RAPPORT - PROJET DE C++

JEU DE PUISSANCE 4 AVEC INTERFACE GRAPHIQUE SOUS SDL ET INTELLIGENCE ARTIFICIELLE (ALGORITHME NEGAMAX)

Etienne Fontaine – Claire Rogier

ENSAE Deuxième année

1. Description de ce que fait le programme

Le programme est un jeu de Puissance 4 contre l'ordinateur. Trois niveaux d'intelligence artificielle sont proposés : facile, moyen et difficile. La difficulté est fonction de la profondeur utilisée pour l'algorithme du NegaMax (simplification de l'algorithme MiniMax). A la fin de chaque partie, le joueur a la possibilité de relancer une partie en pouvant changer de niveau de difficulté, le tout sans avoir à relancer le programme. A la fin de chaque partie, un compteur de victoires, défaites et matchs nuls rappelle le bilan du jeu à l'utilisateur. Le programme est présenté sous forme d'interface graphique, réalisée à l'aide de la librairie externe SDL. Une autre librairie, SDL_ttf est aussi utilisée pour pouvoir afficher du texte sur l'écran. Nous avons utilisé deux classes pour réaliser le programme : la classe Partie qui s'occupe en particulier de l'intelligence artificielle, et la classe Interface qui a pour rôle l'affichage des différents objets utilisés (grille, pions rouges du joueur, pions jaunes de l'ordinateur).

2. Problèmes rencontrés et solutions choisies

Un des premiers problèmes rencontrés a été l'installation de la librairie SDL, et plus particulièrement de SDL_ttf pour afficher du texte. De fait, l'interface graphique ne fonctionne pas de la même manière que la sortie console pour afficher du texte, d'où la nécessité d'utiliser SDL_ttf. Après avoir essayé à de multiples reprises d'installer SDL et SDL_ttf sur Visual Studio (en essayant de nombreux tutoriels sur Internet, tous sans succès), nous avons réussi à installer les librairies sous CodeBlocks, et c'est donc sur CodeBlocks que nous avons réalisé le projet.

Nous avons tout d'abord commencé pas coder la « base » du programme en utilisant une sortie console plutôt qu'une interface graphique, et en utilisant une stratégie aléatoire pour l'intelligence artificielle. Ceci étant réalisé, nous avons pu commencer à travailler sur l'interface graphique et sur l'intelligence artificielle.

Pour l'interface graphique, nous n'avons pas rencontré de problèmes particuliers, si ce n'est l'installation de SDL. Le problème venait plutôt du code en lui-même car nous n'arrivions pas à faire alterner le joueur et l'ordinateur pour poser un pion. Nous avons donc tout d'abord intégré l'intelligence artificielle (toujours sous forme aléatoire) dans chacun des cas possibles (c'est-à-dire les 7 colonnes disponibles pour le joueur). Nous avons ensuite réussi à simplifier le code en utilisant la classe Partie. D'autres problèmes nous ont posé des difficultés : réussir à arrêter le jeu quand 4 pions étaient alignés (utilisation d'un booléan pour y remédier) et affichage d'un compteur de victoires, défaites et matchs nuls (conversion d'un integer en char pour y remédier, la fonction cout de la console n'étant ici pas disponible).

Pour l'intelligence artificielle, il fallait déterminer un bon algorithme et nous avons choisi l'algorithme du NegaMax qui nous semblait bien convenir pour le jeu de Puissance 4.

3. Description de l'architecture du programme

Nous avons créé une classe **Partie** qui contient différents éléments :

- le plateau de jeu **_p** de 42 cases que l'on complètera au fur et à mesure de la partie par des 0 (case occupée par l'ordinateur) ou des 1 (case occupée par le joueur). La case en haut à gauche (resp. à droite) a pour coordonnées 0,0 (resp 0,6). Celle en bas à gauche (resp à gauche) 5,0 (resp 5,6).

- un entier **_compteur** qui va compter le nombre de stratégies qui aboutissent à une victoire pour le joueur ou l'ordinateur dans l'IA.

- un entier **_valeur** qui va compter le nombre de stratégies qui aboutissent à une victoire pour l'ordinateur dans l'IA tout en comptant négativement les stratégies qui aboutissent à une victoire pour le joueur dans l'IA. Notre IA a pour but de renvoyer un pourcentage de réussite pour l'ordinateur en faisant la division de _valeur par _compteur.

- un entier **_gain** qui compte le nombre de parties gagnées par le joueur.

- un entier **_perte** qui compte le nombre de parties gagnées par l'ordinateur.

- un entier **_nul** qui compte le nombre de parties qui aboutissent à un match nul.

- un bool **_fin** déterminant si un alignement de 4 pions ou plus a été réalisé

Le constructeur : **Partie(int c,int v, int g,int p,int nombre, bool f)**

On initialise chacun des éléments de la classe Partie.

On initialise le tableau _p à 10. 10 signifiera que la case est vide et peut donc être jouée.

Le destructeur : **Partie :: ~Partie()**

On libère la place occupée par le plateau du jeu _p.

Le constructeur copie : **Partie(const Partie &source)**

On copie toutes les informations concernant un autre objet de type Partie.

Les fonctions get : int **getCompteur()** renvoie _compteur.

int **getValeur()** renvoie _valeur.

int **getGain()** renvoie _gain.

int **getPerte()** renvoie _perte.

int **getNul()** renvoie _nul.

int **getFin()** renvoie _fin.

Les fonctions set : void **setCompteur(int c)** réinitialise _compteur à la valeur c.

void **setValeur(int v)** réinitialise _valeur à la valeur v.

void **setGain(int g)** réinitialise _gain à la valeur g.

void **setPerte(int p)** réinitialise _perte à la valeur p.

void **setNul(int nombre)** réinitialise _nul à la valeur nombre.

void **setFin(bool f)** réinitialise _fin à la valeur f.

Les fonctions add : void **addCompteur(int c)** ajoute c au _compteur.

void **addValeur(int v)** ajoute v à _valeur.

void **addGain(int g)** ajoute g à _gain.

void **addPerte(int p)** ajoute p à _perte.

void **addNul(int nombre)** ajoute nombre à _nul.

Les fonctions sub : void **subCompteur(int c)** enlève c au _compteur.

void **subValeur(int v)** enlève v à _valeur.

void Partie :: raz() réinitialise _compteur et _valeur à 0 car on va recommencer une partie de puissance 4. Dans la même logique, le tableau est réinitialisé à 10 dans toutes les cases pour signaler qu'elles sont vides.

int coupJouable(int j) renvoie la ligne (de 0 à 5) dans laquelle on peut jouer dans la colonne j. Si la fonction renvoie -1, soit la colonne n'existe pas, soit la colonne est remplie. Si la colonne n'est pas remplie, on parcourt la colonne en partant du haut et on s'arrête dès qu'on rencontre une case remplie. Une case est soit remplie à 0 (case occupée par l'ordinateur) soit à 1 (case occupée par le joueur).

bool Partie :: rempli () indique si le tableau est rempli ou pas. On parcourt pour cela tout le tableau et on regarde si une case est vide c'est-à-dire égale à 10.

int Partie :: vainqueur () renvoie 1 si le joueur a 4 pions alignés, 0 si l'ordinateur a 4 pions alignés et 10 s'il n'y a aucun gagnant. On cherche alors 4 pions alignés en horizontal, vertical, diagonale nord-est / sud-ouest ou diagonale nord-ouest /sud-est.

int IA()renvoie l'indice de la colonne dans laquelle l'ordinateur doit jouer. Pour cela, on regarde d'abord si l'ordinateur peut gagner la partie en jouant dans une colonne de 0 à 6. Si oui, on renvoie le numéro de colonne correspondant car l'ordinateur est assuré de gagner dans ce cas. On regarde ensuite si le joueur peut gagner la partie la prochaine fois qu'il jouera. On parcourt alors toutes les colonnes. Si le joueur peut gagner, on renvoie le numéro de colonne correspondant pour empêcher le joueur de remporter la partie. Si rien n'est renvoyé suite aux deux boucles for, on regarde quelle colonne est la meilleure stratégie pour l'ordinateur. On réinitialise pour cela à chaque fois _compteur et _valeur à 0. Si le coup est possible dans la colonne, on fait jouer l'ordinateur dans la colonne dite. On appelle alors la fonction Algorithme avec une profondeur de 1 pour déterminer la meilleure stratégie.

int Algorithme(int profondeur) va permettre d'obtenir les pourcentages de réussite de l'ordinateur _valeur/_compteur pour savoir dans quelle colonne il vaut mieux jouer. On commence par créer un tableau col qui va stocker pour chaque colonne dans quel ligne il est possible de jouer. -1 dans col[j] si la colonne j est remplie. L'entier n indique c'est au tour de qui de jouer. Si la profondeur est impaire, c'est au joueur de jouer donc n=1. Sinon, c'est à l'ordinateur de jouer donc n=0. On parcourt ensuite toutes les colonnes. S'il est possible de jouer dans la colonne j, on joue le coup en remplissant la case par n. On regarde alors si ce coup permet d'avoir une victoire. Si on a une victoire que ce soit pour le joueur ou l'ordinateur, on incrémente notre compteur de victoire _compteur. Si on a une victoire de l'ordinateur, on incrémente le compteur de l'ordinateur _valeur de 1 car c'est un bénéfice pour l'ordinateur. Si c'est au contraire une victoire pour le joueur, c'est un désavantage pour l'ordinateur donc on décrémente _valeur de 1. On arrête alors de s'aventurer dans la fonction Algorithme et on return. S'il n'y a pas de victoire, on continue de s'enfoncer dans la fonction Algorithme : si on est toujours en dessous de la profondeur souhaitée (stockée dans level : level vaut 5 au niveau1, 6 au niveau 2 et 7 au niveau 3), on recommence l'opération : on parcourt les colonnes. S'il est possible de jouer dans la colonne j, on joue le coup et on réappelle la fonction Algorithme avec une profondeur +1.

Nous avons aussi créé une classe **Interface** :

- Définition de la fenêtre SDL
- Zones en haut et en bas de la fenêtre pour couvrir le texte qui y sera affiché
- Images bmp de la grille du jeu et des pions du joueur et de l'ordinateur
- Coordonnées pour placer les pions sur la fenêtre de jeu

Fonction **initialisationSDL** : définit les objets écran (fenêtre de jeu), grille et pions pour le joueur (rouge) et l'ordinateur (jaune)

Fonction **afficherDuTexte** : définit une police (times ici) et affiche un texte donné à une position donnée

Fonction **pageDeDemarrage** : affiche la page de démarrage du jeu

Fonction **choixMenu** : permet de quitter le jeu depuis la page de démarrage ou de passer à la page suivante, qui sera la page de choix du niveau de difficulté

Fonction **choixNiveau** : demande le niveau de difficulté et permet de quitter le jeu ou de lancer la partie avec un niveau de difficulté donné

Fonction **choixColonne** : réceptionne le numéro de la colonne dans laquelle le joueur joue et lance la fonction jouer pour placer le pion si la colonne est vide. Teste aussi si la grille est remplie pour déclarer un match nul

Fonction **reinitialisationGrille** : affiche la grille de jeu sur l'écran ainsi que le numéro des colonnes et les instructions pour jouer

Fonction **reinitialisationPosition** : reblitte (afficher de nouveau) la grille (pour une question d'esthétisme)

Fonction **jouer** : permet de jouer un coup et de déterminer si ce coup permet au joueur ou à l'ordinateur de gagner la partie, auquel cas lance la fonction jeuFini et incrémente le compteur de victoires ou de défaites

Fonction **jouerUnCoupJoueur** : détermine la ligne vide en fonction de la colonne choisie par le joueur pour jouer, puis place ce pion sur la grille en fonction de sa position et donne une valeur à cette case dans le tableau (valeur égale à 1)

Fonction **jouerUnCoupOrdi** : détermine la ligne vide en fonction de la colonne choisie par l'IA pour jouer, puis place ce pion sur la grille en fonction de sa position et donne une valeur à cette case dans le tableau (valeur égale à 0)

Fonction **jeuFini** : efface le texte en haut et en bas de l'écran pour le remplacer par un message annonçant la victoire, la défaite ou le match nul et affichant le compteur de victoires, défaites et matchs nuls, ainsi que l'option rejouer ou quitter

FICHIER main.cpp

- Inclure les librairies SDL et SDL_ttf pour afficher le texte
- Création de variables globales utilisées dans le programme
- Fonction int main (int argc, char *argv[]) lançant SDL et SDL_ttf, affichant la page d'accueil, puis le choix du menu puis boucle while pour jouer à une ou plusieurs parties. Enfin, appel des destructeurs des classes Interface et Partie, fermeture de SDL et de SDL_ttf et fin du programme

4. Guide d'utilisation du programme

Le but du jeu est simple : réussir à aligner 4 pions ou plus de sa couleur (rouge dans le cas du joueur) en une ligne horizontale, verticale ou diagonale. Le premier à réussir cela a gagné. Le programme commence par un écran d'accueil expliquant à l'utilisateur comment quitter le programme ou comment continuer, vient ensuite l'écran de choix du niveau de jeu (facile, moyen ou difficile). Après sélection du niveau (grâce au pavé numérique), la grille du jeu s'affiche et la partie peut commencer. Le jeu se joue avec le pavé numérique en saisissant un chiffre entre 1 et 7 (1 pour la colonne la plus à gauche, 7 pour la colonne la plus à droite). En cas de victoire, défaite ou match nul, un message s'affiche en haut de la fenêtre et le compteur de victoires, défaites et matchs nuls du joueur est actualisé en haut à droite de la fenêtre. Le joueur peut ensuite relancer une partie en appuyant sur la touche 'Entrée' ou quitter le jeu en appuyant sur 'Echap' ou en cliquant sur la croix rouge en haut à droite.