Projet: Hasami Shogi
(version simplifiée)

Tuteur : LIETARD Thibault

Étudiants : PIERRE Théo
REAU Vincent

Date : 21/01/2022

Sommaire

Introduction

Dans ce projet nous devons programmer une version simplifiée du jeu de plateau Hasami Shogi, ce programme sera une version ou deux joueurs humains s'affrontent. Nous devons travailler en équipe pour réaliser ce projet et respecter le cahier des charges.

Cahier des charges

Lors d'une partie, le plateau doit être modélisé (et initialisé), et les coups seront saisis au tour par tour par chaque joueur (avec un format précis de saisie à respecter). Les règles du jeu doivent être respectées (voir l'annexe pour le détail des règles, il y a donc plusieurs contraintes liées à la validité des déplacements de chaque joueur (Voir Analyse du problème).

Analyse du problème

La principale difficulté de ce projet est de contrôler la validité des coups saisis.

Pour chaque déplacement, il faut vérifier ces conditions :

-1: Les coordonnées saisies sont biens comprises dans le plateau ,
-2: La case de départ doit contenir un pion de la couleur du joueur qui se déplace,
-3: La case d'arrivée doit se situer sur la même ligne ou colonne que la case de départ (ce qui revient à dire que le déplacement est horizontal/vertical),
-4: la case d'arrivée doit être vide.

Il existe deux types de déplacements, les déplacements normaux et les sauts:

-Un saut est un déplacement de deux cases dont la case entre le départ et l'arrivée est non vide, il suffit donc de vérifier que la case d'arrivée est vide, ce qu'on fait déjà (4), pour qu'un déplacement soit un saut il faut donc juste vérifier qu'on se déplace de exactement deux cases

Pour un déplacement normal (sans saut) il faut vérifier que les cases entre le départ et l'arrivée sont vides

Pour que le déplacement soit valide il faut donc que le déplacement soit un déplacement normal ou un saut valide on peut donc utiliser une condition avec un OU.
la condition sera donc: (les cases intermédiaires sont vides OU la distance entre le départ et l'arrivée vaut exactement 2)

Une fois le déplacement contrôlé, il faut aussi vérifier si une capture s'est produite auquel cas on devra supprimer les pions capturés, il faut donc effectuer une vérification des captures après chaque déplacement.

Description des structures

Pour modéliser le tableau, on définit deux structures:

La structure Case qui contient:

-un booléen occupé, indiquant si la case est occupée

-un caractère valeur qui vaut 'B' 'N' ou '.' selon si la case est occupée par un pion blanc noir, ou si elle est vide

Explication:

Booléen occupé:

Cette variable n'est pas nécessaire puisqu'il suffirait de vérifier si le caractère valeur vaut '.' pour savoir si la case est occupée ou non, mais nous avons choisi de passer par cette variable car elle simplifie la compréhension de nos programmes.
Ce booléen sera donc utilisé pour vérifier si des cases sont occupées et donc de tester la validité d'un déplacement .Elle servira aussi pour vérifier si une capture a eu lieu.

Caractère valeur:

Ce caractère contient le symbole qui correspond à ce qui se trouve sur la case, donc 'B' 'N' ou '.'. Elle sert donc à représenter graphiquement les cases mais aussi pour toutes les vérifications de capture et de contraintes de déplacement.

La structure Plateau qui contient:

-Cas une matrice 9x9 de Case

-un entier nbN qui contient le nombre de pions noirs sur le plateau

-un entier nbB qui contient le nombre de pions blancs sur le plateau

Explication:

matrice Cas:

Cette matrice représente notre plateau de jeu, elle comporte donc des cases qui ont une valeur et un état, cette matrice devra être initialisé en début de partie et modifiée à chaque tour en mettant à jour les cases de départ et d'arrivée de chaque mouvement, et les cases capturées.

Entiers nbN et nbB:

Les entiers nbN et nbB correspondent au nombre de pions de chaque joueur. ils sont initialisés à 18 et serviront à savoir si on doit arrêter la partie (si nbN ou nbB est inférieur ou égal a 5) et savoir qui a gagné. Ils devront donc être modifiés à chaque capture.

Algorithme du sous programme gérant la capture:

```
Action capture( P, lcour, ccour, joueurcour)
|       Données: lcour: entier: ligne du pion qui vient d'être déplacé
|                 ccour: entier: colonne du pion qui vient d'être déplacé
|                 joueurcour: couleur du pion qui vient d'être déplacé
|
|       Données/Résultats: P: Plateau du jeu
|
|       Locales: l: entier: indice de parcours de ligne
|                c: entier  indice de parcours de colonne
|                nb: entier  nombre de pions du joueur adverse
|
|       {initialisation de nb}
|       Si (joueurcour='B') Alors
|       |       nb ← P.nbN;
|       Sinon
|       |       nb ← P.nbB;
|       Fsi
|
|       {initialisation de l pour la capture vers le haut}
|       l ←  lcour-1
|       Tant que ((l>=1) et (P.cas[l][ccour].occupe=VRAI) et (P->cas[l][ccour].valeur!=joueurcour)) Faire
|       |       l ← l-1
|       Ftq
|
|       Si  ( l>=1 et P.cas[l][ccour].valeur=joueurcour) Alors
|       |       nb ← nb-(lcour-l)+1;
|       |       Pour l allant de l+1 à lcour-1 Faire
|       |       |       P.cas[l][ccour].occupe←FAUX;
|       |       |       P.cas[l][ccour].valeur ←  '.';
|       |       FPour
|       FSi
|
|
|       {initialisation de l pour la capture vers le bas}
|       l ← lcour+1;
|       Tant que(l<=9 et P.cas[l][ccour].occupe=VRAI et P.cas[l][ccour].valeur!=joueurcour) Faire
|       |       l ← l+1
|       Ftq
|
|       Si (l<=9 et P.cas[l][ccour].valeur=joueurcour) Alors
|       |       nb ← nb-(l-lcour)+1;
|       |       Pour l allant de l-1 à lcour+1 avec un pas de -1 Faire
|       |       |       P.cas[l][ccour].occupe ← FAUX;
|       |       |       P.cas[l][ccour].valeur ← '.';
|       |       |
|       |       FPour
|       FSi
|
|
|       {vérification d'une capture vers la gauche}
|       c ← ccour-1
|       Tant que ((c>=1) et (P.cas[lcour][c].occupe=VRAI) et (P.cas[lcour][c].valeur!=joueurcour)) Faire
|       |       c ← c-1
|       Ftq
|
|       Si ((c>=1) et (P.cas[lcour][c].valeur=joueurcour))Alors
|       |       nb ← nb-(ccour-c)+1
|       |       Pour c allant de c à ccour -1Faire
|       |       |       P.cas[lcour][c].occupe ← FAUX
|       |       |       P.cas[lcour][c].valeur ← '.'
|       |       Fait
|       Fsi
|
|       {vérification d'une capture vers la droite}
|       c ← ccour+1
|       Tant que ((c<=9) et (P.cas[lcour][c].occupe=VRAI) et (P.cas[lcour][c].valeur!=joueurcour) Faire
|       |       c ← c + 1
|       Ftq
|
|
|       Si ((c<=9) et (P.cas[lcour][c].valeur=joueurcour)) Alors
|       |       nb ← nb-(c-ccour)+1
```

```
|       |       Pour c allant de c à ccour +1 Pas -1 Faire
|       |       |       P.cas[lcour][c].occupe ← FAUX;
|       |       |       P.cas[lcour][c].valeur ← '.'
|       |       Fait
|       Fsi
|
|
|       {Attribution du nombre de pions du joueur adverse à la nouvelle valeur de nb}
|       Si (joueurcour='B') Alors
|       |       P.nbN ← nb;
|       Sinon
|       |       P.nbB ← nb
|       Fsi
FAction
```

Avancement du projet

Première partie:

Lors de la première séance nous avons d'abord effectué une analyse du sujet en définissant comment nous allions mettre en forme ce jeu, nous avons donc défini les structures, puis nous avons réfléchi aux contraintes à respecter vis à vis des déplacements. La principale difficulté de cette séance était de bien choisir la manière dont nous allions organiser nos sous-programmes.Nous avons ensuite réalisé les sous-programmes *remplissage_Plateau* et *affichage_Plateau* et rédigé le début du compte rendu.

Deuxième partie:

Lors de la deuxième séance nous avons chacun réalisé des sous-programmes qui sont des vérifications de validité des mouvements comme par exemple les fonctions *joueurTour* ou *arrivée_vide.* La principale difficulté était de savoir comment contrôler un saut, nous avons choisi de vérifier que la distance entre le départ et l'arrivée était de 2, en effet un saut sera toujours un déplacement d'une distance de deux cases en ligne droite, ainsi pour n'importe quel mouvement, en ce qui concerne les cases intermédiaires, il faut que ces cases soient vides sauf dans le cas ou le déplacement est de deux cases, dans ce cas, peu importe si la case intermédiaire est occupée ou non puisqu'on peut sauter par dessus (il faut bien sur que la case d'arrivée soit vide).

Troisième partie:

Nous avons décidé d'avancer chez nous en plus des séances de cours afin de pouvoir poser des questions si nécessaire, il nous restait à réaliser la capture des pions, pour cela nous avons eu plusieurs idées mais la plus simple que nous ayons trouvé est de contrôler après chaque déplacement si ce nouveau pion déplacé à engendré une capture, ce qui veut dire qu'il faut analyser les cases dans les 4 directions et vérifier si une séquence correspond à une capture.

Quatrième partie:

Nous avons fini notre programme à la fin de la troisième séance, pour cette dernière séance nous nous sommes donc répartis les tâches restantes comme la réalisation des fichiers test et l'écriture en pseudo-code de notre sous-programme de capture.

Organisation générale:

Nous réalisons nos programmes en parallèle en partageant notre code à chaque nouvelle version, nous commentons nos fonctions directement quand nous les réalisons sauf si nous pensons qu'elles peuvent être améliorées. Dans ce cas, nous attendons de trouver une manière de l'optimiser avant de la commenter.

Nous n'avons pas ressenti d'étapes vraiment difficiles durant ce projet, nos erreurs étaient presque toujours dues à un mauvais

indice dans la matrice, ce qui se résout facilement avec quelques tests.

Tests effectués

Afin de tester notre programme, nous avons créé un test pour vérifier le fonctionnement de chaque sous-programme. Cela permet de tester tous les sous-programmes individuellement et de voir si ceux-ci sont opérationnels (après chaque coup invalide nous avons effectué un coup valide pour éviter une erreur de segmentation).

Liste des tests effectués:

-test_dansPlateau.txt (respect des limites du plateau)
-test_joueur.txt (le pion de départ est bien un pion du joueur)
-test_arriveVide.txt (la case d'arrivée n'est pas occupée)
-test_verti_hori.txt (le déplacement se fait uniquement selon une direction)
-test_cheminVide.txt (les cases entre le départ et l'arrivée sont vides)
-test_saut (on teste si le saut fonctionne et que le saut n'est faisable qu'au dessus d'une seule case)

Afin de tester les différents sous-programmes en même temps, nous avons également réalisé un fichier test qui correspond à une partie englobant tous les tests nommé "test_main.txt" .

Listing commenté

Voir les commentaires des sous-programmes.

Conclusion

Pour conclure, nous avons trouvé ce projet intéressant et amusant à réaliser. Nous n'avons pas eu le sentiment d'être réellement bloqué durant celui-ci. Nous avons réussi à travailler ensemble grâce à de nombreux échanges lors des séances ce qui nous a permis d'éviter certaines confusions ou conflits de point de vue. Nous sommes satisfaits du travail que nous avons réalisé et nous pensons avoir répondu correctement aux consignes.

Annexe:

Règles du Jeu

Voici comment se présente le plateau:

```
    1  2  3  4  5  6  7  8  9
A   N  N  N  N  N  N  N  N  N
B   N  N  N  N  N  N  N  N  N
C   .  .  .  .  .  .  .  .  .
D   .  .  .  .  .  .  .  .  .
E   .  .  .  .  .  .  .  .  .
F   .  .  .  .  .  .  .  .  .
G   .  .  .  .  .  .  .  .  .
H   B  B  B  B  B  B  B  B  B
I   B  B  B  B  B  B  B  B  B
```

18 pions noirs

18 pions blancs

Déroulement du jeu :

Les pions blancs commencent, puis les deux joueurs se placent à chaque tour un pion de leur couleur.

Déplacements :

-Les pions se déplacent d'un nombre quelconque de cases horizontalement ou verticalement, mais jamais en diagonale.

-Un pion peut sauter par dessus un pion adjacent (ami ou adversaire) afin de se rendre sur la case inoccupée juste derrière (sans capturer le pion sauté).

Captures :

Un joueur capture un pion ou une série de pions adjacents de son adversaire par encadrement entre deux de ses pions (horizontalement ou verticalement), l'un de ces deux pions doit être celui que le joueur vient de déplacer. Les deux pions du joueur et le pion ou les pions de son adversaire doivent être adjacents.

Fin de la partie :

Un joueur gagne quand son adversaire n'a plus que 5 pions ou moins.