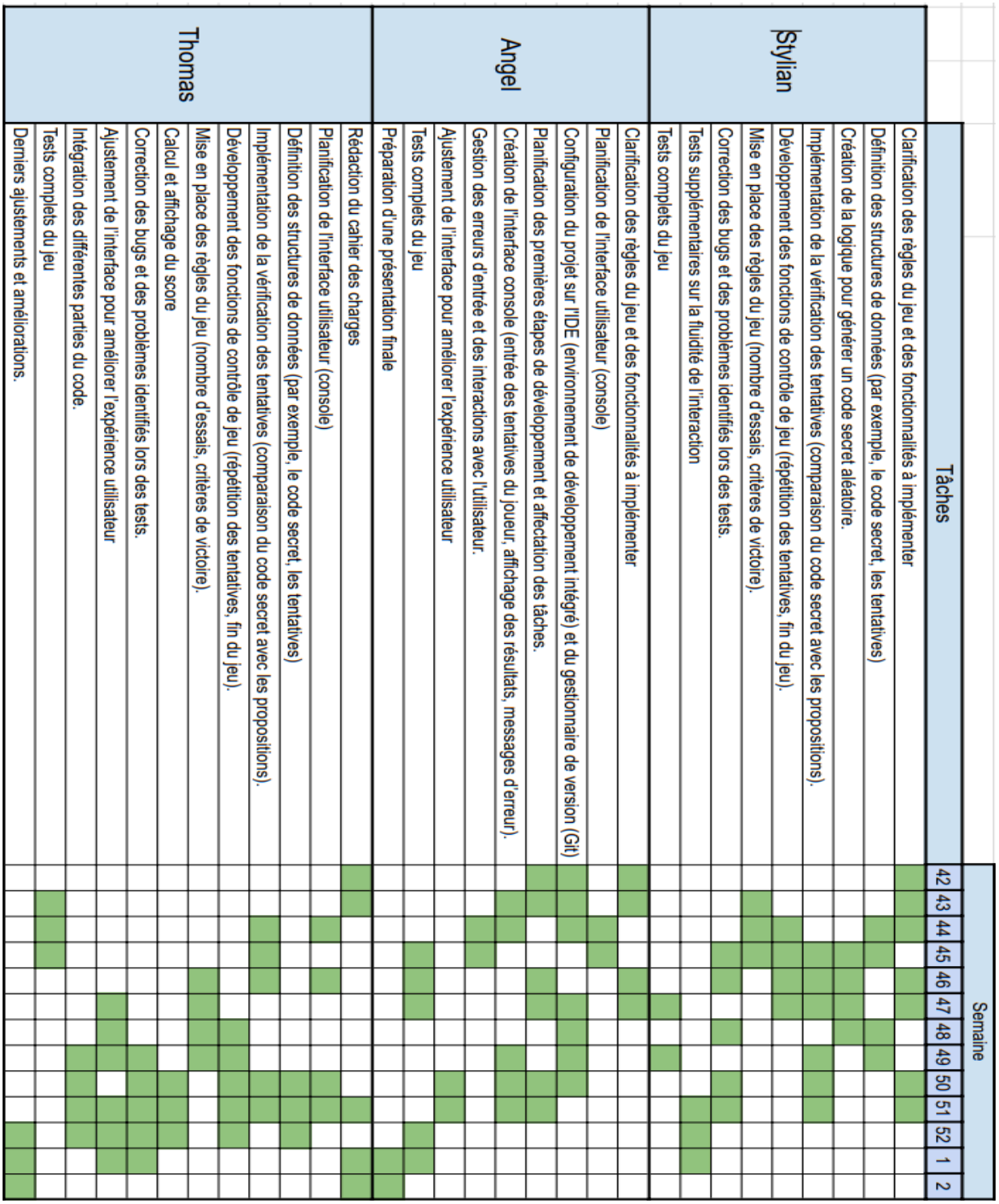
Mois 3 : Interface Utilisateur et Tests

- Semaine 49-50 : Développement de l'interface utilisateur
  - Création de l'interface console (entrée des tentatives du joueur, affichage des résultats, messages d'erreur).
  - Gestion des erreurs d'entrée et des interactions avec l'utilisateur.
- Semaine 51 : Tests unitaires et débogage
  - Correction des bugs et des problèmes identifiés lors des tests.
- Semaine 52 : Amélioration de l'interface et des retours utilisateur
  - Ajustement de l'interface pour améliorer l'expérience utilisateur
  - Tests supplémentaires sur la fluidité de l'interaction

Mois 4 : Finalisation et Documentation

- Semaine 1 : Intégration et tests complets
  - Intégration des différentes parties du code.
  - Tests complets du jeu
  - Détection et correction de bugs.
- Semaine 2 : Finalisation et présentation
  - Préparation d'une présentation finale
  - Derniers ajustements et améliorations.

- 3.2 Répartition des tâches



4 - Analyse et conception

- 4.1 Architecture générale du programme

L'architecture générale du programme de votre jeu peut être organisée en plusieurs modules ou fichiers, chacun ayant une responsabilité distincte. Voici un aperçu de l'architecture générale et des modules qui est présents :

- **Module Principal (`main.c`)**
- **Module de Sauvegarde (`save.c` et `save.h`)**
- **Module de Gestion du Jeu (`jeu.c` et `jeu.h`)**

- 4.2 Description des modules et fonctions

1. Module Principal (`main.c`)

- **Responsabilité** : Point d'entrée du programme. Ce fichier est responsable de l'affichage du menu principal, de la gestion des choix de l'utilisateur et de l'initialisation des différentes parties du jeu.
- **Fonctionnalités** :
  - Affichage du menu principal.
  - Gestion de la boucle principale du jeu.
  - Appel des différentes fonctions de jeu selon l'option choisie (commencer une partie, afficher les règles, gérer les statistiques, etc.).
  - Appel aux fonctions de sauvegarde et de chargement

2. Module de Sauvegarde (`save.c` et `save.h`)

- **Responsabilité** : Gérer toutes les opérations de sauvegarde et de chargement des parties, ainsi que les statistiques du jeu.
- **Fonctionnalités** :
  - Créer une nouvelle sauvegarde.
  - Sauvegarder les tentatives du joueur et les informations de la partie.
  - Afficher une sauvegarde existante.
  - Modifier les statistiques du jeu.
  - Lire et afficher les statistiques.

3. Module de Gestion du Jeu (`jeu.c` et `jeu.h`)

- **Responsabilité** : Contient les fonctions principales liées à la logique du jeu, telles que l'initialisation de la partie, la gestion des tentatives, et la vérification du code secret.
- **Fonctionnalités** :
  - Initialisation du jeu (choix des joueurs, du mode et de la difficulté).
  - Gestion des règles du jeu.
  - Traitement des tentatives et comparaison avec le code secret.
  - Détection des couleurs correctes et des positions.

5 - Algorithmes développés

Dans cette partie, les fonctions afficher feront en sorte que le texte soit affiché dans la console et les fonctions écrire feront en sorte que le texte soit écrit dans le fichier ouvert.

- 5.1 Génération aléatoire de la combinaison secrète

Pour générer une combinaison aléatoire nous avons utilisé des fonctions pré-faites des bibliothèques du langage C.

Fonction codealeatoire(code, diff, nbrcouleur)
Arguments
code : liste de caractère
diff : caractère entier entre 1 et 3
nbrcouleur : entier égal à 4

```
Début fonction
        couleurs = ['r', 'y', 'g', 'o', 'w', 'p', 'b']
        indices = [0, 1, 2, 3, 4, 5, 6]
        max = 7
        Si diff == '1' ou '2' :
                Pour i de 0 à nbrcol - 1 :
                        alea = rand() % max
                        code[i] = couleurs[indices[alea]]
                        indices[alea] = indices[max - 1]
                        max -=1
                fin pour
        fin si
        Sinon :
                Pour i de 0 à nbrcol - 1 :
                        alea = rand() %
                        max code[i] = couleurs[alea]
                fin pour
        fin sinon
Fin Fonction
```

- 5.2 Gestion des statistiques (temps et tentatives)

Pour bien structurer les statistiques du mastermind nous avons utiliser plusieurs fonctions qui puissent lire, creer un fichier sur les statistiques du joueur si il n'y en a pas encore et enfin modifier ce fichier pour chaque joueurs.

Fonction lirestatistique(nom)
Arguments
nom : liste de caractere

```
Début fonction
    statistique = “statistique” + nom
    stat = open(statistique, “r”)
    Si stat == NULL
        close(stat)
        retourner 0
    Fin si
    ligne = []
    Tant que gets(ligne, 1000, stat) != NULL
        afficher ligne
    Fin tant que
    close(stat)
    retourner 1
Fin fonction
```

```
Fonction creerstat(nom)
Arguments
nom : liste de caractères

Début Fonction
    statistique = “statistique”+nom
    stat = open(statistique, “w”)
    écrire “Nom : %s” nom
    victoire = 0
    defaite = 0
    temps = 0.0
    tentative = 0.0
    écrire “Victoire : %d” victoire
    écrire “défaite : %d” defaite
    écrire “Temps moyen : %lf” temps
    écrire “Nombre de tentatives moyennes : %lf” tentative
    close(stat)
Fin fonction
```

Par la suite, nous avons introduit une structure pour récupérer les données des statistiques du joueur que l’on souhaite voir.

```
typedef struct {
   int victoire;
   int defaite;
   double tentativemoy;
   double tempsmoy;
} stats;
```

Avec cette structure on peut maintenant créer la fonction pour récupérer les données.

```
Fonction retour(nom)
Arguments
nom : liste de caractères
```

```
Début fonction
    statistique = “statistique”+nom
    stat = open(statistique, “r”)
    ligne = []
    stats stati
    Tant que gets(ligne, 1000, stat) != NULL
        Si strstr(ligne, “Victoire” != NULL
            stati.victoire = ligne[11]
        Fin si
        Si strstr(ligne, "Defaite") != NULL
            stati.defaite = ligne[10] - '0';
        Fin si
        Si strstr(ligne, "Temps") != NULL
            fscanf(stat, "%lf", &stati.tentativemoy);
        Fin si
        Si strstr(ligne, "tentatives") != NULL
            fscanf(stat, "%lf", &stati.tentativemoy);
        Fin si
    Fin tant que
    close(stat)
    retourner stati
fin fonction


Fonctoin modifstatistique(nom, vicdef, temps, tentative)
Arguments
nom : liste de caractères
vicdef : entier indiquant la victoire du joueur 1 ou la défaite du joueur 0
temps : temps que le joueur a mis pour deviner le code
tentative : nombre de tentatives qu’a fait le joueur pour deviner le code

Début fonction
    statistique = “statistique” + nom
    stat = open(statistique, “w”)
    Si stat == NULL
        afficher “Il n’ya a pas encore de statistique sur ce joueur”
        close(stat)
        retourner
    Fin si
    close(stat)
    stats ret = retour(nom)
    stat2 = open(statistique, “w”)
    écrire “Nom : %s ” nom
    victoire = ret.victoire
    defaite = ret.defaite
    tot = victoire + defaite
    SI vicdef == 1
        victoire +=1
    Fin si
```

```
Si vicdef == 0
    defaite +=1
tpsmoy = (tot*ret.tempsmoy + temps)/(victoire + defaite)
tentmoy = (tot*ret.tentativemoy + tentative)/(victoire+defaite)
écrire “Victoire : %d ” victoire
écrire “Defaite : %d ” defaite
écrire “Temps moyen : %lf ” tpsmoy
écrire “Nombre de tentatives moyennes : %lf” tentmoy
close(stat)
```

- 5.3 Sauvegarde des parties

Afin de sauvegarder les parties, nous avons choisi 3 fonctions afin de créer la sauvegarde, la modifier et la finir.

Fonction creersave(nom, code, nbrcouleur)
Arguments
nom : liste de caractère indiquant le nom de la sauvegarde
code : liste de 4 caractère contenant le code aléatoire
nbrcouleur : entier égal à 4

```
Début fonction
        save = open(nom, "w ")
        Si save == NULL
            afficher " Erreur d’ouverture "
            retourner
        Fin si
        écrire " Nouvelle partie \n"
        écrire " Code secret : "
        Pour i allant de 0 à nbrcouleur-1 :
            écrire " code[i] "
        Fin pour
        écrire " \n "
        fermer save
Fin Fonction
```

Fonction modifsave(nom, joueur, couleurbonne, nbrcouleur, tentative)
Arguments
nom : liste de caractères contenant le nom de la sauvegarde
joueur : liste de 4 caractères contenant les essais du joueur
couleurbonne : entier indiquant le nombre de bonne couleurs
nbrcouleur : entier égal à 4
tentative : entier indiquant le chiffre de la tentative en cours

```
Début fonction
        save = open(nom, "a ")
        écrire " Tentative %d " tentative
        Pour i allant de 0 à nbrcouleur-1 :
            écrire " joueur[i] "
```

```
                Fin pour
                écrire " dont %d couleurs bonnes " couleurbonne
                écrire " \n "
                fermer save
fin fonction

Fonction finsave(nom)
Argument
nom : liste de caractères indiquant le nom de la sauvegarde

début fonction
                save = open(nom, "a ")
                écrire " FIN "
                fermer save
Fin fonction
```

- 5.4 Intelligence du joueur contrôlé par l'ordinateur

Pour cette partie nous avons utilisé un algorithme connu pour son efficacité et sa facilité d'exécution. Il se divise en 3 parties :

- Définir toute les combinaisons possibles (ici on a 7 couleurs donc 7^4 combinaisons
- Obtenir un feedback sur le nombre de bonne couleurs bien placé et mal placé après chaque essai
- Filtrer les combinaisons jusqu'à obtenir la meilleure combinaison

```
Fonction allcombinaisons(combinaisons)
Arguments :
combinaisons : matrice de taille nbrcouleurtotal * nbrcouleurdeviner

Début fonction
        couleurs = ['r', 'y', 'g', 'o', 'w', 'p', 'b']
        x = 0
        pour i allant de 0 à 6 :
                pour i allant de 0 à 6 :
                        pour k allant de 0 à 6 :
                                pour l allant de 0 à 6 :
                                        combinaisons[x][0] = couleurs[i]
                                        combinaisons[x][0] = couleurs[i]
                                        combinaisons[x][0] = couleurs[i]
                                        combinaisons[x][0] = couleurs[i]
                                        x+=1
                                                fin pour
                                fin pou
                        fin pour
                fin pour
fin fonction
```

```
Fonction getfeedback(combinaison, code)
Arguments
combinaison : liste de 4 caractères
code : liste de 4 caractères
début fonction
        fb = [0,0]
        combicount = [0, 0, 0, 0, 0, 0, 0]   //initialisé de sorte de que chaque numéro
        codecount = [0, 0, 0, 0, 0, 0, 0]    //soit relié à une couleur
        Pour i allant de 0 à 4-1 :
                Si combinaison[i] == code[i] :
                        fb.noir +=1
                        Fin si
                Sinon :
                        Ajouter 1 à la position de la couleur dans la liste combicount
                        Ajouter 1 à la position de la couleur dans la liste codecount
                Fin sinon
        Pour i allant de 1 à 7 :
                Si combicount[i] < codecount[i] :
                        fb.blanc += combicount[i]
                Fin si
                Sinon
                        fb.blanc += codecount[i]
                Fin sinon
        Fin pour
        retourner fb
Fin fonction

Fonction filtrecombinaisons(combinaisons, combi ; fb, filtrecombi, indice)
Arguments
combinaisons : matrice contenant les toute les combinaisons
combi : liste de caractère, essai de l'ordinateur
fb : liste de deux entiers correspondant aux couleurs bonne et bien placé (noir) et
bonne mais mal placé (blanc)
filtrecombi : matrice contenant les combinaisons dont les couleurs bien placé
correspondent à celle de combi
indice : entier indiquant le nombre de combinaisons restante
Début fonction
        maxind = 0
        Pour i allant de 0 à indice-1 :
                temp = [combinaisons[i][0], combinaisons[i][1], combinaisons[i][2], [i][3]]
                feedback = getfeedback
                Si fb.noir == feedback.noir et fb.blanc == feedback.blanc :
                        filtrecombi[maxind][0] = temp[0]
                        filtrecombi[maxind][1] = temp[1]
                        filtrecombi[maxind][2] = temp[2]
                        filtrecombi[maxind][3] = temp[3]
                        maxind+=1
                Fin si
        Fin pour
Fin fonction
```

6 - Connaissances acquises

6.1 Compétences techniques développées

Ce projet nous a permis d'apprendre plusieurs compétences de programmation, notamment l'utilisation de listes et de tableaux ainsi que la manipulation de fichiers via le langage C. Ces compétences nous ont aidé à gérer les combinaisons de couleurs et les propositions des utilisateurs, la conception d'algorithmes précis, ainsi que l'implémentation de fonctionnalités avancées comme la sauvegarde et le chargement de parties. Nos compétences en termes de gestion des erreurs ont été renforcées, en effet, plus nous avancions dans ce projet plus nous rencontrions d'éventuels problèmes et nous avons réussi à les surpasser. Enfin, l'utilisation d'outils de développement tels que CodeBlock ou un compilateur en ligne de commande a permis de déboguer et d'optimiser le programme efficacement.

6.2 Leçons tirées du projet

Le projet a souligné l'importance d'une planification rigoureuse et d'une organisation méthodique pour atteindre les objectifs fixés. Structurer le programme de manière modulaire a simplifié le processus de développement et permis d'apporter des améliorations ciblées sans impacter le reste du code. La gestion des imprévus, tels que les bugs ou les difficultés rencontrées lors de l'implémentation de certaines fonctionnalités comme les statistiques ou les sauvegardes, a renforcé la résilience et la capacité à trouver des solutions alternatives. Les tests réguliers se sont avérés essentiels pour identifier rapidement les erreurs et garantir la robustesse du programme. Enfin, ce projet a permis de mieux comprendre les défis liés au développement d'un logiciel tout en ouvrant des perspectives d'amélioration, comme l'intégration future d'une interface graphique pour enrichir l'expérience utilisateur.

7 - Difficultés rencontrées

7.1 Problèmes techniques

Plusieurs défis techniques ont été rencontrés au cours du développement du projet. L'un des premiers problèmes a concerné la gestion des combinaisons de couleurs et leur vérification, notamment pour détecter correctement les couleurs bien placées et mal placées sans erreurs de comptage. La génération d'une combinaison aléatoire de couleurs a également présenté des difficultés, en raison d'une mauvaise

initialisation du générateur aléatoire, entraînant des résultats prévisibles ou non diversifiés. La mise en œuvre de la fonctionnalité de sauvegarde et de chargement s'est avérée complexe, notamment pour garantir l'intégrité des données et leur compatibilité entre différentes exécutions du programme. Enfin, des problèmes de segmentation fault et de dépassement de tableau ont été observés lors de la gestion des entrées utilisateur, en raison de l'absence initiale de validations robustes.

7.2 Solutions apportées

Pour ces problèmes, plusieurs approches ont été prises. la correction des combinaisons aléatoires sans doublons, les couleurs bien placées. Pour la génération d'une combinaison, la fonction aléatoire a été initialisée avec (*srand(time())*), garantissant des résultats variés à chaque exécution. Pour la gestion des sauvegardes, une structure de données claire a été définie et sérialisée dans des fichiers texte, avec des contrôles pour vérifier la validité des données avant leur chargement. Les erreurs liées aux entrées utilisateur ont été corrigées en ajoutant des validations strictes et en limitant les dépassements de tableau grâce à des boucles conditionnées. De plus, des tests réguliers et l'utilisation d'un débogueur ont permis de localiser rapidement les erreurs et d'améliorer la robustesse du programme.

8 - Améliorations et évolutions possibles

8.1 Suggestions pour l'amélioration du programme

Le programme actuel peut être amélioré sur plusieurs aspects pour offrir une expérience plus fluide et optimisée. Une des premières pistes d'amélioration concerne l'interface utilisateur : remplacer l'affichage textuel par une interface graphique intuitive permettra aux joueurs de sélectionner les couleurs directement via des clics, rendant le jeu plus visuel et agréable. Ensuite, l'optimisation des algorithmes, notamment ceux utilisés pour vérifier les combinaisons, pourrait réduire leur complexité, ce qui améliorerait les performances dans des configurations plus exigeantes (longueur élevée des combinaisons ou nombre important de couleurs). De plus, des efforts supplémentaires pourraient être faits pour renforcer la gestion des erreurs : une validation encore plus rigoureuse des entrées utilisateur permettrait d'éviter les comportements imprévus. Enfin, l'ajout de messages explicatifs et de tutoriels interactifs offrirait une meilleure prise en main du jeu, en particulier pour les nouveaux utilisateurs.

8.2 Idées pour des extensions futures

Dans une optique d'évolution, plusieurs idées pourraient enrichir le programme et en faire une plateforme plus complète. Par exemple, l'ajout de nouveaux modes de jeu, comme un mode chronométré où le joueur doit résoudre la combinaison dans un temps limité, ou un mode compétitif multijoueur permettant de comparer les scores, apporterait une dimension ludique supplémentaire.

Par ailleurs, une version connectée pourrait être envisagée : les joueurs pourraient sauvegarder leurs parties et consulter leurs statistiques en ligne, rendant le jeu accessible sur plusieurs appareils. Une telle fonctionnalité pourrait s'accompagner d'une compatibilité multiplateforme, avec des versions pour appareils mobiles ou navigateurs web, en exploitant des bibliothèques graphiques ou des technologies.

Enfin, la personnalisation des règles et paramètres du jeu, comme le choix du nombre de couleurs ou de la longueur des combinaisons, offrira une expérience sur mesure et adaptée aux préférences de chaque utilisateur.

9 - Conclusion

Ce projet a permis de concevoir et de développer une version numérique du jeu Mastermind en langage C, respectant les règles traditionnelles tout en intégrant des fonctionnalités supplémentaires. Parmi les réalisations majeures, on peut citer l'implémentation d'un système de vérification efficace des combinaisons, la possibilité de sauvegarder et de charger des parties, ainsi que la création d'un système de statistiques permettant d'analyser les performances des joueurs. Le programme a été conçu de manière modulaire, ce qui facilite sa maintenance et son évolution. Malgré les défis rencontrés, notamment liés à la génération aléatoire de combinaisons et à la gestion des entrées utilisateur, des solutions robustes ont été mises en œuvre pour garantir un fonctionnement fluide et fiable.