# ITI1520 Automne 2011 – Devoir 4

Disponible : **Samedi, 5 Novembre**
Date de remise : **Samedi, 19 Novembre, avant 22h00**

**Instructions**

Vous devez faire ce travail en **ÉQUIPE DE DEUX PERSONNES** et suivre toutes les instructions décrites dans le manuel de laboratoire quant à la soumission de devoirs par le Campus Virtuel. Des points seront déduits si vous ne suivez pas ces consignes. Un seul devoir à remettre par équipe de deux personnes. Voici les consignes spécifiques à ce devoir :

* La réponse à la Question 1 doit se trouver dans un fichier Word appelé D4Q1.doc. Incluez également le fichier Java D4Q1.java. Collez le contenu du fichier java dans le fichier Word D4Q1.doc après y avoir inséré la trace pour la Question 1 b).
* La réponse à la Question 2 doit se trouver dans un fichier Java appelé D4Q2Lib.java. Collez le contenu du fichier java dans un fichier Word appelé D4Q2.doc. Nous vous avons donné un squelette de programme dans D4Q2Lib.java. Nous vous avons aussi donné un fichier entier qui s’appelle D4Q2TTT.java ; S’il vous plait, ne soumettez pas ce fichier dans votre répertoire. L’insertion du source code de Java dans un fichier Word va permettre à l’AE d’insérer des commentaires sur le code source dans le fichier Word.
* Compressez tous les fichiers .doc, .java et .class dans une archive intitulée d4\_xxxxxx.zip où xxxxxx doit être remplacé par votre numéro d’étudiant, et soumettez votre archive au Campus Virtuel.

**Barème (sur 100)**

* Consignes et standards: 10 points
* Question 1: 45 points (15+1515)
* Question 2: 45 points
* Question 3: 30 points

## Question 1 (45 points)

**Question 1a.** Écrivez un algorithme **récursif** dans le format Algorithme qui permettra de convertir un entier positif *décimal (*en base 10) **Déc** (entre 0 et 1023) en un entier *binaire* (en base 2) **Bin.** Un entier en base 10 utilise les 10 chiffres 0, 1, …, 9, alors qu’un entier en base 2 utilse seulement les deux chiffres 0 et 1. Par exemple:

Déc: 29
Bin: 11101
(car **1**\*24 + **1**\*23 + **1**\*22 + **0**\*21 + **1**\*20 = 29)

Déc: 999
Bin: 1111100111

**Question 1b.** Tracez votre algorithme pour D = 13 **Question 1c.** Traduisez votre algorithme en une **méthode Java** **récursive** (dans la classe D4Q1). Testez votre méthode en utilisant les test Junit que vous trouverez dans D4Q1test.java. Utilisez le bouton Test, plutôt que Rn en DrJava. Ne soumettez pas les tests. Vous pouvez ajouter plus de test pour vous convaincre que votre algorithme marche comme il faut.

## Question 2 (45 points)

C’est l’heure du jeu! Vous devez compléter un programme pour le jeu de Tic-Tac-Toe .

Une bonne partie de ce program vous est donnée dans deux fichiers: [D4Q2TTT.java](file:///F%3A%5CD4Q2TTT.java) (qui est complet, pas besoin de le modifier) et [D4Q2Lib.java](file:///F%3A%5CD4Q2Lib.java) (que vous devez compléter). L’annexe (voir ci-dessous) contient un exemple des messages affichés pendant le jeu.

1. La méthode *main* contrôle le jeu et contient les étapes suivantes :
	1. Demande à l’usager s’il veut commencer un jeu; si la réponse n’est pas o ou O, le programme se termine. Sinon, main dirige le jeu en utilisant les opérations suivantes:
		1. Efface le tableau (avec la méthode *void effaceTableau(char [][] tab)*).
		2. Affiche le tableau (avec la méthode *void afficheTableau(char [][] tab)*).
		3. Fait une partie (avec la méthode *void joue(char [][] tab, char joueur)* – incluant les demandes pour de nouvelles positions de pion sur le tableau de jeu).
		4. Vérifie si le joueur a gagné ou s’il y a match nul (avec la méthode *boolean verifieQuiAGagne(char [][] tab)*).
		5. Si le jeu n’est pas fini, continue (c'est-à-dire, retourne à iii).

Note: la matrice du tableau de jeu est créée dans la méthode main et passée comme référence aux autres méthodes.

* 1. Après chaque jeu, demande à l’usager s’il veut recommencer une nouvelle partie (c'est-à-dire, retourne à a).
1. La méthode *verifieQuiAGagne* appelle les méthodes suivantes :
	1. *char testLignes(char [][] tab)* – pour voir si on trouve une ligne complète et homogène.
	2. *char testCols(char [][] tab)* – pour voir si on trouve une colonne complète et homogène.
	3. *char testDiags(char [][] tab)* – pour voir si on trouve une diagonale complète et homogène.
	4. *boolean testMatchNul(char [][] tab)* – pour voir s’il y a match nul.

Note : une ligne,colonne ou diagonale complète et homogène correspond à une victoire (du joueur ou de l’ordinateur).

Votre travail consiste à compléter les méthodes suivantes qui ont été rassemblées dans une librairie, la classe A4Q2Lib. Ces méthodes incomplètes se trouvent donc dans le fichier A4Q2Lib.java et s’appellent:

* *void effaceTableau(char [][] tab)*
* *boolean verifieGagner(char [][] tab)*
* *char testLignes(char [][] tab)*
* *char testCols(char [][] tab)*
* *char testDiags(char [][] tab)*
* *boolean testMatchNul(char [][] tab)*

Notes:

1. La méthode verifieQuiaGagne affiche le message “Match nul” plutôt que “Joueur X a gagné” quand il y a match nul.
2. Les coordonnées d’une nouvelle position sont données par l’utilisateur sur la même ligne. Elles seront lues en utilisant ITI1520.readIntLine().
3. Vous utiliserez de nombreuses commandes « if » dans votre programme. La plupart de ces commandes auront une section « else » vide. Afin d’écrire du code plus clair, il est désirable d’éliminer la section « else » de la commande, mais d’indiquer cette omission avec un commentaire de la manière suivante :

If (condition)

{

 // <Commandes à exécuter lorsque la condition est vraie>

} // pas de else – rien de plus à faire

1. Ne soyez pas intimidés par le nombre de méthodes à écrire. La plupart de ces méthodes sont très courtes et se ressemblent. Par exemple, il n’y a pas grande différence entre la vérification du statut d’une ligne et d’une colonne.
2. Les méthodes testLignes, testCols, and testDiags, retournent l’un des caractères suivants : ‘-‘, ‘X’ or ‘O’. Si ‘-‘ est retourné, alors personne n’a gagné, sinon le caractère du joueur qui a gagné est retourné.
3. Nous ne vous demandons pas de soumettre les modèles de vos algorithmes. C’est toujours une bonne idée de concevoir vos algorithmes sur papier avant de les coder en Java, mais vous pouvez travailler comme il vous sait gré.
4. De la documentation supplémentaire sur chaque méthode est disponible dans le fichier D4Q2Lib.java.
5. SVP, ne soumettez que le fichier D4Q2Lib.java; Ne faites aucun changement au fichier D4Q2TTT.java et ne le soumettez pas (il est complet).

**Note : Vous recevrez 5 points en Bonus si vous implémentez** testLignes **ou** testCols **en utilisant une méthode récursive, sans boucle!!!!!**

**Annexe – exemple de jeu**

Voulez-vous commencer une nouvelle partie (O ou N): O

 0 1 2

0 - - -

1 - - -

2 - - -

Joueur X, SVP indiquez une ligne et une colonne de 0 a 2:

1 1

 0 1 2

0 - - -

1 - X -

2 - - -

Joueur O, SVP indiquez une ligne et une colonne de 0 a 2:

0 0

 0 1 2

0 O - -

1 - X -

2 - - -

Joueur X, SVP indiquez une ligne et une colonne de 0 a 2:

1

Joueur X, SVP indiquez une ligne et une colonne de 0 a 2:

1 3

Joueur X, SVP indiquez une ligne et une colonne de 0 a 2:

2 2

 0 1 2

0 O - -

1 - X -

2 - - X

Joueur O, SVP indiquez une ligne et une colonne de 0 a 2:

0 0

La position 0 0 est occupée

Joueur O, SVP indiquez une ligne et une colonne de 0 a 2:

0 2

 0 1 2

0 O - O

1 - X -

2 - - X

Joueur X, SVP indiquez une ligne et une colonne de 0 a 2:

1 0

 0 1 2

0 O - O

1 X X -

2 - - X

Joueur O, SVP indiquez une ligne et une colonne de 0 a 2:

0 1

Joueur O a gagné!

 0 1 2

0 O O O

1 X X -

2 - - X

Voulez-vous recommencer une partie (O ou N): N

**Amusez-vous bien!!!**