HE2B ESI

Nom : _____

Prénom : _____

Groupe : _____

Identifiant : _____

/ 20

Haute École de Bruxelles-Brabant
École Supérieure d'Informatique
Bachelor en Informatique

22 mai 2017

Développement – 1ère

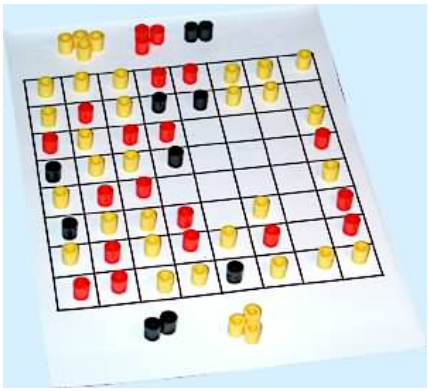
Examen Première session

Algorithmique

- *l'examen dure 3h ;*
- *sauf mention contraire, on peut considérer que les données lues ou reçues ne comportent pas d'erreurs ;*
- *veillez à rendre vos solutions modulaires. Ceci est* ***très*** *important.*

1 Description du jeu Butin (0 point)

i Règles du jeu



Nous allons réfléchir à quelques algorithmes nécessaires à la réalisation d'une version simplifiée du jeu *Butin*. Ce jeu, inspiré du jeu de dames, fait s'affronter 2 joueurs sur un plateau de jeu de 8×8 cases sur lesquels sont disposés aléatoirement 64 pions de 3 couleurs :

— 34 pions jaunes ;

— 20 pions rouges ;

— 10 pions noirs.

Le but du jeu est d'obtenir le maximum de points en enlevant des pions du plateau. Les pions ont les valeurs suivantes :

— 1 pion jaune vaut 1 point ;

— 1 pion rouge vaut 2 points ;

Version du document : 18 mai 2017

— 1 pion noir vaut 3 points.

Un joueur dont c'est le tour choisit un pion jaune. Il peut aussi, si c'est possible au départ du pion qu'il a choisi, capturer un ou plusieurs autres pions du plateau. Comme aux dames, on capture un pion en sautant au dessus si la case au delà est libre. Si, à la position où le pion preneur arrive, il est à nouveau possible de capturer un pion en sautant, cette capture est autorisée. Un saut ne peut être fait que selon une ligne ou selon une colonne. Un saut selon une diagonale est interdit. Chaque pion possède donc au maximum 4 cases voisines qui peuvent être occupées par des pions qui peuvent être capturés.

Autrement dit, à chaque tour, un joueur

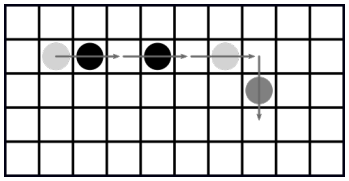
— choisit un pion jaune, le **pion preneur** ;

— capture 0, un ou, consécutivement, plusieurs pions en sautant par dessus si la case derrière le pion capturé est libre ;

— enlève le pion jaune preneur et tous les pions capturés du plateau de jeu.

Cette règle implique que si un joueur sélectionne un pion jaune qui ne peut prendre aucun pion en sautant par dessus, il doit simplement l'enlever du plateau de jeu. Un joueur n'est jamais bloqué à moins qu'il ne reste aucun pion jaune sur le plateau.

Dans l'exemple ci-contre le joueur sélectionne le pion jaune en position (1,1) et capture successivement 4 pions : 2 noirs, 1 jaune (gris clair) et 1 rouge (gris foncé).



Le jeu se termine lorsqu'un joueur retire le dernier pion jaune présent sur le plateau.

Dans ce cas, chaque joueur additionne les points des pions qu'il a capturés. Nous ne compterons pas les points ici.

Pour cet examen

Lors de cet examen, vous ne coderez pas entièrement ce jeu. Lisez bien ce qui est demandé dans les questions suivantes.

Dans la suite nous vous conseillons d'utiliser les structures Position et Case ci-dessous :

```
// la position (0,0) correspond au coin supérieur gauche du plateau de jeu.
structure Position
    ligne : entier
    colonne : entier
fin structure
```

```
structure Case
    // la couleur 0 correspond à une case vide
    // la couleur 1 correspond à une case occupée par un pion jaune.
    // la couleur 2 correspond à une case occupée par un pion rouge
    // la couleur 3 correspond à une case occupée par un pion noir
    couleur : entier
    points : entier
fin structure
```

Par extension, une case est dite de la couleur du pion qui l'occupe.

2 Jeu Butin : valider un coup (8 points)

Écrire l'algorithme validerCoup qui reçoit la position du pion jaune sélectionné, la liste des positions des pions à capturer et le plateau de jeu.

```
algorithme validerCoup(positionInitiale↓ : Position, prises↓ : Liste de Positions, plateau↓ : tableau de 8 × 8 de Case)
```

Cet algorithme valide les données encodées par le joueur avant d'effectuer les prises :

- en vérifiant que la position initiale du pion preneur sélectionné est dans le plateau et correspond bien à une case jaune ;
- en vérifiant, successivement pour chaque position que le pion preneur occupera,
  - que cette position est dans le plateau,
  - que la position de la prise suivante est bien voisine du preneur et est bien une case occupée,
  - que la position d'arrivée après avoir fait une prise est valide (libre et dans le plateau) ;

Il est utile d'écrire un algorithme qui, au départ d'une position et de celle d'un voisin trouve la position d'arrivée après un saut au dessus dudit voisin.

N'oubliez pas d'écrire tous les modules nécessaires pour que l'algorithme soit simple à lire et à comprendre.

```
algorithme estValide(positionInitiale↓ : Position,prises↓ : Liste de Positions,plateau↓ : tableau de 8 × 8 Pions) → booléen
    valide : booléen
    indice : entier
    départ,prise,arrivée : Positions
    indice ← 0
    départ ← positionInitiale
    tant que valide ET indice < prises.taille() faire
        prise ← prises.get(indice)
        arrivée ← trouveArrivée(départ,prise)
        valide ← estValide(départ,prise,arrivée,plateau) ET estUnique(prises,prise)
        indice ← indice+1
        départ ← arrivée
    fin tant que
    retourner valide
fin algorithme
```

```
algorithme estValide(départ↓,prise↓,arrivée↓ : Positions,plateau↓ : tableau de 8 × 8 Pions)
→ booléen
    retourner estDans(départ,8) ET estDans(prise,8) ET estDans(arrivée,8) ET est-
Jaune(départ,plateau) ET estVoisin(départ,prise) ET estOccupée(prise,plateau) ET NON
estOccupée(arrivée,plateau)
fin algorithme
```

```
algorithme estDans(position↓ : Position,taille↓ : entier) → booléen
    retourner position.ligne >= 0 ET position.colonne >= 0 ET position.ligne < taille ET
position.colonne < taille
fin algorithme
```

```
algorithme estJaune(position↓ : Position,plateau↓ : tableau de 8 × 8 Pions) → booléen
    retourner plateau[position.ligne,position.colonne].couleur = 1
fin algorithme
```

```
algorithme estVoisin(départ,prise↓ : Positions) → booléen
    deltaLigne,deltaColonne : entier
    deltaLigne ← prise.ligne - départ.ligne
    deltaColonne ← prise.colonne - départ.colonne
    retourner (deltaLigne = 0 ET (deltaColonne = 1 OU deltaColonne = -1)) OU (delta-
Colonne = 0 ET (deltaColonne = 1 OU deltaColonne = -1))
fin algorithme
```

```
algorithme estOccupée(position↓ : Position,plateau↓ : tableau de 8 × 8 Pions) → booléen
    retourner plateau[position.ligne,position.colonne].couleur ≠ 0
fin algorithme
```

```
algorithme trouveArrivée(départ,prise↓ : Positions) → Position
    deltaLigne,deltaColonne : entier
    arrivée : Position
    deltaLigne ← prise.ligne - départ.ligne
    deltaColonne ← prise.colonne - départ.colonne
    si deltaColonne = 0 alors
        arrivée.ligne ← prise.ligne + deltaLigne
        arrivée.colonne ← départ.colonne
    sinon
        arrivée.ligne ← départ.ligne
        arrivée.colonne ← prise.colonne + deltaColonne
    fin si
fin algorithme
```

3 Jeu Butin : fin du jeu (6 points)

Écrire un algorithme qui vérifie si le jeu est terminé, c'est-à-dire qu'il n'y a pas de pion jaune présent sur le plateau.

```
algorithme estTerminé(plateau↓ : tableau de 8 × 8 Pions) → booléen
```

```
algorithme estTerminé(plateau↓ : tableau de 8 × 8 Pions) → booléen
    aUnJaunebooléen
    ligne,colonneentier
    ligne ← 0
    tant que valide ET ligne < 8 faire
        colonne ← 0
        tant que valide ET colonne < 8 faire
            aUnJaune ← estJaune(plateau,ligne,colonne)
            colonne ← colonne +1
        fin tant que
        ligne ← ligne +1
    fin tant que
    retourner NON aUnJaune
fin algorithme
```

4 Réflexion : bataille navale (6 points)

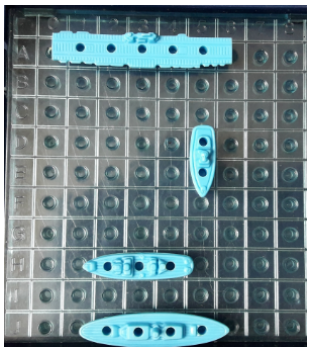
Pour cette question, vous ne devrez pas coder des algorithmes mais les décrire en texte selon 2 propositions alternatives de représentation des données, et présenter les arguments pour ou contre une représentation des données pour chaque algorithme.

La bataille navale, appelée aussi touché-coulé, est un jeu de société dans lequel deux joueurs A et B doivent placer leurs 4 navires sur une grille tenue secrète et tenter de toucher les navires adverses. Nous considérons ici une version simplifiée du jeu. Les règles sont donc différentes de celles que vous pourriez connaître.

Chaque joueur dispose donc de 2 tableaux. Un vide dans lequel il placera à chaque tour les informations reçues de l'adversaire à propos des bateaux de celui-ci (et l'eau qui entoure ceux-ci) et un tableau secret avec ses bateaux.

Chaque joueur à son tour indique à l'adversaire une position. L'adversaire répond en indiquant le bateau touché (ou coulé lorsque toutes les cases d'un bateau ont été touchées) ou «dans l'eau» si aucun bateau n'est touché.

Le gagnant est celui qui parvient à couler tous les navires de l'adversaire avant que tous ses navires ne le soient.



Les 4 navires sont classés ci-dessous par la place qu'ils occupent sur le plateau de jeu :

- 1 porte-avions de 5 cases ;
- 1 croiseur de 4 cases ;
- 1 torpilleur de 3 cases ;
- 1 sous-marin de 2 cases.

Nous considérons ici un morceau du jeu, le cas restreint du tableau secret du joueur A.

Pour programmer ce jeu, il faut disposer des deux algorithmes suivants :

— un coup proposé par B touche-t-il un navire de A ?

```
algorithme proposeCoup(ligne, colonne↓ : entiers) → booléen
```

— le jeu est-il terminé (pour A), c'est-à-dire que tous ses bateaux ont-été coulés ?

```
algorithme estTerminé() → booléen
```

Nous vous proposons deux structures de données différentes pour implémenter le plateau de jeu.

— un tableau de $n \times m$ entiers, la valeur 0 représente une case vide. Les différents bateaux sont représentés par les entiers suivants :

  — 1 porte-avions de 5 cases est représenté par le nombre 5 ;

  — 1 croiseur de 4 cases est représenté par le nombre 4 ;

  — 1 torpilleur de 3 cases est représenté par le nombre 3 ;

  — 1 sous-marin de 2 cases est représenté par le nombre 2.

  Au début du jeu un joueur peut placer ses 4 bateaux comme suit [1]

	0	1	2	3	4	5	6	7
0	0	**5**	**5**	**5**	**5**	**5**	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	**2**	0	0
4	0	0	0	0	0	**2**	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	**3**	**3**	**3**	0	0	0
8	0	0	0	0	0	0	0	0
9	0	0	**4**	**4**	**4**	**4**	0	0

— une liste des positions des cases occupées par les bateaux du joueur, les positions sont définies via la structure ci-dessous :

```
structure CaseBateaux
    // identifiant permet d'identifier le bateau
    // identifiant vaut 5 pour le porte-avions, 4 pour le croiseur,...
    identifiant : entier
    ligne : entier
    colonne : entier
fin structure
```

Dans notre exemple la liste contient les positions des 4 bateaux du joueur. Cette liste contient 14 éléments :

0. $[5, 0, 1]$
1. $[5, 0, 2]$
2. $[5, 0, 3]$
3. $[5, 0, 4]$
4. $[5, 0, 5]$
5. $[2, 3, 5]$
6. $[2, 4, 5]$
7. $[3, 7, 2]$
8. $[3, 7, 3]$
9. $[3, 7, 4]$
10. $[4, 9, 2]$
11. $[4, 9, 3]$
12. $[4, 9, 4]$
13. $[4, 9, 5]$

1. Dans le tableau suivant, la première ligne et la première colonne donnent les indices des colonnes et des lignes respectives.

Pour chacun des deux algorithmes évoqués ci-dessus

1. donnez, en français, une brève description de l'algorithme pour chacune des représentations (donc 4 descriptions : 2 représentations pour les 2 algorithmes) ;
2. pour chaque algorithme, discutez le pour et le contre des deux implémentations.

N'oubliez pas dans votre réflexion de réfléchir à une manière de définir qu'une case d'un bateau a été touché. Par contre il ne faut PAS ici discuter des pour ou contre par rapport à d'autres algorithmes que vous devriez écrire pour composer un programme fonctionnel complet du jeu.

Aide mémoire

Cet aide-mémoire peut vous accompagner lors d'une interrogation ou d'un examen. Il vous est permis d'utiliser ces méthodes sans les développer. Par contre, si vous sentez le besoin d'utiliser une méthode qui n'apparait pas ici, il faudra en écrire explicitement le contenu.

Manipuler les nombres

hasard(n : entier) → entier
: Donne un entier entre 1 et n.

Manipuler les chaines

Remarque : lorsqu'on indique caractère, on signifie une chaine de longueur 1.

chaine[i]
: Désigne le i^{e}caractère de la chaine (en commençant à 1).
Ex : texte[2] ← "a" ou **afficher** texte[1]

chaine1 + chaine2
: Produit une chaine qui est la concaténation des deux chaines.

long(chaine : chaine) → entier
: Donne la longueur de la chaine (nb de caractères).

estLettre(car : caractère) → booléen
: Cette fonction indique si un caractère est une lettre. Par exemple elle retourne vrai pour "a", "e", "G", "K", mais faux pour "4", "$", "@"...

estMinuscule(car : caractère) → booléen
: Permet de savoir si le caractère est une lettre minuscule.

estMajuscule(car : caractère) → booléen
: Permet de savoir si le caractère est une lettre majuscule.

estChiffre(car : caractère) → booléen
: Permet de savoir si un caractère est un chiffre. Elle retourne vrai uniquement pour les dix caractères "0", "1", "2", "3", "4", "5", "6", "7", "8" et "9" et faux dans tous les autres cas.

majuscule(texte : chaine) → chaine

Retourne une chaine où toutes les lettres du texte ont été converties en majuscules.

minuscule(texte : chaine) → chaine

Retourne une chaine où toutes les lettres du texte ont été converties en minuscules.

numLettre(car : caractère) → entier

Retourne toujours un entier entre 1 et 26. Par exemple numLettre("E") donnera 5, ainsi que numLettre("e"). Cette fonction traite donc de la même manière les majuscules et les minuscules. numLettre retournera aussi 5 pour les caractères "é", "è", "ê", "ë"...). Attention, il est interdit d'utiliser cette fonction si le caractère n'est pas une lettre !

lettreMaj(n : entier) → caractère

Retourne la forme majuscule de la n^{e} lettre de l'alphabet (où n sera obligatoirement compris entre 1 et 26). Par exemple, lettreMaj(13) retourne "M".

lettreMin(n : entier) → caractère

Idem pour les minuscules.

chaine(n : réel) → chaine

Transforme un nombre en chaine. Ex : chaine(42) retourne la chaine "42" et chaine (3,14) donnera "3,14".

nombre(ch : chaine) → réel

Transforme une chaine contenant des caractères numériques en nombre. Ainsi, nombre("3,14") retournera 3,14. C'est une erreur de l'utiliser avec une chaine qui ne représente pas un nombre.

sousChaine(ch : chaine, pos : entier, long : entier) → chaine

Permet d'extraire une portion d'une certaine longueur d'une chaine donnée, et ceci à partir d'une position donnée.

position(ch : chaine, sous-chaine : chaine) → entier

Permet de savoir si une sous-chaine donnée est présente dans une chaine donnée. Elle permet d'éviter d'écrire le code correspondant à une recherche. La valeur de l'entier renvoyé est la position où commence la sous-chaine recherchée. Par exemple, position("algorithmique", "mi") retournera 9. Si la sous-chaine ne s'y trouve pas, la fonction retourne 0.

La liste

```
classe Liste de T                                                    // T est un type quelconque
 public:
    constructeur Liste de T()                                        // construit une liste vide
    méthode get(pos : entier) → T                                    // donne un élément en position pos
    méthode set(pos : entier, valeur : T)                            // modifie un élément en position pos
    méthode taille() → entier                                        // donne le nombre actuel d'éléments
    méthode ajouter(valeur : T)                                      // ajoute un élément en fin de liste
    méthode insérer(pos : entier, valeur : T)                        // insère un élément en position pos
    méthode supprimer()                                              // supprime le dernier élément
    méthode supprimerPos(pos : entier)                               // supprime l'élément en position pos
    méthode supprimer(valeur : T) → booléen                          // supprime l'élément de valeur donnée
    méthode vider()                                                  // vide la liste
    méthode estVide() → booléen                                      // la liste est-elle vide ?
    méthode existe(valeur ↓ : T, pos ↑ : entier) → booléen           // recherche un élément
fin classe
```