Présentation AAP

AUJOULAT Antoine
BALLIGAND Cassio
BLAIS Ryan

Introduction

Objectif du Projet :

Développer la structure d'un super morpion pour un tournoi.

Différents algorithmes s'affrontent dans des parties de super morpion.

Étapes de Développement :

1) Mise en Place de la Structure de Base du Super Morpion
- Construction initiale et configuration du jeu.

2) Développement de Trois Programmes Clés
- Création de programmes essentiels pour la fonctionnalité du jeu.
- Importance des programmes pour la compétitivité dans le tournoi.

Structures de base (morpion et super morpion)

```c
typedef enum {ROND , CROIX , VIDE} T_valeur;

typedef struct {
    T_valeur valeur;
} T_case;

typedef struct {
    T_case cases[MAXLEN];
    T_valeur trait;
    int jeu_fini; //0 si le jeu n'est pas fi
} T_grille;

typedef struct {
    int score;
    int position;
} Coup;
```

```c
typedef struct {
    T_grille * grilles;
    int dernier_coup_joue;
    int super_jeu_fini;
    T_valeur trait;
 } T_superGrille; // 9 grill


typedef struct {
    int score;
    char * position;
} SuperCoup;
```

Fonctions de Base

char_to_valeur : Convertit caractère en T_valeur (CROIX pour ‘x’, ROND pour ‘o’, VIDE pour autres).

int_to_char : Transforme un entier entre 1 et 9 en caractère pour positions sur plateau.

valeur_to_char : Convertit T_valeur en caractère (‘x’ pour CROIX, ‘o’ pour ROND, chiffre pour VIDE).

oppose : Obtient valeur opposée dans T_valeur, change de joueur après chaque coup.

valeur_to_int : Convertit T_valeur en entier (0 pour VIDE, 1 pour ROND, 2 pour CROIX).

Fonctions avancées et paramétrage

fen_to_grille : Convertit code FEN en plateau de jeu.

coup : Mise à jour du plateau, vérifie fin de partie, tour correct, case libre.

eval_fin : Évalue fin de partie (victoire, match nul, ou jeu continue).

print_grille : Affiche plateau de jeu (‘x’, ‘o’, point pour VIDE).

Programme 1 : tttree

Objectif : Visualisation du jeu de morpion en utilisant la notation DOT.

1. Traitement des Arguments :

Gère le premier argument (notation FEN du plateau de morpion).

2. Conversion FEN en Grille :

Utilise fen_to_grille pour transformer notation FEN en T_grille.

3. Gestion des Erreurs :

Vérifie si la conversion réussit, sinon signale erreur et termine programme.

Génération d'un morpion DOT

```
void grille_to_dot(T_grille grille) {
        static int nb_noeuds = 0;
        printf("m%d [shape=none label=<<TABLE border='0' cellspacing='10' cellpadding='10' style='rounded'
bgcolor='black'>", nb_noeuds);
        printf("<TR>\n");
        for (int i=0; i<3; i++) {
                printf("<TD bgcolor='white'>%c</TD>\n", valeur_to_char_bis((grille.cases)[i].valeur));
        }
        printf("</TR>\n");
        printf("<TR>\n");
        for (int i=3; i<6; i++) {
                printf("<TD bgcolor='white'>%c</TD>\n", valeur_to_char_bis((grille.cases)[i].valeur));
        }
        printf("</TR>\n");
        printf("<TR>\n");
        for (int i=6; i<9; i++) {
                printf("<TD bgcolor='white'>%c</TD>\n", valeur_to_char_bis((grille.cases)[i].valeur));
        }
        printf("</TR>\n");
        if (grille.trait == ROND) printf("<TR><TD bgcolor='red' colspan='3'>m%d</TD></TR></TABLE>>];\n", nb_noeuds);
        if (grille.trait == CROIX) printf("<TR><TD bgcolor='green' colspan='3'>m%d</TD></TR></TABLE>>];\n", nb_noeuds);
        nb_noeuds++;
}
```

Parcours en profondeur

Fonction récursive qui parcourt l'arbre en profondeur jusqu'aux noeuds terminaux

```
void generate_dot_rec(T_grille grille, int noeud, T_valeur joueur, T_grille grille_initiale) {
    int k = maxdepth(grille,joueur) - depth(grille,grille_initiale);
    grille_to_dot(grille);
        if (eval_fin(grille) != 0) {return;}
        T_grille tab[MAXLEN];
    int nb_enfants = g(grille,joueur,tab);;
    T_grille * enfants = malloc(nb_enfants * sizeof(T_grille));
    enfants = genere_enfants_bis(grille,joueur,tab);
    int i;
    for(i=0;i<nb_enfants;i++) {
        printf("  m%d -> m%d;\n", noeud, noeud+i+1+k*i);
        generate_dot_rec(enfants[i], noeud+i+1+k*i, oppose(joueur), grille_initiale);
        k++;
    }

}
```

main.c

```c
int main(int argc, char *argv[]) {


        char *fen = argv[1];
        T_grille grille = fen_to_grille(fen);
        T_valeur joueur = grille.trait;

        if (grille.jeu_fini == -1) {
                fprintf(stderr, "Erreur lors de la conversion de la FEN en grille\n");
        }

        printf("digraph {\n");
        generate_dot_rec(grille, 0, joueur, grille);
        printf("}\n");
        return 0;
}
```

Programme 2 : sm-refresh

Objectif du Programme

Pouvoir jouer au super morpion contre le bot minimax basique.

Afficher de façon graphique et pseudo-graphique chaque mise à jour du super morpion.

Fonctions de Représentation Graphique

grille_to_dot_bis : Affiche grille normale de morpion en utilisant le langage DOT.

grille_gagnée_to_dot_bis : Affiche grille gagnée de morpion en utilisant le langage DOT.

Affichage graphique et pseudo-graphique

Pseudo-graphique

```
void superGrille_to_dot(T_superGrille superGrille) {
        printf("digraph  { a0 [shape=none label=<<TABLE border='0' cellspacing='10' cellpadding='10' style='rounded'
bgcolor='black'> <TR>\n");
        for (int i=0; i<9; i++) {
                if (superGrille.grilles[i].jeu_fini ==1 || superGrille.grilles[i].jeu_fini ==2) {
                        grille_gagnée_to_dot_bis(superGrille.grilles[i]);
                }
                else {
                        grille_to_dot_bis(superGrille.grilles[i]);
                }
        }
        printf("  </TR> </TABLE> >]; }\n");
}
```

Graphique

```
void print_super_grille(T_superGrille super_grille) {
    int i;
    int j;
    for(j=0;j<MAXLEN;j++) {
        for(i=0;i<MAXLEN;i++) {
                switch((super_grille.grilles[i/3 + 3* (j/3)]).cases[i%3 +3*(j%3)].valeur) {
                case CROIX : printf("X  ");break;
                case ROND : printf("O  ");break;
                default : printf("*  ");break;
            }
            }
        printf("\n");
    }
    printf("\n");
    //ensuite on affiche la grande grille en reprenant le code de print_grille
    //pour une grille de morpion classique
    int k;
    for(k=0;k<MAXLEN;k++) {
        switch(((super_grille.grilles[MAXLEN-1]).cases[k]).valeur) {
            case CROIX : printf("X  ");break;
            case ROND : printf("O  ");break;
            default : printf("*  ");break;
        }
        if ((k+1)%3==0) printf("\n");
    }
}
```

Calcul du meilleur coup pour le bot

On utilise l'algorithme minimax pour calculer le meilleur coup

Utilisation du type "Coup" pour récupérer la valeur du minimax + la position du coup à jouer

```
Coup minimax(T_grille node, int depth, T_valeur traitOrdi) {
    if (depth == 0 || estNoeudTerminal(node)) {
        return (Coup){evaluer(node, traitOrdi), -1}; }
    Coup val;
    T_grille tab[MAXLEN];
    int nombre_enfants = g(node, traitOrdi, tab);
    T_grille * enfants = malloc(sizeof(T_grille) * nombre_enfants);
    enfants = genere_enfants_bis(node,traitOrdi, enfants);
    int * num_enfants = genere_enfants(node, traitOrdi, tab);

    if(traitOrdi == node.trait) {
        val.position = -1;
        val.score = INT_MIN;
        int i;
        for(i=0;i<nombre_enfants;i++) {
            if (val.score < minimax(enfants[i],depth - 1, traitOrdi).score) {
                val.score = minimax(enfants[i],depth - 1, traitOrdi).score;
                val.position = num_enfants[i];
            }
        }
    }


    else {
        val.position = -1;
        val.score = INT_MAX;
        enfants = genere_enfants_bis(node,traitOrdi, enfants);
        int i;
        for(i=0;i<nombre_enfants;i++) {
            if (val.score > minimax(enfants[i],depth - 1, traitOrdi).score) {
                val.score = minimax(enfants[i],depth - 1, traitOrdi).score;
                val.position = num_enfants[i];
            }
        }
    }
    return val;
```

main.c

Transformation du coup d'échec en nombre comprenant le numéro de grille et le numéro de case

```c
int main(int argc, char ** argv) {
        int depth = argv[2][0] - '0';

        char * echecs = argv[1];
        char * coup = echecs_to_coup(echecs);
        int numero_grille = coup[0] - '0';
        int numero_case = coup[1] - '0';


        static T_superGrille superGrille;
        for (int i=0; i<MAXLEN; i++) {
                for (int j=0; j<MAXLEN; j++) {
                        superGrille.grilles[i].cases[j].valeur = VIDE;
                }
        }

        superGrille = coup_super_grille(superGrille, numero_grille, numero_case, superGrille.trait);

        char * coup_bot = meilleur_coup(superGrille, oppose(superGrille.trait), depth);
        superGrille = coup_super_grille(superGrille, coup_bot[0]-'0', coup_bot[1]-'0', oppose(superGrille.trait));

        print_super_grille(superGrille);
        superGrille_to_dot(superGrille);

        return 0;
}
```

Programme 3 : sm-bot

Introduction au Programme

Objectif :

Utiliser un algorithme negamax combiné avec un élagage alpha-beta pour calculer le meilleur coup possible.

Mettre en place une stratégie.

Utilisation du négamax avec élagage

```
Coup negamax(T_grille node, int depth, int alpha, int beta, T_valeur joueur) {
    if (depth == 0 || estNoeudTerminal(node)) {
        return (Coup){evaluer(node, joueur), -1}; } // Position est -1 pour indiquer qu'aucun co


    Coup val = {INT_MIN, -1};
    T_grille enfants[MAXLEN];
    int nombre_enfants = g(node, joueur, enfants);
    int * num_enfants = genere_enfants(node, joueur, enfants);

    for (int i = 0; i < nombre_enfants; i++) {
        if (-negamax(enfants[i], depth - 1, -beta, -alpha, oppose(joueur)).score > val.score) {
            val.score = -negamax(enfants[i], depth - 1, -beta, -alpha, oppose(joueur)).score;
            val.position = num_enfants[i];
        }

        if (val.score >= beta) {
            return val;
        }

        alpha = max(alpha,val.score);
    }

    return val;
```

Mise en place d’une stratégie

```
SuperCoup strategie(T_superGrille super_grille, T_valeur joueur, int depth) {
    int score_total = 0;
    SuperCoup super_coup = {0,-1};
    super_coup.score = INT_MIN;
    int scores[MAXLEN] = {0};
    for(int i = 0;i<MAXLEN;i++) {
        score_total = score_total + valeur_case(i) * minimax(super_grille.grilles[i],2,super_grille.trait).score *
negamax(super_grille.grilles[i],depth,INT_MIN, INT_MAX,joueur).score;
        scores[i] = valeur_case(i) * minimax(super_grille.grilles[i],2,super_grille.trait).score *
negamax(super_grille.grilles[i],depth,INT_MIN, INT_MAX,joueur).score;
        if(scores[i] > super_coup.score) {
            super_coup.score = scores[i];
            super_coup.position = i;
        }
    }
    return super_coup;
```

main.c

Conversion de la fen en super grille
Affichage du meilleur coup du bot sous forme de string

```c
int main(int argc, char ** argv) {
    char * fen = concat(argv[1],"\0");
    int time = argv[2] - '0';
    if(time <= 0) return 0;
    T_superGrille super_grille = fen_to_super_grille(fen);
    char * best_coup = meilleur_coup(super_grille,super_grille.trait,2);
    T_superGrille new_grille = coup_super_grille(super_grille,best_coup[0] - '0',best_coup[1] - '0',super_grille.trait);
    print_super_grille(new_grille);
    printf("%s",meilleur_coup(super_grille,super_grille.trait,2));
    return 0;
```

Perspectives d’amélioration

Défis à Relever programme 1

Gérer efficacement la structure complexe des données, en particulier pour les super grilles.

Optimiser l’algorithme pour une meilleure visualisation de l’arbre de jeu.

Problématique du Programme 2

Résolution du problème de segmentation fault affectant l’affichage semi-graphique de la super grille.

-> Comment ?

Débogage intensif et revue minutieuse du code.

Identification et correction des erreurs de gestion de la mémoire et des références invalides.

Vision du Programme 3 - Stratégie à développer

Mise en place d’une stratégie de jeu plus avancée et prédictive.

Anticipation des coups en tenant compte de la grille de jeu suivante.

Planification de mouvements à l’avance, basée sur les actions probables de l’adversaire.

Conclusion

Difficultés

Complexité de la programmation de la structure du super morpion.

Résolution des problèmes de segmentation fault.

Coordination et communication compliquées par la distance.