Les 7 merveilles du monde des 7 couleurs

KABIRY François, FÉLIX Martial, THOBIE Niels

I - Introduction

Dans ce projet, nous avons codé le jeu des 7 couleurs de la célèbre compagnie Infogrames Entertainment © où 2 joueurs font une bataille de territoires pour imposer sa couleur au reste du plateau. Le but de ce projet est d'apprendre à coder en C de manière ludique et pédagogique en utilisant tous les outils que nous avons appris à maîtriser pendant cette UE.

II – Voir le monde en 7 couleurs

La première étape est de construire notre plateau de jeu. Nous commençons par créer une variable globale qui sera la taille de notre plateau, que nous supposons être un carré de dimension $N \times N$. Nous définissons la taille du monde grâce à la ligne *#define taille_monde N* et nous avons créé un tableau global de caractères : *monde* grâce à la ligne *char monde[taille_monde][taille_monde].*

Ensuite, nous avons créé en tableau global de caractères *couleur* qui comprend nos 7 merveilleuses couleurs que nous nommons A, B, C, D, E, F et G pour simplifier la prise en main du projet. Cela étant, nous plaçons nos deux joueurs ^ et *v* respectivement au coin en haut à droite et en bas à gauche de notre tableau *monde* soit respectivement à l'indice [0][taille_monde -1] et à l'indice [taille_monde -1][0].

Enfin, nous avons créé deux boucles *for* imbriquées parcourant chacune la taille du tableau N et permettant de remplir le caractère d'indice [i][j] de *monde* par un caractère aléatoire de la liste *couleur*. Pour cela, nous utilisons la fonction *rand()%nb_couleur* qui retourne un entier entre 0 et nb_couleur qui vaut 7 dans notre cas. Nous rajoutons une condition tel que nous ne remplaçons pas les cases du monde où nous avons placé initialement les 2 joueurs. Par soucis de lisibilité, nous synthétisons cette suite d'actions par une fonction que nous appelons *initialisation* qui ne retourne rien et qui a comme arguments d'entrée nos deux joueurs que nous définissons pour le moment comme des caractères. Juste après cette initialisation, nous définissons une fonction *affichage* ne retournant rien et n'ayant aucun argument d'entrée et qui va se contenter d'afficher chaque ligne et de faire un retour à la ligne afin d'afficher tout le tableau dans notre console grâce à la fonction *printf*. Afin d'afficher des couleurs nous avons utilisé des charactères spéciaux (« \e[0 ;37m ») qui permet de changer la couleur des charactères qui sont afficher après. Nous avons donc créé des fonctions qui impriment le charactère couleur puis le charactère que l'on veut afficher et enfin un charactère qui reset la couleur, puis nous avons créé une autre fonction qui imprime les lettres de la bonne couleur en fonction de quelle lettre ils ont.

A ce moment, nous sommes capables d'afficher le plateau dans la console. Maintenant, nous devons être capables de mettre à jour notre monde après qu'un joueur a effectué un coup. Pour cela, nous avons créé une fonction *majCoup(char couleur, char joueur),* prenant en argument d'entrée le joueur et la couleur que nous souhaitons remplacer, qui va transformer le monde de la manière suivante : le joueur choisi de jouer *couleur*, si *couleur* est adjacente à *joueur*, elle se transforme en *joueur*. La spécificité est que si cette *couleur* a également des cases *couleur* adjacentes (s'il y a un bloc de *couleur* adjacent à une case *joueur*), alors tout ce bloc se transforme en joueur. Afin de résoudre ce problème, nous implémentons dans cette fonction une boucle while de la manière suivante : tant que toutes les cases *couleur* présentent sur le tableau ne sont pas adjacentes à des cases *joueur*, nous remplaçons les cases *couleur* par *joueur* et nous reparcourons le monde jusqu'à ce que cette condition soit respecté. Cette condition est vérifiée par un entier *test* qui va être modifié dès que nous changeons une case *couleur* en une case *joueur* et nous sortons de la boucle while si l'entier *test* reste inchangé après avoir parcouru le monde.

On peut vérifier cette fonction en affichant le monde après avoir joué un coup valide (il y a effectivement la couleur souhaitée à côté d'une case joueur) ou en vérifiant que le monde reste inchangé si le coup est invalide (

il n'y a pas la couleur souhaitée à côté d'un joueur). Le pire cas possible pour cet algorithme est dans le cas où la totalité du monde n'est constitué que d'un bloc d'une seule et unique couleur et que le joueur est initié en bas à droite du monde.

III - À la conquête du monde

Nous avons maintenant la possibilité de faire jouer un tour par un joueur. Il est donc temps à nos deux joueurs de partir à la conquête du monde. Afin de permettre aux joueurs de conquérir le monde, nous faisons une boucle demandant, à tour de rôle, aux joueurs, le coup qu'ils veulent jouer. Néanmoins, notre implémentation a, pour l'instant, le défaut de ne pas permettre à une partie de se finir, en effet, il n'y a aucune condition de victoire. Afin de savoir quand arrêter la partie, il suffit de connaître la proportion du territoire occupé par chacun des joueurs, en effet, quand un joueur contrôle plus de 50 % du territoire ou que les deux joueurs sont tous deux à 50 % (cas de l'égalité), la partie doit finir. Afin de calculer ce pourcentage, on modifie la fonction *affichage* afin qu'elle calcule le nombre de case *joueur* et qu'elle le divise par le nombre de case totale. Nous calculons donc ce pourcentage à la fin de chaque boucle, nous l'affichons après le plateau de jeu et enfin avons rajouté ces conditions.

Pour bien prendre en compte le pourcentage dans notre fichier main, nous avons pensé à créer une structure joueur qui aura plusieurs paramètres : son symbole, son nombre de case et son pourcentage. Avec cette structure, nous avons créé une fonction *createJoueur()* permettant de créer un joueur en allouant de la mémoire grâce à la fonction *malloc()* et permettant d'initialiser son nombre de case et son pourcentage à 0. Grâce à cette création de structure, nous avons modifié la fonction *affichage* pour qu'elle prenne en argument nos deux structures joueurs au lieu de caractères représentant nos joueurs afin qu'elle modifie directement les éléments des structures des deux joueurs. Nous avons également dû modifier la fonction *majCoup* pour qu'elle prenne en argument la structure d'un des deux joueurs.

IV – Stratégie de l'aléa

Pour pouvoir implémenter une intelligence artificielle, nous avons dû changer la manière par laquelle nous obtenions la couleur jouée par un certain joueur. Nous avons donc créé la fonction *getColor(*joueur)* qui renvoie la couleur à jouer durant ce tour, si le joueur est humain nous lui demandons la couleur qu'il veut rentrer, et si le joueur est une IA il renvoi une couleur au hasard. Pour que cela marche correctement nous avons donc aussi modifier la *struct joueur* pour qu'elle inclut une variable *int IA* qui décrit si le joueur en question est une IA ou pas. Pour permettre une meilleure lisibilité du code nous avons aussi créé une fonction *doRound(*joueur)* qui permet de jouer un tour de jeu en demandant une couleur avec *getColor(*joueur)* puis la joue avec *majCoup()* et qui est appelé alternativement avec chaque joueur en fonction du numéro de tour.

Nous avons ensuite voulu faire en sorte que cette IA aléatoire choisisse une couleur qui lui donnera au moins une case, pour faire cela, nous avons dû créer une fonction qui permet de prédire le nombre de case que va convertir un certain coup. Nous avons donc créé la *struct monde* qui contient la taille du monde, le tableau qui contient toutes les cases du monde ainsi qu'un deuxième tableau qui peut être modifier à volonté sans affecter le tableau principal. Ainsi nous allons pouvoir créer une fonction *prediction*() qui fonctionne comme majCoup() mais qui s'applique au deuxième tableau et qui renvoie le nombre de case convertit. Nous avions d'abord utilisé cette fonction en faisant une boucle *do … while* qui applique la fonction à une couleur aléatoire et qui sortait de la boucle si nous avions une prédiction non nulle. Mais nous avons dû changer la façon de faire les choses car si le joueur n'avait pas de coup valide, il se bloquait dans une boucle infinie. Finalement nous avons donc d'abord fait une liste avec les résultats des 7 prédictions et si cette liste est remplie de 0 on joue la couleur 'A' (qui ne fera rien du coup).

V – La loi du plus fort

Nous avons ensuite voulu implémenter une intelligence artificielle qui joue le coup qui lui rapporte le plus de case a ce tour ci. Pour cela, nous simplement utilisé la fonction prédiction décrit plus haut et chercher son max avec les 7 couleurs disponible. Nous avons ensuite mis à l'intérieur d'une fonction *playGame(int taille_monde, int IA1, int IA2, int Affichage_type)*. Cette fonction prend en argument d'entrée la taille du plateau,

si les joueurs sont des humains (alors IA vaut 0), des IA aléatoires (alors IA vaut 1) ou des IA glouton (alors IA vaut 2) et si on souhaite afficher le plateau (Affichage_type = 1) ou non (Affichage_type=0). Nous avons mis dans cette fonction toutes les instructions pour jouer une partie et nous avons lancer cette fonction 100 fois pour voir laquelle des deux IA est la meilleure. Les résultats sont clairs : l'intelligence artificielle aléatoire n'est pas de taille face à l'IA glouton.

VI – Les nombreuses huitièmes merveilles du monde

Nous avons ensuite implémenté une intelligence artificielle gloutonne prévoyante qui joue le premier coup de la meilleure combinaison de deux coups. Pour faire cela nous avons réutilisé la fonction *predictionMax()* que nous avions définis pour l'IA glouton et nous l'avons modifié pour inclure une double boucle qui test toute les combinaison de deux coup. Nous avons aussi fait que si deux combinaisons donnent le même nombre de case, on prendra celle qui donne le plus de case au premier coup pour éviter le cas où il n'y a qu'un coup possible (par exemple « F » mais la fonction prend la combinaison « A puis F » ce qui ne marche pas). La complexité de la prédiction est $O(tailleMonde^2)$ donc faire une double boucle et 2 prédictions a une complexité de $O(nbCouleur^2 * tailleMonde^2)$. Si l'on veut faire une prédiction sur les n prochain coup il faut appeler la fonction prédiction n fois (on peut facilement le faire avec une boucle) en utilisant les valeurs d'une liste de couleur que l'on incrémenter comme un nombre de base 7.

Nous avons aussi implémenté une IA hégémonique qui cherche à maximiser son périmètre. Malheureusement cela n'a pas été très efficace car elle peut laissez des cases toutes seules dans son territoires qui incrémente le périmètre de 4. Il aurait fallu enlever les cases se trouvant entouré de 4 case de la couleur joueur mais nous n'avons pas pu le faire par manque de temps. Cette IA pourrait être implémenté de manière plus optimisé mais nous n'avons pas pu le faire pas manque de temps.

VI – Conclusion

Nous avons pu, tout au long de ce projet, créer notre plateau de jeu, faire en sorte que 2 joueurs puissent s'affronter ou bien qu'un joueur affronte une IA avec différents algorithmes de stratégie. Nous aurions aimé aller plus loin afin d'imaginer différents modes de jeu ou différents algorithmes pour nos IA mais les différents changements que nous avons dû apporter à nos fonctions tout au long des nouveaux ajouts nous a pris beaucoup de temps. Cependant, nous avons su rendre hommage à la compagnie qui a développé ce jeu : Infogrames Entertainment © et nous continuerons à vous fournir du contenu additionnel en DLC.

Bibliographie

https://replit.com/@MartialFelix/Projet-7-couleurs

https://github.com/Francokab/Projet-7-couleurs