PROJET AP

HASAMI SHOGI

IS3 - Promo 2024

LE BRIS Maxime et DEBOUVRIES Ryan

Présentation du projet

Dans le cadre du développement de nos compétences et de l'apprentissage du langage C nous devons réaliser une version simplifiée du jeu de plateau Hasami Shogi. Il s'agit d'un jeu se jouant sur un plateau carré de 81 cases avec des pions blancs et noirs. Chaque joueur dispose de 18 pions disposés sur deux lignes. Le but du jeu est de capturer les pions adverses de manière rectiligne. Si un joueur n'a plus que 5 pions ou moins alors il a perdu et la partie est terminée, c'est aux pions blancs de commencer. Ce projet nous permettra de pratiquer le travail en équipe (binôme) ainsi que de peaufiner nos connaissances de manière ludique.

Cahier des charges

Pour ce projet, nous devions écrire un code C permettant à deux utilisateurs de jouer au Hasami Shogi. Le code doit permettre d'afficher en début de partie et à chaque début de tour, le plateau de jeu afin de permettre de voir la position des pions des deux équipes. Pour chaque tour, le programme doit lire le déplacement d'un pion d'une case à une autre. Lors d'un déplacement, le code doit modifier le plateau de jeu s'il y a eu capture, c'est-à-dire que deux pions de la même couleur encadrent des pions de la couleur opposée adjacents. La partie est terminée lorsqu'une équipe a au plus 5 pions sur le plateau de jeu.

Analyse du problème et résolution

Afin de répondre à ces différentes demandes du code nous aurons plusieurs sous-programmes :

- Le sous-programme *creationPlateau*, correspond à la première tâche qui consiste tout simplement à initialiser le plateau à son état de départ de la partie soit deux lignes par couleur de part et d'autre du plateau et des lignes vides entre les deux camps.

- Le sous-programme *lectureCoup*, celui-ci vérifiera si le déplacement peut être effectué en contenant une boucle tant que. Tant que le déplacement est impossible, la saisie sera demandée à nouveau. Le déplacement sera possible si c'est le tour du joueur (pion bougé de la bonne couleur) (nous utiliserons un switcher pour cela), si le déplacement est rectiligne, s'il n'y a pas de pion sur le chemin du déplacement et si le point d'arrivée est bien vide. S'il y a un pion adjacent au pion déplacé, on peut le « sauter » mais on doit s'arrêter à la case suivante.

- Le sous-programme *capture*, celui-ci permettra de rechercher à travers les quatres sens possible en partant du dernier pion déplacé si il y a un autre pion de la même couleur et si oui de prendre le plus proche. Ensuite il vérifiera si entre les deux pions il n'y a que des pions adverses et si oui effectuera une "capture".

- Le sous-programme *affichage*, lui permettra d'afficher le plateau avec les chiffres et lettres de coordonnées ainsi que les pions suite aux changements effectués. On affichera également le nombre de pions de chaque joueur. Un de nos choix important effectué est le fait que nous créons "artificiellement" l'affichage des coordonnées de cases.

Pour ce qui est de la fonction *main*, elle contiendra une boucle while qui s'arrêtera lorsque l'un des nombre de pions devient inférieur ou égal à 5. Dans ce cas, on vérifie le nombre de pions de chaque joueur pour indiquer le vainqueur.

Description des structures utilisées

Pour la réalisation du projet, notre code C comporte 2 structures comme évoqué dans l'analyse du projet.

La première structure est le type "Case", elle comporte deux éléments:
- occupé: cette donnée fonctionne comme un opérateur booléen, elle vaut 1 donc VRAI si la elle est occupée et 0 sinon donc FAUX. Nous avons défini en amont deux constantes "VRAI" et "FAUX" qui valent respectivement 1 et 0
- couleur: cette donnée permet de savoir quel joueur occupe la case ou bien si aucun des deux ne l'occupe. Elle vaut donc "B" ou "N" selon si un pion blanc ou noir l'occupe, et elle vaut "." s'il n'y pas de pion.

La seconde structure est le type "Plateau", elle possède 3 éléments:
- nbBlanc: cet élément va stocker le nombre de pions restant à l'équipe blanche et servira donc à déterminer l'issue de la partie.
- nbNoir: idem que pour nbBlanc, on stocke ici le nombre de pions noirs afin de déterminer l'équipe gagnante.
- jeu: jeu est une matrice de taille 9x9 (comme le plateau de jeu) qui constitue notre plateau de jeu.

Nous avons en effet décidé de stocker les nombres de pions dans une structure afin de pouvoir y accéder rapidement et de ne pas lancer une création de variables à chaque exécution du programme.
Enfin, le fait d'avoir une matrice de cases nous permet de stocker à la fois l'occupation d'une case mais aussi le pion qui l'occupe.

Mise en algorithme

Pour commencer voici la création de nos structures (écriture en pseudo-code).

structure Case:
occupé : Booléen
couleur : Caractère
fin

structure Plateau:
jeu : Matrice [TAILLE][TAILLE] de Case
nbBlanc : Entier
nbNoir : Entier
fin

- Sous-programme *creationPlateau*:

Le but de ce sous-programme est d'initialiser le Plateau avec les pions dans leur position de départ, puis lorsque c'est fait de retourner le Plateau.
Voici donc les informations que nous manipulons:

Locale: plateau: Plateau - Plateau de jeu qui sera retourné par la fonction
Locale: i: Entier - Indice de parcours de ligne
Locale: j: Entier - Indice de parcours de colonne

Résolution: Ce sera donc une fonction. Pour les 2 premières lignes de la matrice, on remplit couleur par "N" et on passe la valeur de occupé des cases à VRAI. Pour les 2 dernières lignes, on remplit couleur par "B" et on passe la valeur de occupé à VRAI. Enfin, pour les cases d'indices i allant de 3 à 7, on remplit couleur par "." et on donne la valeur FAUX à occupé.
Le parcours de la matrice se fait grâce aux indices i et j.

- Sous-programme *lectureCoup*:

Le but de ce sous-programme est de lire un déplacement saisi par un utilisateur puis de l'effectuer s'il est valide (c'est-à-dire s'il respecte les règles du jeu).
Voici donc les informations que nous manipulons:

Donnée/Résultat: jeu: Matrice[9][9] de Cases - matrice du plateau de jeu
Résultat: coup: Vecteur[4] d'entiers - coup saisi par l'utilisateur

Donnée: tour: caractère - switcher que nous utilisons pour connaître la couleur qui doit jouer

Locale: saisie_caracteres: Chaîne de caractères - variable de récupération de la saisie utilisateur avant traitement.

Locale: i: indice de parcours de ligne

Locale: j: indice de parcours de colonne

Résolution: Ce sera donc une action. Tout d'abord, le programme va demander une saisie de l'utilisateur tant que le déplacement sera impossible, c'est-à-dire que

- si ce n'est pas un pion de la bonne couleur qui est déplacé
- la case d'arrivée ne sera pas vide
- le déplacement ne sera pas rectiligne c'est-à-dire que l'on ne se déplace pas que selon la colonne ou la ligne de la matrice et s'il y a un pion sur le chemin
- s'il y a un pion adjacent et que la case d'arrivée n'est pas la case suivant le pion sauté

A chaque lecture de saisie, étant donné que la saisie est du type *LettreChiffre,LettreChiffre,* nous convertissons les lettres en chiffre via le code ASCII et nous supprimons la virgule. Ainsi, nous pouvons plus facilement vérifier les conditions du déplacement dans notre matrice.

Une fois les conditions de déplacement validées, nous parcourons la matrice pour nous rendre sur la case d'arrivée souhaitée par l'utilisateur pour y modifier (puisqu'il s'agit d'une Case) l'occupation (donc occupé passe à VRAI) et la couleur (le "." est remplacé par la couleur du pion déplacé). Puis nous vidons la case de départ.

- Sous-programme *capture*:

Le but de ce sous-programme est à partir du dernier coup joué de repérer un autre pion de la même couleur en parcourant les deux directions et leurs deux sens respectifs (positifs et négatifs), puis une fois repéré de vérifier si la capture est possible. C'est à dire qu'entre les deux pions il n'y a que des pions adverses.
Voici donc les informations que nous manipulons:

Donnée: tour: Caractère - switcher permettant de vérifier la couleur qui doit jouer

Donnée: coup - Vecteur[4] d'entiers: coup effectué par l'utilisateur

Donnée/Résultat: plateau: Plateau - Plateau de jeu

Locale: i: Entier - Indice de parcours de ligne et de colonne

Locale: pos: Entier - Position courante d'un pion trouvé pour la capture

Résolution: Ce sera donc une action. Le programme va réaliser les mêmes étapes quatres fois, mais dans quatres sens différents en partant du dernier pion déplacé. Dans le sens positif et négatif de la ligne, puis dans le sens positif et négatif de la colonne (vers le haut, le bas, la droite et la gauche).

La première étape est de parcourir les cases jusqu'à ce qu'on est soit trouver un pion de la même couleur qui sera donc le plus proche, soit atteint la limite du plateau, dans ce cas on s'arrête là car il n'y a pas d'autre pion dans ce sens.

La suivante est, si on a trouvé un pion, de parcourir l'espace entre ces deux pions et de vérifier qu'aucune case n'y est vide, ce qui impliquerait que cet espace est rempli de pions adverses, sinon on s'arrête ici.

La dernière étape sera alors de réaliser la capture, c'est-à-dire rendre vide les cases entre les deux pions et décompter les pions enlevés au compteur de l'équipe à qui on vient de les enlever.

Ici la difficulté à été de bien réfléchir à ce qu'on devait parcourir car, il y avait de nombreuses erreurs de bornes à prévoir, nous avons donc dû y passer un peu de temps pour être sûr de ne pas oublier ou prendre trop de case.

Le sous-programme est donc le suivant

```
Action capture(plateau, coup, tour):
   D: coup: Vecteur[4] d'entiers
   D: tour: Caractère
   D/R: plateau: Plateau
   L: i: Entier
      pos: Entier

   {Parcours de la ligne sens positif}
   i <- coup[3]+1
   Tantque ((i < 10) ET (plateau.jeu[i][coup[4]].couleur != tour)) Faire
      i <- i+1
   Fait
   Si (i < 10) Alors
      pos <- i
      i <- coup[3]+1
      Tantque ((i < pos) ET (plateau.jeu[i][coup[4]].couleur != '.')) Faire
         i <- i+1
      Fait
      Si (i = pos) Alors
         Pour i allant de coup[3]+1 à pos-1 faire
            plateau.jeu[i][coup[4]].couleur <- '.'
            plateau.jeu[i][coup[4]].occupe <- FAUX
         Fait
```

```
        Si (tour='B') Alors
            plateau.nbNoir <- plateau.nbNoir - (pos-coup[3]-1)
        Sinon
            plateau.nbBlanc <- plateau.nbBlanc - (pos-coup[3]-1)
        Fin si
    Fsi
Fsi

{Parcours de la ligne sens négatif}
i <- coup[3]-1
Tantque ((i > 0) && (plateau.jeu[i][coup[4]].couleur != tour)) Faire
    i <- i-1
Fait
Si (i > 0) Alors
    pos <- i
    i <- coup[3]-1
    Tantque ((i > pos) && (plateau.jeu[i][coup[4]].couleur != '.')) Faire
        i <- i-1
    Fait
    Si (i = pos) Alors
        Pour i allant de pos+1 à coup[3]-1 faire
            plateau.jeu[i][coup[4]].couleur <- '.'
            plateau.jeu[i][coup[4]].occupe <- FAUX
        Fait
        Si (tour='B') Alors
            plateau.nbNoir <- plateau.nbNoir - (coup[3]-pos-1)
        Sinon
            plateau.nbBlanc <- plateau.nbBlanc - (coup[3]-pos-1)
        Fin si
    Fsi
Fsi

{Parcours de la colonne sens positif}
i <- coup[4]+1
Tantque ((i < 10) && (plateau.jeu[coup[3]][i].couleur != tour)) Faire
    i <- i+1
Fait
Si (i < 10) Alors
    pos <- i
    i <- coup[4]+1
    Tantque ((i < pos) && (plateau.jeu[coup[3]][i].couleur != '.')) Faire
        i <- i+1
    Fait
    Si (i = pos) Alors
        Pour i allant de coup[4]+1 à pos-1 faire
            plateau.jeu[coup[3]][i].couleur <- '.'
            plateau.jeu[coup[3]][i].occupe <- FAUX
        Fait
        Si (tour='B') Alors
            plateau.nbNoir <- plateau.nbNoir - (pos-coup[4]-1)
        Sinon
            plateau.nbBlanc <- plateau.nbBlanc - (pos-coup[4]-1)
```

```
            Fin si
        Fsi
    Fsi

    {Parcours de la colonne sens négatif}
    i <- coup[4]-1
    Tantque ((i > 0) && (plateau.jeu[coup[3]][i].couleur != tour)) Faire
        i <- i-1
    Fait
    Si (i > 0) Alors
        pos <- i
        i <- coup[4]-1
        Tantque ((i > pos) && (plateau.jeu[coup[3]][i].couleur != '.')) Faire
            i <- i-1
        Fait
        Si (i = pos) Alors
            Pour i allant de pos+1 à coup[4]-1 faire
                plateau.jeu[coup[3]][i].couleur <- '.'
                plateau.jeu[coup[3]][i].occupe <- FAUX
            Fait
            Si (tour='B') Alors
                plateau.nbNoir <- plateau.nbNoir - (coup[4]-pos-1)
            Sinon
                plateau.nbBlanc <- plateau.nbBlanc - (coup[4]-pos-1)
            Fin si
        Fsi
    Fsi
Faction
```

Avancement du projet et problèmes rencontrés

Nous avons bien gérer notre projet et en prenant un peu de temps hors des séances attribuées, nous avons pu finaliser le projet en moins d'une semaine. La première séance nous a permis de bien faire l'analyse du projet pour ensuite écrire une grande partie des sous programmes en pseudo-code car nous avons préféré écrire tous nos programmes en pseudo-code avant de les traduire en langage C. Au cours de la seconde séance nous avons également pu commencer à rédiger notre code C, notamment la création des structures, les sous-programmes les plus courts ainsi qu'une partie de la fonction main en attendant de finaliser les sous-programmes pour la terminer. Enfin, nous avons pris un peu de temps sur le Week-End pour rédiger le pseudo-code du sous-programme capture, sa traduction en code C pour pouvoir en début de semaine tester notre code avec les fichiers txt fournis sur le Moodle. Il s'en est suivi quelques corrections du code C afin que celui-ci soit opérationnel.

Les problèmes que nous avons pu rencontrer sont les quelques erreurs lors de la rédaction du code C. Mais principalement, le fait d'être en distanciel car nous avons tous deux eu des problèmes de santé nous empêchant de nous rendre aux séances prévues mais nous avons tout de même avancé à un bon rythme pour finaliser le projet.

Tests effectués

Afin de confirmer au mieux la fonctionnalité de notre projet nous avons tout d'abord effectué les tests mis à disposition par nos professeurs. Le premier liste des coups permettant de faire gagner les pions blancs et le second permet de faire gagner les pions noirs. Tous deux ont fonctionné dès le premier essai. Ensuite nous avons décidé de réaliser une partie testant toutes les fonctionnalités de nôtre application. Elle commence par tester le déplacement sur un autre pion, puis le déplacement en diagonale, ensuite elle essaie de déplacer un pion avec un pion sur le chemin. Puis nous faisons jouer les deux camps afin de réaliser une capture des pions blancs par les pions noirs d'une ligne de 6 pions dans la direction horizontale. Ensuite en continuant la partie nous réalisons une capture verticale et nous testons le fait que si un pion est entre deux pions adverses mais qu'il y a aussi des cases vides alors il n'y a pas de capture. Ce test a été concluant dès le troisième essai dû à diverses erreurs de bornes de parcours avec la première colonne et la dernière colonne. Désormais le jeu fonctionne parfaitement et nous ne trouvons aucune erreur en testant diverses parties nous même.

Conclusion

Pour conclure, lors de ce projet nous avons pu mettre en pratique nos connaissances afin de le mener à bien. De plus, nous avons su faire preuve d'une bonne capacité de communication afin de terminer ce projet très rapidement. Nous pouvons ajouter que notre capacité d'analyse de problème nous a été d'une grande aide pour prendre les bonnes décisions quant à la création de nos structures et fonctions. Certes nos fonctions de lecture de coup et de capture sont longues mais efficaces.

Ensuite, la plus grande difficulté de ce projet, également la partie la plus longue, a été d'identifier les différentes erreurs possibles lors des déplacements, et de traduire cela en langage de programmation avec un maximum d'efficacité. De plus, le choix que nous pouvons remettre en question est le fait que les coordonnées du plateau soient créées à chaque appel de la fonction d'affichage.

Pour finir, ce projet a été enrichissant pour les deux partis du binôme. Nous avons fait preuve d'une grande qualité de *pair working* afin de mener à bien ce projet.

The end