DORE Martin
DUPIRE Arthur

Spécialité IS, 3e année

Rapport Projet Algorithmique et Programmation

Tuteur : C. Rohmer

Année 2021-2022

Introduction et contexte

Il nous a été demandé, au sein de la spécialité Informatique et Statistique, de réaliser un projet Algorithmique et Programmation dans le cadre de notre formation à Polytech Lille.

Effectué en première année de cycle ingénieur, celui-ci doit nous permettre d'appliquer les connaissances théoriques et informatiques acquises durant l'Unité d'Enseignement Fondements Informatiques, et en particulier en programmation. Ce projet est donc la consécration de l'enseignement suivi dans cette matière.
Notre binôme est constitué de Martin DORE et Arthur DUPIRE, et notre tuteur est C. Rohmer.

Présentation du projet

Le sujet qui nous a été attribué s'intitule "Hasami Shogi". Il consiste, comme son nom l'indique, à programmer une version simplifiée du jeu Hasami Shogi.
De manière simplifiée, en voici les règles et le déroulement :

Chaque joueur dispose de 18 pions disposés en 2 lignes. Les blancs commencent, et le joueur déplace un de ses pions verticalement ou horizontalement. Il est possible de sauter par dessus un pion d'une case adjacente, qu'il soit de la même couleur ou non.
Si le déplacement d'un pion provoque l'encadrement de pions de couleur opposée entre deux pions de ce même pion déplacé, alors les pions de la couleur opposée concernés sont capturés, et la case qu'ils occupaient devient alors vide.

Analyse du projet

Le projet que nous avons à réaliser présente certaines problématiques que nous devons résoudre en réalisant des choix dans la conception du programme selon l'interprétation du sujet.

Nous avons alors réussi à percevoir les contours du travail qui nous est demandé en analysant le projet, avant même de commencer sa réalisation.

Tout d'abord, nous avons fait la liste des tâches à réaliser et des fonctionnalités à développer, dans l'ordre, afin d'obtenir un programme final qui permet à deux joueurs de réaliser une partie de Hasami Shogi. Nous obtenons la décomposition de la résolution suivante :

- Créer la grille, remplie avec les pions noirs et blancs de chaque côté et un centre vide (remplie par des points) ;

- Afficher la grille à l'écran en temps réel ;

- Permettre aux deux joueurs de déplacer leurs pions, l'un après l'autre ;

- Étudier si un pion déplacé en capture d'autres ;

- Réaliser et mettre à jour le compte du nombre de pions restant pour chaque joueur, à la suite de chaque coup joué ;

- Tester si victoire il y a en testant le nombre de pions restant ;

- Changer de joueur à la suite de chaque coup joué.

Nous avons fait le choix de créer une matrice de caractères de dimension 9 par 9 et d'afficher les indices des cases en dehors de celle-ci, comme nous l'avions fait en travaux pratiques dans le cadre du module Algorithmique et Programmation.

Nous avons décidé de ne pas utiliser de structures *case* pour identifier les cases du plateau de jeu, en effet, 3 caractères suffisent à décrire les états possibles de chaque case : **'.'** si la case est vide, **'B'**, si le pion sur la case est blanc et **'N'** s'il est noir.

Nous nous sommes également demandé si nous devions réaliser d'autres structures cartésiennes, en l'occurrence des structures Pion et Joueur.

Finalement, nous avons fait le choix de ne pas faire de structure Pion car nous avons jugé que la simple modification de la matrice était suffisante, et que manier des objets de type Pion ne présentait aucun avantage par rapport à modifier les valeurs de la matrice.

De même, nous avons fait le choix de ne pas créer de type Joueur, car, à l'exception de renseigner un nom (en attribut), on pouvait se passer de cet objet sans que cela rende l'écriture du programme plus difficile. A la place, une variable *player* de type entier permet aux sous-programmes de connaître le joueur en cours.

Explication des fonctions

Fonctions et Actions principales

- **Action : initialisation_plateau(plateau)**

Donnée / Résultat: plateau, Tableau [1,…,9][1,…,9] de caractères
Locales : i, j, Entiers
L'appel de cette fonction permet de positionner les pions blancs et noirs de chaque côté du plateau. Les variables locales i et j sont les indices de parcours du tableau de caractères.

	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

Plateau initialisé

À noter que l'affichage du plateau ci-dessus est rendu possible à l'aide de la fonction affichage_plateau, qui affiche la matrice 9x9 ainsi que les indices de ligne et de colonne.

- **Action changement_joueur(joueur)**

D/R : joueur, Pointeur d'entier
L'appel de cette fonction permet de modifier le joueur à l'issue d'un coup. Dans notre programme, il s'agit de la variable *joueur* qui est modifiée (soit 1 soit 2).

- **Fonction deplacement_possible**(plateau, player, caseDep, caseArr)**: booléen**

D : plateau, Tableau [1,...,9][1,...,9] de caractères
player, Entier
caseDep, Vecteur[2] de caractères
caseArr, Vecteur[2] de caractères

Locales : lArr, cArr, lDep, cDep, Entiers
indParcours, Entier
bloque, Entier

Les quatre variables locales lArr, cArr, lDep et cDep sont récupérées des deux vecteurs caseDep et caseArr. Elles ne sont pas indispensables, mais elles facilitent l'écriture et la lisibilité du programme.
indParcours sert d'indice de parcours dans la fonction, tandis que la variable de type entier bloque nous sert de booléen : elle prend les valeurs 0 et 1, mais nous utilisons les constantes globales TRUE et FALSE. C'est la valeur de cette variable bloque qui est retournée par la fonction.

Tout d'abord, avant d'écrire cette fonction, nous avons dressé la liste des conditions nécessaires pour qu'un déplacement soit possible :
- Le joueur bouge un pion qui lui appartient;
- La case d'arrivée est libre;
- La case se situe dans la même ligne ou colonne;
- Le pion ne rencontre pas de pion;
- Le pion saute un pion adjacent : il ne peut parcourir que deux cases.

Ainsi, nous avons vérifié que toutes les conditions étaient respectées dans la fonction deplacement_possible.

Nous testons tout d'abord que la case est sur le plateau, selon les coordonnées rentrées par le joueur.
Ensuite, on vérifie que la case de départ n'est pas vide.
Nous vérifions par la suite que le pion bougé par le joueur lui appartient bien, puis que la case d'arrivée est vide. Enfin, nous testons si le déplacement saisi s'effectue bien en ligne droite.

Une fois toutes ces vérifications satisfaites, nous testons si un pion bloque le déplacement : c'est la partie la plus longue de la fonction. Il faut en effet réaliser le test pour les 4 directions physiquement possibles. Pour chaque direction, le programme regarde si le pion est bloqué, et, si oui, s'il peut le sauter. Ainsi, si le pion est bloqué et ne peut pas sauter le pion concerné, alors le déplacement est impossible. Dans le cas contraire, le déplacement est réalisable.

- **Action deplacement_pion(plateau, caseDep, caseArr)**

D : caseDep, caseArr, Vecteurs [2] de caractères
D/R : plateau, Tableau [1,...,9][1,...,9] de caractères

Nous réalisons dans cette fonction les déplacements de pions saisis par les joueurs. En réalité, cette fonction n'est appelée dans la fonction principale main.c que si le déplacement est déterminé possible par la fonction deplacement_possible : il n'est pas utile de tester la faisabilité du déplacement directement dans deplacement_pion. Même si cela aurait été réalisable, nous avons fait le choix de réaliser le test en dehors de la fonction.

CAPTURE PION

Algorithme gérant la capture des pions :

Action capture_pion(plateau, caseArr, nbBlancs, nbNoirs)

Données : - caseArr, Tableau [2] de caractères

Données/Résultats : - plateau, Tableau [1,…,9] [1,…,9] de caractères

- nbBlancs, Entier

- nbNoirs, Entier

Locales : - indParcours, Entier

- pionCaptures, Entier

- lArr, cArr, Entiers

- pion, Caractère

lArr ← caseArr[0]

cArr ← caseArr[1]

pion ← plateau[lArr] [cArr]

pionCaptures ← 0

{Captures horizontales}

{Capture à droite}

indParcours ← cArr + 1

Tant que ((plateau[lArr][indParcours] != pion) **et** (plateau[lArr][indParcours] != ' . ') **et** (indParcours <8) **faire**

    indParcours ← indParcours + 1

fait

Si (plateau[lArr][indParcours] = Pion) **alors**

    Tant que (indParcours > cArr+1) **faire**

        Plateau[lArr][indParcours - 1]= ' . '

        indParcours ← indParcours – 1

        pionCaptures ← pionCaptures + 1

    fait

fsi

{Capture à gauche}

indParcours ← cArr - 1

Tant que ((plateau[lArr][indParcours] != pion) **et** (plateau[lArr][indParcours] != ' . ') **et** (indParcours >0) **faire**

    indParcours ← indParcours - 1

fait

Si (plateau[lArr][indParcours] = Pion) **alors**

    Tant que (indParcours < cArr - 1) **faire**

        plateau[lArr][indParcours + 1] = ' . '

```
            indParcours ← indParcours + 1
            pionCaptures ← pionCaptures + 1
        fait
fsi

{Captures verticales}
{Capture en bas}
indParcours ← lArr + 1
Tant que ((plateau[indParcours][cArr] != pion) et (plateau[indParcours][cArr] != ' . ' ) et (indParcours <8) faire
        indParcours ← indParcours + 1
fait
Si (plateau[indParcours][cArr] = Pion) alors
        Tant que (indParcours > lArr + 1) faire
            plateau[indParcours - 1][cArr] = ' . '
            indParcours ← indParcours - 1
            pionCaptures ← pionCaptures + 1
        fait
fsi

{Capture en haut}
indParcours ← lArr - 1
Tant que ((plateau[indParcours][cArr] != pion) et (plateau[indParcours][cArr] != ' . ' ) et (indParcours >0) faire
        indParcours ← indParcours - 1
```

```
fait
Si (plateau[indParcours][cArr] = pion) alors
        Tant que (indParcours < lArr - 1) faire
                plateau[indParcours + 1][cArr] = ' . '
                indParcours  ← indParcours + 1
                pionCaptures ← pionCaptures + 1
        fait
fsi



Si (pion= ' B ' ) alors
        nbNoirs ← nbNoirs – pionCaptures
sinon
        nbBlancs ← nbBlancs – pionCaptures
fsi
fin action
```

Fonctions et Actions de simplification

Nous avons fait le choix d'écrire plusieurs actions et fonctions qui, si elles ne sont pas indispensables, permettent de simplifier le code obtenu et d'autres sous-programmes. Elles ne participent pas directement à l'élaboration des fonctionnalités possibles lors de la partie entre les joueurs, mais elles nous sont utiles.

En effet, ce sont des fonctions intermédiaires qui nous permettent de décomposer des actions, des vérifications de condition, ou encore des calculs en dehors des fonctions qui, elles, rendent possible des fonctionnalités propres au jeu (déplacement de pion, capture,...).

Ces fonctions intermédiaires sont les suivantes :

- **Action affichage_joueur(joueur)**

D : joueur, Entier
VL : pion, Caractère
L'appel de cette fonction permet d'indiquer quel est le joueur devant effectuer une action, ainsi que sa couleur. Cette action n'est utile que pour les utilisateurs du programme.
Dans le programme, c'est la variable joueur qui est testée afin de savoir à qui c'est le tour, et la variable pion sert lors de l'affichage de la phrase d'information aux joueurs.

- **Fonction abs(x) : Entier**

D : x, Entier
Cette fonction sert à renvoyer la valeur absolue de la variable donnée de type entier en paramètre : elle renvoie x si x>=0 et -x sinon. Ce sous-programme est appelé dans la fonction distance suivante :

- **Fonction distance(a,b) : Entier**

D : a, b, Entiers
Cette fonction permet de calculer la distance entre deux pions, en effectuant la soustraction entre les coordonnées auxquelles ils se trouvent. La fonction distance est utilisée via son appel dans la fonction déplacement possible, à quatre reprises.

- **Fonction estSurPlateau(a) : Booléen**

D : a, Entier

L'appel de cette fonction permet de renvoyer un entier (1 ou 0), qui désigne si le pion est sur le plateau. Nous avons défini au début du programme les deux constantes globales TRUE et FALSE, qui prennent respectivement les valeurs 1 et 0. Ainsi, nous pouvons utiliser cette fonction comme si elle renvoie un booléen et non un entier, ce qui améliore la clarté de notre programme en le rapprochant d'un langage plus proche du nôtre.

- **Action demande_coord(caseDep, caseArr)**

D/R : caseDep, caseArr, Vecteurs [2] de caractères

Cette action demande à l'utilisateur les coordonnées de départ et d'arrivée, qui sont stockées dans des vecteurs de taille 2.

- **Action viderBuffer()**

Locale : c, Entier

L'appel successif de plusieurs *scanf* peut causer des problèmes : afin de les éviter, nous vidons les buffers après chaque appel de **demande_coord.**

Problèmes rencontrés :

Matrice 10x10

Nous avons initialement créé une matrice 10x10, dont la première ligne était composée de {1,2,3,4,5,6,7,8,9} et la première colonne {A,B,C,D,E,F,G,H,I}. Cependant, nous avons par la suite été confronté à divers problèmes résultants de ce choix, tel que la perception par l'algorithme de l'indice de ligne B comme étant un pion blanc. Comme nous nous sommes rendus compte de ce programme tardivement, il a fallu modifier les fonctions **initialisation_plateau, affichage_plateau** ainsi que les bornes des indices de parcours dans toutes les autres parties du programme.

Pions qui se mangent “tout seul”

Après avoir effectué de nombreux tests pour trouver les limites de notre programme, nous nous sommes aperçu que dans certains cas, la fonction **capture_pion** capturait tous les pions du joueur adverse si ces derniers se trouvaient sur les bords du plateau. En ajoutant des restrictions (0<...<8 comme le plateau est une matrice 9x9) sur les indices de parcours de la fonction, nous avons pu résoudre ce problème.

Avancement du projet

Afin de travailler en simultané sur le programme, nous avons fait le choix d’utiliser l’outil *repl.it*.
De cette façon, nous nous sommes dispensés des problèmes de versionnage des fichiers et des erreurs que cela peut engendrer. En outre, le fait de travailler en en même temps sur le programme nous a permis d’avancer très rapidement sur les sous-programmes les plus simples (création/affichage du plateau…) et ainsi passer plus de temps sur les fonctions de déplacement et capture de pion qui sont plus complexes. La contrepartie du travail du travail synchronisé est que cela nécessite une bonne communication afin que le programme reste cohérent dans la syntaxe et que les actions de l’un n’engendre pas d’erreur sur le travail de l’autre.
A la fin de chaque séance, nous avons téléversé notre fichier sur moodle afin que notre tuteur puisse suivre nos progrès.

➔ Séance 1

Analyse du projet et création des fonctions simples : affichage du plateau et initialisation et rédaction de rapport d’analyse.

➔ Séance 2

Création de l’action de déplacement et tests sur des déplacements simples, début de l’écriture des actions déplacement_possible et capture_pion.

➔ Séance 3

Finalisation de déplacement_possible et modification de capture_pion et apparition des premiers bug liés au choix initial de matrice.

➔ Séance 4

Modification complète du programme avec la nouvelle matrice et fin du programme. Apparition des problèmes dans les captures de pions.

➔ Séance 5

Debug complet du programme et résolution des problèmes.

➔ Travail en dehors des séances

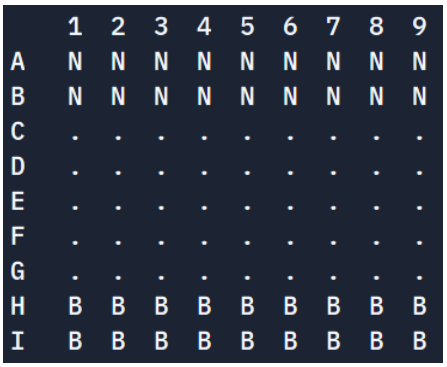
Rédaction du rapport et ajouts de fioritures dans le programme (Commentaires sur l’ensemble des sous-programmes, vérification de l’indentation…)

Tests des fonctions

Tests simples

➔ **Initialisation du plateau et affichage**

On peut voir que l'affichage se fait de manière lisible et que le plateau est dans la bonne configuration pour commencer la partie.

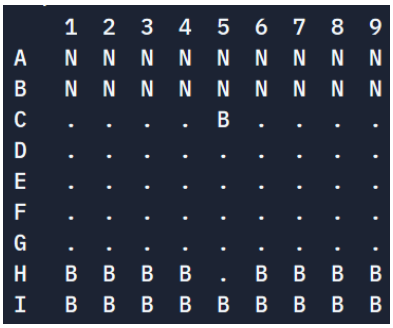


	1	2	3	4	5	6	7	8	9
A	N	N	N	N	N	N	N	N	N
B	N	N	N	N	N	N	N	N	N
C	.	.	.	.	.	.	.	.	.
D	.	.	.	.	.	.	.	.	.
E	.	.	.	.	.	.	.	.	.
F	.	.	.	.	.	.	.	.	.
G	.	.	.	.	.	.	.	.	.
H	B	B	B	B	B	B	B	B	B
I	B	B	B	B	B	B	B	B	B

Plateau initialisé

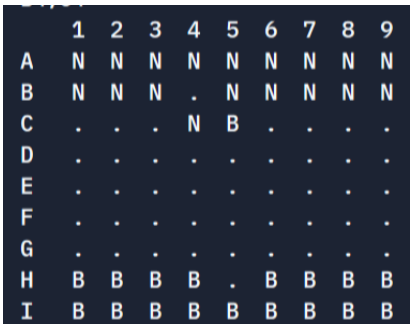
➔ **Demande de coordonnées et déplacements simples :**

Dans cet exemple, les blancs jouent H5,C5 puis les noirs jouent B4,C4. Les déplacements des deux joueurs s'effectuent sans encombre de manière verticale.



	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	.	.	.	.	B	.	.	.	.
D	.	.	.	.	.	.	.	.	.
E	.	.	.	.	.	.	.	.	.
F	.	.	.	.	.	.	.	.	.
G	.	.	.	.	.	.	.	.	.
H	B	B	B	B	.	B	B	B	B
I	B	B	B	B	B	B	B	B	B

Plateau avant déplacement

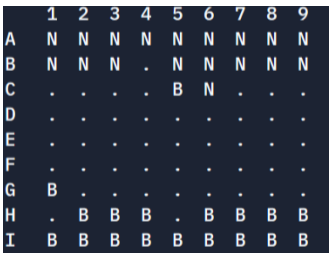


	1	2	3	4	5	6	7	8	9
A	N	N	N	N	N	N	N	N	N
B	N	N	N	.	N	N	N	N	N
C	.	.	.	N	B	.	.	.	.
D	.	.	.	.	.	.	.	.	.
E	.	.	.	.	.	.	.	.	.
F	.	.	.	.	.	.	.	.	.
G	.	.	.	.	.	.	.	.	.
H	B	B	B	B	.	B	B	B	B
I	B	B	B	B	B	B	B	B	B

Plateau après déplacement

➔ **Saut de pion :**

En suivant les coups précédents, les noirs jouent C4,C6. Ce déplacement permet de vérifier à la fois le saut de pion ainsi que le déplacement horizontal.

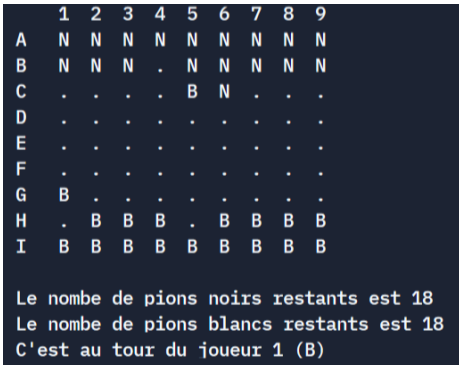


Plateau après le saut du pion noir

Tests des limites

Tests des déplacements possibles

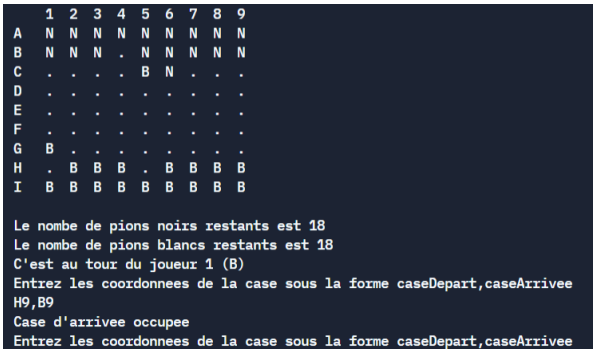
Le but de cette partie est de donner au programme une suite des déplacements impossibles, de vérifier qu'ils ne sont pas effectués et que les bons messages d'erreur s'affichent. Dans chaque cas, nous partirons de la *situation 1* :



Situation 1

➔ **Déplacement sur une case occupée**

En indiquant le déplacement H9,B9 théoriquement impossible au programme, nous obtenons le message d'erreur suivant :



Message erreur 1

Le programme indique alors à l'utilisateur de saisir à nouveau un déplacement, et ce jusqu'à ce qu'il soit possible. C'est également le cas pour les erreurs suivantes. Toutefois, dans un souci de lisibilité, nous avons préféré couper l'image au niveau de l'erreur concernée.

➔ **Déplacement depuis une case vide**

Dans le cas suivant, le déplacement n'est pas possible car aucun pion n'est situé sur la case de départ.

```
C'est au tour du joueur 1 (B)
Entrez les coordonnees de la case sous la forme caseDepart,caseArrivee
H5,E5
Case de depart vide
```

Message erreur 2

➔ **Déplacement pas en ligne droite**

Le message d'erreur suivant s'affiche car le déplacement saisi par le joueur ne correspond pas à un déplacement en ligne droite.

```
Entrez les coordonnees de la case sous la forme caseDepart,caseArrivee
G1,E6
Deplacement en ligne droite uniquement
```

Message erreur 3

➔ **Déplacement d'un pion qui n'est pas de la bonne couleur**

Contrairement à précédemment, l'erreur suivante s'affiche car la case de départ comprend bien un pion, mais qui n'est pas de la couleur du joueur à qui c'est le tour.

```
Entrez les coordonnees de la case sous la forme caseDepart,caseArrivee
C6,F6
Vous ne pouvez bouger que vos propres pions
```

Message erreur 4

➔ **Déplacement en dehors du plateau**

Ce message d'erreur s'affiche lorsque le déplacement d'un pion demande une sortie du plateau :

```
Entrez les coordonnees de la case sous la forme caseDepart,caseArrivee
G1,M45
Une des coordonnees n'est pas sur le plateau
```

Message erreur 5

➔ **Déplacement bloqué par un autre pion**

Lorsque le déplacement demandé par un joueur est impossible en raison d'un pion qui bloque celui-ci, l'interface affiche le message :

```
Entrez les coordonnees de la case sous la forme caseDepart,caseArrivee
H2,H5
Un pion bloque le deplacement
```

Message erreur 6

Tests des captures

➔ Capture simple

Suite aux coups H4,C4 des blancs puis B3,C3 des noirs, nous observons que la capture des deux pions blancs en C4 et C5 s'est bien réalisée, conformément aux règles du Hasami Shogi :

```
Entrez les coordonnees de la case sous la forme caseDepart,caseArrivee
B3,C3
   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
C  . . N . . N . . .
D  . . . . . . . . .
E  . . . . . . . . .
F  . . . . . . . . .
G  B . . . . . . . .
H  . B B . . B B B B
I  B B B B B B B B B

Le nombe de pions noirs restants est 18
Le nombe de pions blancs restants est 16
C'est au tour du joueur 1 (B)
```

Capture de pions

On observe également que le nombre de pions blancs restants a été mis à jour dans le message d'information : initialement de 18, il est désormais de 16.

➔ Capture en croix

Dans une partie plus avancée, il est possible de réaliser une capture dite "en croix", c'est-à-dire que la capture se fait dans trois directions différentes. Il est impossible de capturer dans la direction d'où le pion arrive : c'est donc le maximum de directions de capture possible.

Voici le résultat obtenu, avant et après capture, suite au coup I4,G4 :

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

Le nombe de pions noirs restants est 18
Le nombe de pions blancs restants est 18
C'est au tour du joueur 2 (N)
```

Plateau de jeu avant capture "en croix"

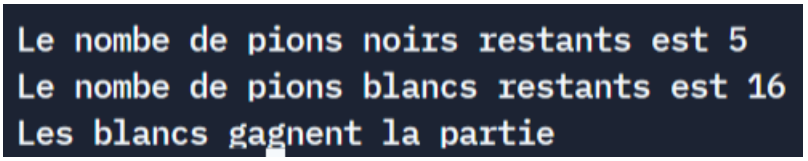
```
   1 2 3 4 5 6 7 8 9
A  N N N . N N N N N
B  N . N . N N . N .
C  . . . . B . . . N
D  B . . N . . . . .
E  . . . . . . . . B
F  . . . . . . . . B
G  . N . N . . N . .
H  . B . . . . B . B
I  B B B . . B B . B

Le nombe de pions noirs restants est 18
Le nombe de pions blancs restants est 13
C'est au tour du joueur 1 (B)
```

Plateau de jeu après capture "en croix"

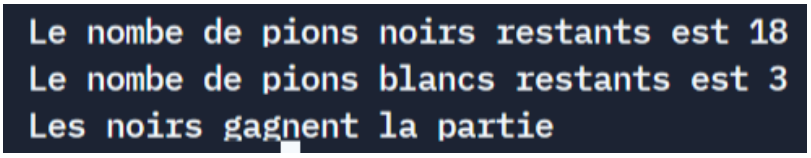
Nous relevons également que le nombre de pions blancs a été mis à jour, conformément au nombre de captures de pions réalisées.

Test d'une partie complète

Nous testons également notre programme avec les fichiers texte *victoireBlancs.txt* et *victoireNoirs.txt* . On peut voir que la trentaine d'actions s'effectue sans encombre et qu'à chaque étape, les règles du jeu sont respectées. A la fin de la partie, nous obtenons ces résultats :



victoire des Blancs



victoire des Noirs

Conclusion

Pour conclure, ce projet à permis de concrétiser les enseignements du module *Algorithmique et Programmation*. En effet, afin de modéliser une partie de Shogi nous avons dû faire appel à des matrices, vecteurs ou autres pointeurs. Travailler en binôme sur ce genre de projet est d'autant plus enrichissant car cela permet de mettre en commun des compétences, d'avoir une autre vision et de résoudre des problèmes plus complexes que l'on n'aurait pas forcément su résoudre individuellement. C'est également bénéfique car cela nous initie au travail en groupe ainsi qu'aux contraintes engendrées.

Bilan personnel : Martin Doré

Pour ma part j'ai beaucoup apprécié réaliser ce projet, en effet avec Arthur nous nous sommes bien répartis le travail et bien entendu dès le départ ce qui a facilité le bon avancement de ce dernier. Le projet était à la fois suffisamment simple pour que l'on puisse le réaliser en peu de temps mais également suffisamment compliqué pour nécessiter une bonne entraide et un travail rigoureux.

Bilan personnel : Arthur Dupire

J’ai particulièrement apprécié réaliser ce projet en binôme. En effet, celui-ci nous a permis de mettre en application des savoirs acquis dans le cadre du module Algorithmique et Programmation.

Si peu d’heures dans l’emploi du temps étaient consacrées à la réalisation de ce projet, nous avons pu réaliser une interface satisfaisante et aboutie, conforme aux résultats attendus.

Enfin, programmer et modéliser le jeu du Hasami Shogi en équipe m’a permis d’apprendre et de m’enrichir d’une méthode de travail différente de la mienne. J’ai également appris à me servir du site Replit, qui permet d’écrire du code C à deux simultanément sur un même projet.