Résolution de Problèmes

2023-2024

RAPPORT PROJET

JEU DE MOULIN

Réalisé par :

Gourri Abdellatif

Mouh Kaoutar

Taleb Douae

Tarmoumia Hiba

Enadrement :

Mme Addou Malika

Table des matières

1 Introduction

1.1 Contexte du projet :

Le projet de création du jeu de Moulin émerge dans un contexte où les jeux de société traditionnels rencontrent un regain d'intérêt, notamment avec l'avènement de l'intelligence artificielle. L'objectif principal de ce projet est de concevoir et développer des versions informatiques du jeu de Moulin, permettant aux joueurs d'affronter un adversaire virtuel basé sur les algorithmes de recherche MiniMax et Alpha-Beta. Ce rapport présente en détail le processus de création, les choix technologiques, ainsi que les défis rencontrés et les solutions apportées tout au long du développement.

1.2 Objectifs du projet :

Les principaux objectifs de ce projet sont multiples. Tout d'abord, il s'agit de concevoir et développer des versions informatiques fidèles du jeu de Moulin, en respectant ses règles et ses spécificités. Ensuite, l'implémentation des algorithmes MiniMax et Alpha-Beta permettra d'offrir une expérience de jeu compétitive et stimulante pour les utilisateurs, en les confrontant à un adversaire virtuel capable de prendre des décisions basées sur une analyse approfondie du plateau de jeu. Enfin, ce projet vise à explorer et à comprendre les défis liés à la modélisation de problèmes de jeu et à l'implémentation d'algorithmes d'intelligence artificielle dans un contexte ludique.

1.3 Aperçu du contenu du rapport :

Ce rapport est divisé en plusieurs sections qui couvrent différents aspects du projet. Après cette introduction, la section suivante présente une vue d'ensemble du jeu de Moulin, en explorant son histoire, ses règles et son attrait pour les joueurs. Ensuite, la conception et le développement du jeu sont détaillés, en mettant en lumière les choix technologiques, l'architecture logicielle et les défis rencontrés lors de l'implémentation. La section consacrée à l'implémentation du code examine en détail la structure du code source, la conception de l'interface graphique et les performances du jeu. Enfin, une analyse des stratégies de résolution utilisées, ainsi que des perspectives d'évolution, concluent ce rapport.

2 Présentation du Jeu de Moulin :

2.1 Historique et origines du jeu

Le jeu de Moulin, également connu sous le nom de Nine Men's Morris, trouve ses racines dans l'Antiquité, remontant à plusieurs millénaires. Il a été découvert dans diverses civilisations à travers le monde, notamment en Égypte, en Grèce, et chez les Celtes. Son attrait réside dans sa simplicité apparente et sa profondeur stratégique, ce qui en fait un jeu intemporel. L'adaptation informatique du jeu de Moulin perpétue cette tradition séculaire en le rendant accessible à un public moderne tout en préservant son essence historique et culturelle.

2.2 Description des règles du jeu :

Le jeu de Moulin se déroule sur un plateau de jeu composé de lignes interconnectées formant une grille. Chaque joueur dispose d'un certain nombre de pions qu'il place tour à tour sur les intersections des lignes. L'objectif est de former des moulins, c'est-à-dire des lignes de trois pions de sa propre couleur, ce qui permet de capturer les pions adverses. Une fois que tous les pions sont placés sur le plateau, les joueurs peuvent déplacer leurs pions le long des lignes pour tenter de former de nouveaux moulins ou bloquer les mouvements de l'adversaire. Le jeu se termine lorsqu'un joueur ne peut plus déplacer de pions ou lorsqu'il ne lui reste que deux pions, ce qui entraîne la victoire de l'adversaire.

2.3 Intérêt et Spécifités du Jeu :

Le jeu de Moulin présente plusieurs caractéristiques qui en font un défi stratégique captivant. Tout d'abord, sa simplicité apparente cache une profondeur tactique qui offre de nombreuses possibilités de jeu et de contre-jeu. De plus, le jeu combine des éléments de placement initial et de déplacement ultérieur, ce qui nécessite une réflexion à long terme et une anticipation des mouvements adverses. Enfin, la capture des pions ennemis ajoute une dimension compétitive supplémentaire, où chaque décision peut avoir un impact significatif sur le déroulement de la partie. L'adaptation informatique du jeu de Moulin offre aux joueurs une expérience immersive et stimulante, tout en préservant l'essence stratégique et ludique du jeu traditionnel.

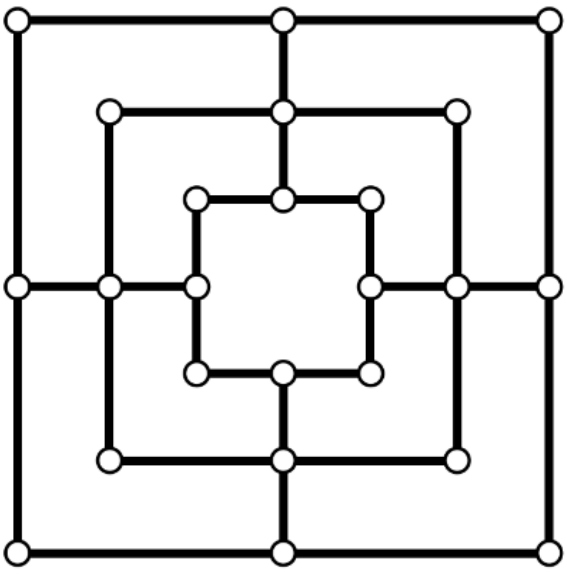
3 Modélisation du Jeu de Moulin

La modélisation du jeu de Moulin a été réalisée à travers la définition de structures de données qui représentent l'état du jeu, les positions sur la grille, ainsi que les nœuds pour l'implémentation d'algorithmes de recherche. Ces structures sont définies dans le fichier `structure.h` et sont essentielles pour gérer les différentes phases du jeu, les mouvements des joueurs, et l'évaluation des états du jeu.

3.1 Structure de la Grille :

La grille de jeu est représentée par une structure `place` qui définit les caractéristiques de chaque position sur la grille :

- **lettre** : Identifiant de la position sur la grille.
- **joueur** : Indicateur du joueur qui occupe la position (0 pour vide, 1 pour le joueur 1, et 2 pour le joueur 2).
- **couleur** : Peut être utilisé pour des extensions du jeu ou pour indiquer des états spéciaux des positions.
- **v1, v2, v3, v4** : Identifiants des positions voisines. Cela permet de naviguer facilement sur la grille et de vérifier les alignements et les mouvements possibles.



3.2 État du Jeu :

L'état global du jeu est représenté par la structure `EtatJeu`, qui contient l'ensemble des positions de la grille et des informations supplémentaires pour suivre l'évolution du jeu :

— **P[24]** : Tableau des positions de la grille.

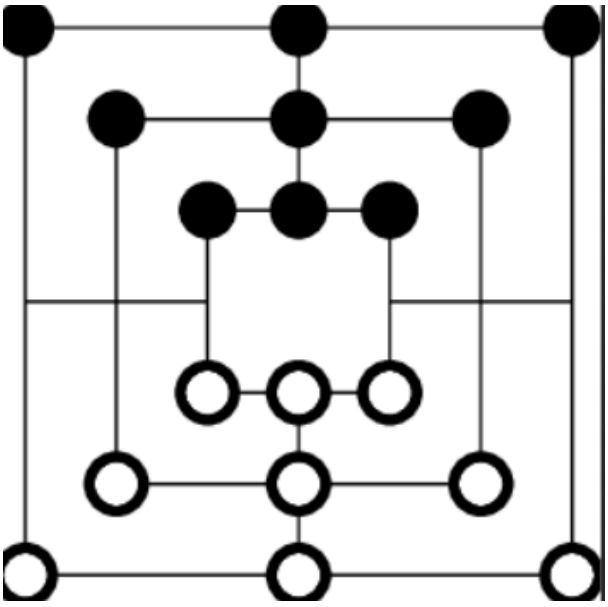
— **pos1** et **pos2** : Comptent les pions de chaque joueur sur la grille.

— **ce** : Coût de l'état, utilisé dans les algorithmes de recherche pour évaluer la qualité d'un état.

— **pos** : Position courante dans certains contextes, pour suivre les mouvements récents.

— **presd** : État de présélection, utile pour certaines mécaniques de jeu ou pour les phases de transition.

— **pC** : Nombre de pions capturés de l'adversaire, utilisé pour suivre le score ou des règles spécifiques du jeu.



3.3 Nœud de l'Arbre de Recherche :

Pour les algorithmes de recherche, comme Minimax, chaque nœud de l'arbre est représenté par la structure `Noeud` :

- **suivant** : Pointeur vers le nœud suivant dans une liste chaînée, permettant de représenter les différents états explorés par l'algorithme.
- **Etat** : Instance de la structure `etat` représentant l'état du jeu à ce nœud particulier.

3.4 Espace des États :

L'espace des états du jeu de Moulin comprend toutes les configurations possibles de la grille et les positions des pions des deux joueurs à un moment donné. Chaque état du jeu est une instance unique de la structure `etat`, définie par les positions des pions sur la grille (`P[24]`), le nombre de pions pour chaque joueur (`pos1` et `pos2`), et d'autres variables de contexte telles que le coût de l'état (`ce`) et les pions capturés (`pC`).

L'algorithme de recherche explore cet espace en générant de nouveaux états à partir des mouvements possibles des joueurs :

1. **États Initiaux** : Au début du jeu, les états initiaux sont définis par des configurations où les pions sont placés sur leurs positions de départ.
2. **Transitions** : Les transitions entre les états sont définies par les règles du jeu de Moulin, où chaque mouvement d'un pion crée un nouvel état.
3. **États Finaux** : Le jeu de Moulin se termine dans deux cas :
   - Lorsqu'un joueur est réduit à seulement deux pièces, ce qui l'empêche de former un moulin et donc de continuer à jouer.
   - Lorsqu'un joueur n'est plus en mesure d'effectuer un mouvement légal, ce qui bloque son avancée dans la partie. Dans ces deux situations, le joueur adverse remporte la partie.

3.5 Utilisation et Intégration des Structures :

Ces structures sont utilisées conjointement pour modéliser le jeu de Moulin, gérer les mouvements des joueurs, et implémenter des algorithmes de recherche et d'évaluation. Elles permettent une gestion efficace de l'état du jeu et facilitent l'implémentation de stratégies de jeu complexes.

3.6 Règles de Production :

Les règles de production du jeu de Moulin déterminent les actions légales qu'un joueur peut effectuer à partir d'un état donné du jeu :

- Chaque joueur commence avec neuf pions de sa couleur, disposés sur les intersections du plateau de jeu.
- Durant la première phase, les joueurs alternent pour placer un pion sur une intersection vide.
- Une fois que tous les pions ont été placés, les joueurs alternent pour déplacer un de leurs pions le long des arêtes du plateau vers une intersection vide adjacente.
- Si un joueur forme un moulin (alignement de trois de ses pions le long d'une ligne droite sur le plateau), il est autorisé à capturer l'un des pions adverses.
- Le joueur adverse peut éviter la capture en déplaçant l'un de ses pions vers une autre intersection.
- Le jeu se termine lorsqu'un joueur réduit son adversaire à deux pions ou lorsqu'un joueur ne peut plus effectuer de mouvement légal.

3.7 Algorithmes de résolution utilisés :

Dans le développement du jeu de Moulin, deux stratégies de recherche sont mises en œuvre : Minimax et AlphaBeta. Ces algorithmes visent à déterminer le meilleur coup à jouer à partir de l'état actuel du jeu.

3.7.1 Stratégie Minimax :

La stratégie Minimax explore les différentes possibilités de jeu jusqu'à une certaine profondeur, en alternant entre la maximisation et la minimisation des scores pour chaque joueur. Dans l'implémentation fournie, cette stratégie évalue chaque coup possible en fonction de critères tels que la formation de moulins, le nombre de pions sur le plateau et la proximité des moulins. Elle attribue ensuite un score à chaque état du jeu, permettant ainsi de déterminer la meilleure ligne de conduite.

Dans le cadre du jeu de Moulin, la stratégie Minimax évalue les états du jeu en fonction de leur potentiel pour le joueur humain et la machine. Elle examine les différentes positions des pions et cherche à maximiser le score pour la machine tout en minimisant celui du joueur

humain. Cette approche permet à l'intelligence artificielle de prendre des décisions compétitives et de rivaliser avec les joueurs humains.

3.7.2 Stratégie AlphaBeta :

D'autre part, la stratégie AlphaBeta est une amélioration de Minimax qui réduit le nombre de nœuds explorés en éliminant les branches inutiles de l'arbre de recherche. En utilisant une technique d'élagage, AlphaBeta permet d'explorer plus efficacement l'arbre de jeu tout en conservant la même qualité de décision que Minimax.

Dans l'implémentation des deux stratégies, des mécanismes sont mis en place pour gérer les différentes phases du jeu, telles que la phase de placement initial des pions et la phase de déplacement des pions. De plus, une fonction heuristique est utilisée pour évaluer la qualité de chaque état du jeu, en prenant en compte des aspects tels que le nombre de pions de chaque joueur et la formation de moulins.

En combinant Minimax ou AlphaBeta avec une fonction heuristique adaptée, l'intelligence artificielle peut prendre des décisions éclairées et rivaliser avec les joueurs humains. Ces stratégies de recherche permettent ainsi de créer une expérience de jeu stimulante et captivante pour les utilisateurs.

3.8 Fonction Heuristique :

La fonction heuristique h est un élément crucial de l'algorithme utilisé dans le jeu de Moulin pour évaluer la qualité de chaque état du jeu. Son rôle est de fournir une estimation de la position actuelle du jeu en attribuant un score à chaque état en fonction de divers critères. Dans l'implémentation fournie, la fonction h parcourt l'ensemble des emplacements sur le plateau de jeu. Pour chaque emplacement, elle évalue plusieurs aspects :

— Elle compte le nombre de pions de chaque joueur ($nb1$ pour le joueur 1 et $nb2$ pour le joueur 2), ce qui donne une indication de l'avantage numérique de l'un par rapport à l'autre.

— Elle accorde des points supplémentaires si un moulin est formé par l'un des joueurs sur un emplacement donné, ce qui signifie que trois pions du même joueur sont alignés le long d'une ligne droite. Ces points supplémentaires sont plus élevés lorsque le moulin

est potentiellement complété par un pion supplémentaire.

— Enfin, elle calcule la différence entre le nombre total de points attribués au joueur 1 et celui attribué au joueur 2, et stocke ce résultat dans la variable $c.ce$.

Cette fonction heuristique est conçue pour évaluer rapidement la qualité relative des états du jeu, en prenant en compte des aspects tels que le nombre de pions de chaque joueur, la formation de moulins et la proximité de ces formations. Elle fournit ainsi une estimation approximative de la position actuelle du jeu, ce qui permet à l'algorithme de recherche de prendre des décisions plus éclairées sur les coups à jouer.

3.9 Représentation Graphique des Arbre de Recherche :

FIGURE 1 – Arbre de Recherche de la Stratégie Minimax

FIGURE 2 – Arbre de Recherche de la Stratégie Alpha-Beta

3.10 Comparaison des Techniques de Recherche :

Minimax

— Explore exhaustivement tous les coups possibles jusqu'à la profondeur spécifiée.

— Peut être lent pour de grandes profondeurs à cause de l'explosion combinatoire des possibilités.

Alpha-Beta

— Optimise Minimax en éliminant les branches non prometteuses de l'arbre de décision.

— Permet une exploration plus rapide et efficace, surtout pour des jeux avec une grande profondeur d'arbre.

4 Implémentation du code

4.1 Structure du code source des versions console :

4.1.1 Introduction :

Cette section présente une vue d'ensemble du code source de notre projet de jeu de Moulin. L'objectif principal du code est de créer une intelligence artificielle capable de jouer au jeu de Moulin contre un joueur humain en utilisant des algorithmes de recherche tels que Minimax et Alpha-Beta. Le code est organisé de manière modulaire pour faciliter la compréhension, la maintenance et l'extension future du projet.

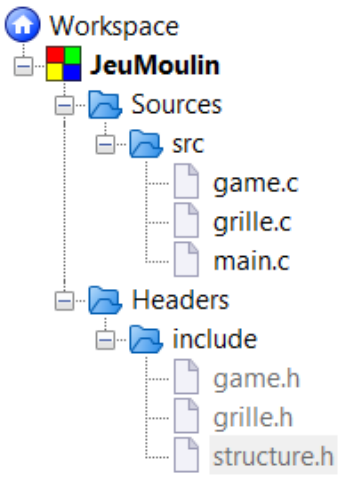
Nous avons divisé notre projet en quatre fichiers principaux :

— `structure.h`

— `game.c`

— `grille.c`

— `main.c`



Chacun de ces fichiers a un rôle spécifique et contribue à différentes parties de la fonctionnalité du jeu. Cette organisation permet de séparer les différentes préoccupations du projet, rendant le code plus lisible et plus facile à gérer. Dans les sections suivantes, nous décrirons en détail le et le rôle de chaque fichier.

4.1.2 Description des Fichiers

Fichier : `structure.c`

Le fichier `structure.h` définit les structures de données fondamentales utilisées dans le jeu de Moulin. Ces structures sont essentielles pour représenter l'état du jeu, les positions des pions, et la gestion des nœuds dans l'arbre de recherche des algorithmes Minimax et Alpha-Beta.

```c
#ifndef STRUCT_H_INCLUDED
#define STRUCT_H_INCLUDED

typedef struct Position
{
    char lettre;    int joueur;
    int couleur;
    char v1, v2, v3, v4; /// voisins de chaque position sur la grille
} Position;
typedef struct EtatJeu
{
    Position position[24];
    int nombrePionsMachine; /// nombre de pions sur table
    int nombrePionsHumain;
    int coutEtat;          /// cout de chaque etat
    int positionActuelle; /// position de chaque pion
    int precedent;
    int pionCaptureAdversaire; /// pion capture de l'adversaire

} EtatJeu;
typedef struct Noeud
{
    struct Noeud *suivant;
    EtatJeu Etat;
} Noeud;

static int nombreNoeuds = 0;

#endif // STRUCT_H_INCLUDED

```

Fonctions et Structures Déclarées

1. `typedef struct Position`

   Cette structure représente une position sur le plateau de jeu de Moulin. Elle contient les informations suivantes :

   — `char lettre` : Identifiant de la position.

   — `int joueur` : Indique quel joueur (humain ou machine) occupe cette position.

   — `int couleur` : Couleur du pion sur cette position.

   — `char v1, v2, v3, v4` : Voisins de chaque position sur la grille, facilitant la navigation et la vérification des règles du jeu.

2. `typedef struct EtatJeu`

   Cette structure représente l'état global du jeu à un moment donné. Elle contient :

   — `Position position[24]` : Tableau des 24 positions du plateau.

   — `int nombrePionsMachine` : Nombre de pions de la machine sur le plateau.

   — `int nombrePionsHumain` : Nombre de pions du joueur humain sur le plateau.

   — `int coutEtat` : Coût associé à cet état du jeu (utilisé pour l'évaluation dans les algorithmes de recherche).

   — `int positionActuelle` : Position actuelle du pion en cours de déplacement.

— `int precedent` : Position précédente du pion avant son déplacement.

— `int pionCaptureAdversaire` : Indique si un pion adverse a été capturé dans cet état.

3. `typedef struct Noeud`

   Cette structure représente un nœud dans l'arbre de recherche. Elle contient :

   — `struct Noeud *suivant` : Pointeur vers le nœud suivant dans la liste (utile pour implémenter une liste chaînée de nœuds).

   — `EtatJeu Etat` : L'état du jeu associé à ce nœud.

4. `static int nombreNoeuds`

   Cette variable statique compte le nombre de nœuds créés. Elle est utilisée pour suivre la taille de l'arbre de recherche et peut aider à la gestion de la mémoire ou à l'analyse de la performance des algorithmes.

Fichier : `grille.c`

Le fichier `grille.c` contient les fonctions nécessaires pour gérer l'affichage et les interactions avec le plateau de jeu de Moulin. Il comprend des fonctions pour afficher le plateau, obtenir les choix des joueurs et démarrer le jeu.

```c
#ifndef GRILLE_H
#define GRILLE_H

#include "../include/structure.h"

int ObtenirChoixInitial();

void AfficherPlateau(EtatJeu e);
void DemarrerJeu(EtatJeu e);
int ObtenirChoixUtilisateur(int Joueur, EtatJeu *e);
void AfficherPion();
extern int NbPionsPoses;

#endif // GRILLE_H
```

Fonctions Déclarées

1. `int ObtenirChoixInitial()`

   Cette fonction permet d'obtenir le choix initial de l'utilisateur pour déterminer qui commence le jeu.

2. `void AfficherPlateau(EtatJeu e)`

   Affiche l'état actuel du plateau de jeu en utilisant la structure `EtatJeu`.

3. `void DemarrerJeu(EtatJeu e)`

   Initialise et démarre le jeu en utilisant l'état de jeu fourni.

4. `int ObtenirChoixUtilisateur(int Joueur, EtatJeu *e)`

   Obtient le choix de mouvement du joueur spécifié et met à jour l'état de jeu en conséquence.

5. `void AfficherPion()`

   Affiche un pion sur le plateau de jeu.

6. `extern int NbPionsPoses`

   Cette variable externe compte le nombre de pions posés sur le plateau, permettant de suivre l'évolution du jeu.

Fichier : `game.c`

Le fichier `game.c` contient les fonctions essentielles pour la logique du jeu de Moulin, y compris les algorithmes de recherche Minimax et Alpha-Beta. Il gère la prise de décision de l'IA et les différentes opérations nécessaires pour évaluer et manipuler l'état du jeu.

```c
#ifndef GAME_H
#define GAME_H
#include <stdbool.h>

#include "../include/structure.h"
int indice_pion(char c, EtatJeu e);
int PasMouvementMachine(EtatJeu c);
int PasMouvement2(EtatJeu c);
void copier(EtatJeu source, EtatJeu *copie);
EtatJeu h(EtatJeu c);
EtatJeu Minimax(EtatJeu e, int p, int pions_a_placer, bool TourMax);
EtatJeu AlphaBeta(EtatJeu e, int p, int pions_a_placer, bool TourMax, int a, int b);
int moulinForme(Position *UnPlateau, int position, int Pion);
int Voisinages(int source, int PositionTest, EtatJeu c, int joueur);
int pionLibreHorsMoulin(Position *P);
extern int nodes_explored_minimax;
extern int nodes_explored_alphabeta;
#endif // SDL2_H

```

Fonctions Déclarées

1. `int indice`$_{p}ion(charc, EtatJeue)$

   $Cette fonction retourne l'indice d'un pion sur le plateau en fonction de son identifiant$`char c et de l'e`

2. `int PasMouvementMachine(EtatJeu c)`

   Détermine si la machine ne peut plus effectuer de mouvement.

3. `int PasMouvement2(EtatJeu c)`

   Détermine si aucun joueur ne peut effectuer de mouvement.

4. `void copier(EtatJeu source, EtatJeu *copie)`

   Copie l'état du jeu `source` vers `copie`.

5. `EtatJeu h(EtatJeu c)`

   Applique une fonction heuristique pour évaluer l'état du jeu `c`.

6. `EtatJeu Minimax(EtatJeu e, int p, int pions`$_{a}p lacer, boolTourMax$)

   $Implémentel'algorithme Minimax pourdéterminerlemeilleurcoupàjouerpourl'IA.$

7. `EtatJeu AlphaBeta(EtatJeu e, int p, int pions`$_{a}p lacer, boolTourMax, inta, intb$)

   $Implémentel'algorithme Alpha-Beta pouroptimiserleprocessusderechercheduneilleurcoupp$

8. `int moulinForme(Position *UnPlateau, int position, int Pion)`

   Vérifie si un moulin est formé avec le pion à la position spécifiée.

9. `int Voisinages(int source, int PositionTest, EtatJeu c, int joueur)`

   Vérifie si deux positions sont voisines sur le plateau de jeu.

10. `int pionLibreHorsMoulin(Position *P)`

    Détermine si un pion est libre et hors d'un moulin.

11. `extern int nodes-explored-minimax`

    Variable externe pour compter le nombre de nœuds explorés par l'algorithme Minimax.

12. `extern int nodes-explored-alphabeta`

    Variable externe pour compter le nombre de nœuds explorés par l'algorithme Alpha-Beta.

Fichier : `main.c`

Le fichier `main.c` est le point d'entrée principal du programme. Il gère l'affichage de l'écran d'accueil, du menu principal et des règles du jeu, ainsi que la logique de jeu.

include\structure.h | *include\grille.h | src\grille.c | include\game.h | src\game.c | src\main.c

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   #include "../include/grille.h"
5   #include "../include/game.h"
6   #include "../include/structure.h"
7
8   #include <stdio.h>
9   #include <stdlib.h>
10  #include <windows.h>
11  #include <stdio.h>
12  #include <windows.h>
13  #include <stdlib.h>
14  #include <string.h>
15  #include <conio.h>
16  #include <time.h>
17
18  #define FALSE 0
19  #define TRUE 1
20
21  void displayWelcomeScreen()
22  {
84  void displayMenu()
85  {
104 void displayRules()
105 {
241 int main()
242 {
```

Fonctions Déclarées

1. `void displayWelcomeScreen()`

   Cette fonction affiche l'écran d'accueil du jeu avec un titre stylisé et les noms des développeurs.

2. `void displayMenu()`

   Cette fonction affiche le menu principal du jeu.

3. `void displayRules()`

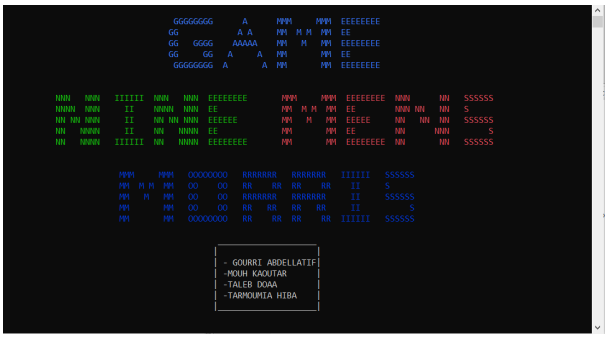
   Cette fonction affiche les règles du jeu.

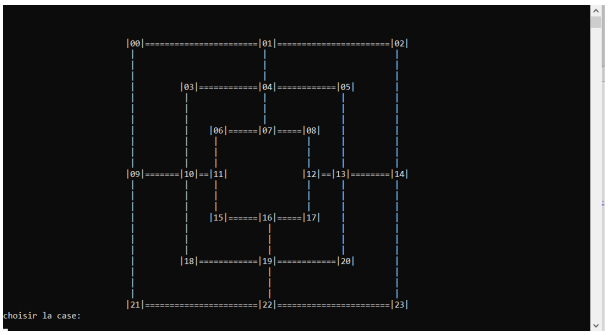
4. `int main()`

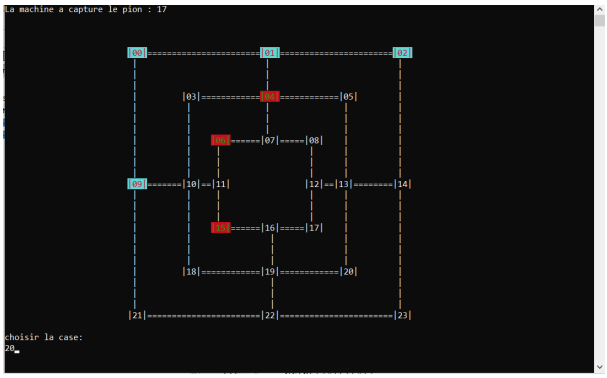
   La fonction principale du programme. Elle initialise le générateur de nombres aléatoires, puis entre dans une boucle infinie où elle affiche l'écran d'accueil et le menu principal. En fonction du choix de l'utilisateur, elle lance le jeu, affiche les règles ou quitte le jeu.

4.2 Résultat de la Version Console :

Dans cette section, nous présentons les résultats obtenus à travers la mise en œuvre du projet et des versions console par des images et des captures d'écran :







4.3 Conception de l'Interface Graphique :

4.3.1 Choix de la bibliothèque graphique :

Pour la conception de l'interface graphique de notre jeu de Moulin, nous avons opté pour la bibliothèque SDL2 (Simple DirectMedia Layer). SDL2 est une bibliothèque de développement multimédia puissante et polyvalente, largement utilisée pour créer des applications graphiques et des jeux. Elle fournit une interface simple pour gérer les graphiques, le son, les entrées de l'utilisateur et d'autres éléments multimédia. Elle est compatible avec de nombreux systèmes d'exploitation, y compris Windows, macOS, et Linux, ce qui assure une portabilité maximale

de notre application. Ces caractéristiques en font un choix idéal pour le développement de notre interface graphique, nous permettant de créer un jeu attrayant et performant.

```c
#ifndef SDL2_H
#define SDL2_H

#include "../include/SDL2/SDL.h"
#include "../include/SDL2/SDL_image.h"
#include "../include/SDL2/SDL_mixer.h"
#include "../include/SDL2/SDL_ttf.h"

#include "../include/struct.h"

void SDL_Inialiser();
void SDLmain(etat e);
int entrer_utilisateur(int Joueur,etat e);
void afficher();
int Nbr_des_pions_poser;

#endif // SDL2_H

```

FIGURE 3 – SDL2

4.3.2 Conception visuelle :

La conception visuelle de notre jeu de Moulin vise à offrir une expérience utilisateur immersive et agréable. Nous avons porté une attention particulière à l'esthétique du plateau de jeu et à l'interface utilisateur globale.

Écran de démarrage : L'écran de démarrage du jeu présente un paysage médiéval avec un château, immergeant les joueurs dans l'ambiance du jeu dès le lancement. Le titre du jeu "Nine Men's Morris" est affiché en grandes lettres blanches, faciles à lire, et centrées pour attirer l'attention des joueurs.

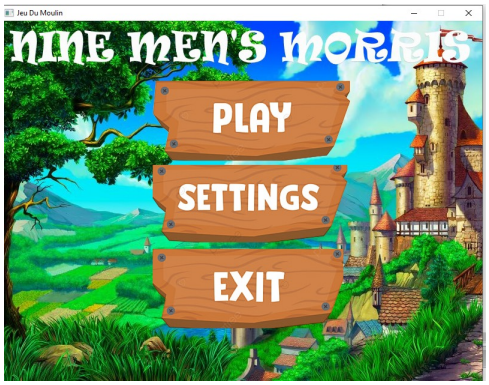


FIGURE 4 – Écran de démarrage

Plateau de jeu : Le plateau de jeu est conçu pour ressembler à un plateau en bois traditionnel, avec des lignes noires marquant les positions possibles pour les pions. Les intersections de ces lignes représentent les emplacements où les pions peuvent être placés. Une texture de bois est utilisée pour donner un aspect réaliste et agréable au plateau.

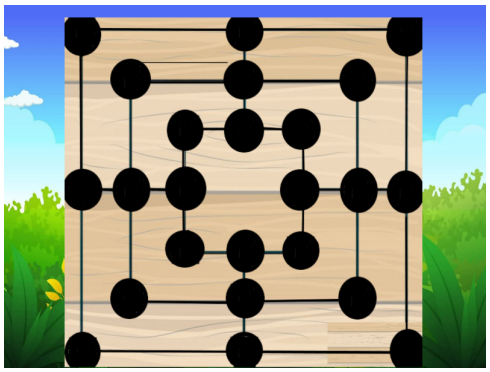


FIGURE 5 – Plateau de jeu de Moulin Au Début

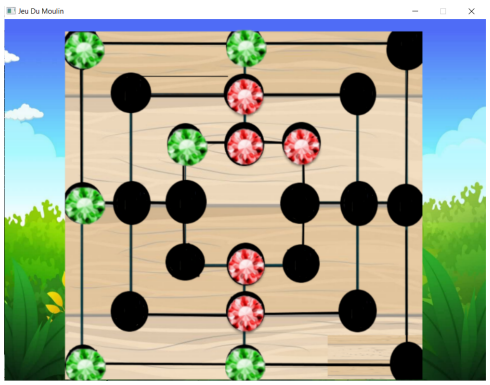


FIGURE 6 – Plateau de jeu de Moulin En Jouant

4.3.3 Implémentation technique

La mise en œuvre technique de l'interface graphique utilise les fonctionnalités de SDL2 pour dessiner le plateau de jeu, gérer les événements d'entrée de l'utilisateur, et afficher les différentes scènes du jeu (écran de démarrage, écran de jeu, etc.). Nous avons également intégré des textures et des images pour améliorer l'aspect visuel et rendre le jeu plus attrayant.

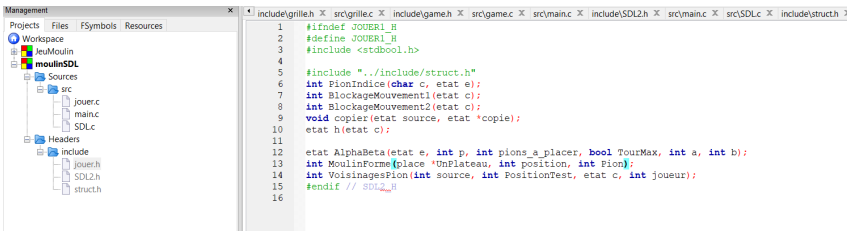


FIGURE 7 – Plateau de jeu de Moulin

5 Evaluation et performances :

5.1 Difficultées Rencontrées :

Lors de la réalisation de notre projet de jeu de Moulin, nous avons rencontré plusieurs problèmes techniques et de modélisation, particulièrement en utilisant les stratégies MiniMax et Alpha-Beta pour une solution optimale. Tout d'abord, l'installation des bibliothèques SDL (Simple DirectMedia Layer) et SFML (Simple and Fast Multimedia Library) a posé des défis importants. La compatibilité des versions avec le système d'exploitation et le compilateur, la configuration complexe des chemins d'accès et des variables d'environnement, ainsi que la documentation parfois insuffisante ont rendu cette étape difficile. De plus, des problèmes liés à l'installation des dépendances, à la liaison dynamique des bibliothèques, et aux incompatibilités entre différentes versions ont compliqué l'utilisation de SFML.

En ce qui concerne la modélisation du problème, la construction de l'arbre de recherche s'est avérée complexe. Implémenter correctement les algorithmes MiniMax et Alpha-Beta nécessitait une gestion rigoureuse des différents états du jeu et des transitions entre eux. La définition d'une profondeur de recherche optimale pour équilibrer la performance et la qualité de la solution était également un défi majeur. De plus, définir une fonction heuristique efficace pour évaluer correctement les positions intermédiaires du jeu et l'optimiser pour qu'elle soit rapide et pertinente a été difficile.

Dans la programmation en C/C++ en mode console, nous avons rencontré des problèmes de gestion de la mémoire, tels que des fuites de mémoire et des erreurs de segmentation dues à un accès incorrect à la mémoire. La gestion des entrées/sorties a aussi été un défi, nécessitant une validation rigoureuse des entrées utilisateur pour éviter les comportements imprévus et des difficultés à afficher le plateau de jeu de manière claire et lisible. De plus, les outils de débogage limités ont compliqué la traque des erreurs complexes.

Le développement de l'interface graphique avec SDL et SFML a également présenté des défis. La gestion des événements de la souris et du clavier, la synchronisation de l'affichage graphique avec les actions du joueur et les calculs de l'IA ont été problématiques. La conception d'une interface utilisateur intuitive et ergonomique, ainsi que la création d'un design visuel attractif mais fonctionnel, ont nécessité des ajustements. Enfin, l'optimisation des performances pour garantir une fluidité de jeu, en particulier lors des calculs intensifs de l'IA, et la gestion

des bugs graphiques lors du rendu des éléments du jeu ont été des aspects critiques.

5.2 Solutions Apportées :

Pour surmonter ces défis, nous avons consulté la documentation officielle et des tutoriels, mis en place des tests unitaires pour vérifier la validité du code et prévenir les erreurs, et optimisé le code pour améliorer les performances et réduire les bugs. Nous avons également ajusté l'interface graphique pour la rendre plus intuitive et esthétique. Ces efforts ont permis de résoudre de nombreux problèmes et d'améliorer l'expérience utilisateur globale, tout en identifiant des perspectives d'amélioration pour les futures versions du jeu.

5.3 Retour d'expérience des utilisateurs :

Lors de l'utilisation de notre jeu de Moulin, les utilisateurs ont partagé divers retours d'expérience, nous permettant de mieux comprendre les points forts et les aspects à améliorer de notre application. Les points positifs incluent une interface utilisateur intuitive et facile à naviguer, avec un design visuel simple et attrayant. Les joueurs ont apprécié la fluidité des mouvements et des transitions, ainsi que la performance de l'intelligence artificielle (IA) utilisant les stratégies MiniMax et Alpha-Beta, offrant un défi intéressant et varié. De plus, l'information sur le nombre de nœuds explorés après chaque coup a été jugée informative, et la mise à jour en temps réel de l'état du jeu réactive et précise. Cependant, certains utilisateurs ont rencontré des difficultés lors de l'installation et de la configuration des bibliothèques nécessaires, telles que SDL et SFML, suggérant qu'une documentation plus détaillée ou un installateur automatisé serait bénéfique. Ils ont également exprimé le désir de plus d'options de personnalisation, comme des niveaux de difficulté et des paramètres visuels modifiables, ainsi que des tutoriels plus détaillés pour les novices. Quelques utilisateurs ont observé des ralentissements lors des calculs intensifs de l'IA, indiquant qu'une optimisation supplémentaire pourrait améliorer la performance globale. Enfin, certains ont suggéré d'ajouter une plus grande variété de stratégies à l'IA pour rendre le jeu encore plus imprévisible et intéressant. Globalement, le retour d'expérience des utilisateurs a été positif, mettant en avant la fluidité du jeu, la performance de l'IA et l'ergonomie de l'interface graphique, tout en fournissant des pistes précieuses pour les futures mises à jour afin d'offrir une expérience de jeu encore plus enrichissante et accessible.

5.4 Performances du jeu :

Les performances du jeu de Moulin ont été largement saluées par les utilisateurs, notamment pour la fluidité des mouvements et des transitions sur le plateau de jeu. L'intelligence artificielle (IA), utilisant les stratégies MiniMax et Alpha-Beta, a montré une grande efficacité en offrant un défi varié et captivant. La capacité de l'IA à évaluer les positions et à prendre des décisions optimales a été particulièrement appréciée, rendant chaque partie intéressante et imprévisible.

Le jeu met à jour l'état en temps réel de manière réactive et précise, ce qui maintient l'engagement des joueurs. Cette réactivité est cruciale pour une expérience utilisateur immersive, où les actions des joueurs sont immédiatement reflétées sans décalage.

Cependant, certains utilisateurs ont noté des ralentissements occasionnels lors des calculs intensifs de l'IA, surtout avec des arbres de recherche profonds. Ces ralentissements peuvent affecter l'expérience de jeu, soulignant la nécessité d'optimiser davantage les algorithmes pour gérer efficacement la complexité sans sacrifier la performance.

Pour résoudre ces problèmes, des optimisations telles que la réduction de la profondeur de recherche, l'amélioration des heuristiques, ou l'utilisation de techniques de parallélisation pourraient être envisagées. En dépit de ces limitations, l'application offre une expérience de jeu fluide et agréable grâce à des algorithmes bien implémentés et à une gestion efficace des ressources. Les joueurs peuvent ainsi apprécier pleinement les stratégies de l'IA et l'interactivité du jeu, faisant du jeu de Moulin une expérience divertissante et enrichissante.

5.5 Suggestions d'Améliorations Futures :

Pour améliorer notre jeu de Moulin à l'avenir, plusieurs suggestions ont été formulées par les utilisateurs. Tout d'abord, l'optimisation des algorithmes de l'intelligence artificielle (IA), notamment MiniMax et Alpha-Beta, pourrait réduire les ralentissements observés lors des calculs intensifs. Utiliser des techniques telles que la parallélisation pour répartir les calculs sur plusieurs cœurs de processeur, et affiner les heuristiques pour des évaluations plus rapides et précises, contribuerait à améliorer significativement la performance globale du jeu.

Ensuite, l'ajout de niveaux de difficulté permettrait de mieux adapter le jeu aux compétences variées des joueurs. Un mode débutant pourrait offrir des défis plus simples avec une IA moins agressive, tandis que des niveaux intermédiaire et expert proposeraient des stratégies de plus en plus complexes et compétitives. Cette personnalisation rendrait le jeu plus accessible aux nouveaux joueurs tout en offrant un défi adéquat aux joueurs expérimentés.

L'intégration de tutoriels interactifs est également une amélioration clé. Ces tutoriels guideraient les nouveaux joueurs à travers les règles de base, les mouvements possibles et les stratégies avancées, en fournissant des exemples pratiques et des conseils en temps réel. Cela rendrait l'expérience de jeu plus éducative et agréable, en aidant les joueurs à se familiariser rapidement avec le jeu.

En outre, enrichir l'interface graphique avec des options de personnalisation augmenterait l'attrait visuel et l'engagement des utilisateurs. Offrir des thèmes visuels variés, comme des plateaux de jeu aux designs différents et des ensembles de pièces personnalisables, permettrait aux joueurs de personnaliser leur expérience de jeu selon leurs préférences. La possibilité de choisir entre différents fonds d'écran, styles de pièces et effets sonores ajouterait une dimension esthétique et immersive supplémentaire. _ Enfin, améliorer la documentation et simplifier l'installation du jeu seraient des aspects essentiels pour une meilleure accessibilité. Fournir un guide d'installation détaillé, accompagné de vidéos tutoriels, et développer un installateur automatisé qui gère les dépendances nécessaires (comme SDL et SFML) réduiraient les barrières techniques pour les utilisateurs. Une FAQ complète et un forum de support en ligne pourraient également aider à résoudre les problèmes courants rencontrés par les joueurs.

En implémentant ces suggestions, nous visons à offrir une expérience de jeu de Moulin plus fluide, personnalisée et accessible, enrichissant l'engagement des utilisateurs et répondant à une gamme plus large de préférences et de compétences. Ces améliorations contribueraient à rendre le jeu encore plus captivant et agréable pour un public diversifié.

6 Conclusion

La réalisation du projet de jeu de Moulin a été un parcours riche en défis techniques et conceptuels. Des difficultés ont émergé lors de l'installation des bibliothèques SDL et SFML, ainsi que dans la modélisation du problème, notamment dans la mise en œuvre des algorithmes MiniMax et Alpha-Beta. La gestion de la mémoire, les entrées/sorties et le développement de l'interface graphique ont également posé des défis significatifs.

Cependant, grâce à la consultation de la documentation, la mise en place de tests unitaires et l'optimisation du code, nous avons pu surmonter ces obstacles. L'ajustement de l'interface graphique pour la rendre plus intuitive a également amélioré l'expérience utilisateur.

Les retours d'expérience des utilisateurs ont été globalement positifs, mettant en avant la fluidité du jeu, la performance de l'IA et l'ergonomie de l'interface graphique. Des suggestions d'améliorations futures ont été formulées, notamment en matière d'optimisation des algorithmes, d'ajout de niveaux de difficulté et d'enrichissement de l'interface graphique.

En conclusion, malgré les défis rencontrés, le projet de jeu de Moulin a permis de développer une application fonctionnelle et divertissante. Les perspectives d'amélioration identifiées offrent des pistes pour enrichir davantage l'expérience de jeu et répondre aux attentes d'un public varié.