Méthodes de test (Partie contre l'Ordinateur)

(Comme la partie contre l'ordinateur comprend des tours où l'humain joue, toutes les méthodes de la partie 1 (Joueur contre Joueur) sont réutilisées. Les six premières méthodes de tests sont donc identiques à la partie 1)

Méthode n°1

Méthode : createGameboard

```
    /**
     * @param stickQuantity
     * @return le tableau de la taille adéquate avec @stickQuantity en premier element
     *
     */
    int[] createGameboard(int stickQuantity){
        int[] gameboard = new int[ stickQuantity ];
        gameboard[0] = stickQuantity;
        return gameboard;
    }
```

Code méthode de test

```
/**
    * Teste un cas de la méthode createGameboard()
    * On ne pourra pas créer un tableau avec moins de 2 allumettes,
    * le tableau doit être jouable au moins une fois
    * La vérification se fait avant l'appel de la méthode
    * @param stickQuantity
    * @param expected
    */
   void testCaseCreateGameboard(int stickQuantity, int[] expected){
      System.out.println(" *******  Test");
      System.out.print("Nombre d'allumette = " + stickQuantity + " ");
      int[] gameboard = createGameboard(stickQuantity);
      System.out.print(Arrays.toString(gameboard) + " ");
      if (gameboard.length == stickQuantity && gameboard[0] == stickQuantity){
         System.out.println("OK");
      } else {
         System.err.println(" ERROR ");
      }
   }

   /**
    * Teste en batterie la méthode createGameboard
    */
   void testCreateGameboard(){
      System.out.println(" *******  Test de la méthode createGameboard *********");
      int[] expected1 = {4,0,0,0};
      testCaseCreateGameboard(4, expected1);
      int[] expected2 = {7,0,0,0,0,0,0};
      testCaseCreateGameboard(7, expected2);
   }
```

Execution :

```
*******  Test de la méthode createGameboard *********
```

```
******* Test
Nombre d'allumette = 4 [4, 0, 0, 0] OK
******* Test
Nombre d'allumette = 7 [7, 0, 0, 0, 0, 0, 0] OK
```

Méthode n°2

Méthode : isPlayable

```
/**
     * @param gameboard
     * @return true s'il est possible de jouer encore au moins un coup
     * @return false sinon
     */
    boolean isPlayable(int[] gameboard){
        boolean playable = false;
        for(int i = 0 ; i < gameboard.length ; i++){
            if (gameboard[i] > 2){
                playable = true;
            }
        }
        return playable;
    }
```

Code méthode de test

```
/**
    * Test un cas de la méthode isPlayable()
    * @param gameboard
    * @param expected
    */
   void testCaseIsPlayable(int[] gameboard, boolean expected){
      System.out.println(" *******  Test ");
      System.out.print(Arrays.toString(gameboard));
      System.out.print(" Attentes : " + expected + " ");
      if (isPlayable(gameboard) == expected){
         System.out.println(" OK ");
      } else {
         System.out.println(" ERROR ");
      }
   }

   /**
    * Test en batterie la méthode isPlayable()
    */
   void testIsPlayable(){
      System.out.println(" *******  Test de la méthode isPlayable() *********");
      int[] gameboard1 = {7,3,0,0};
      testCaseIsPlayable(gameboard1, true);
      int[] gameboard2 = {1,2,7,0};
      testCaseIsPlayable(gameboard2, true);
      int[] gameboard3 = {2,2,3,7};
      testCaseIsPlayable(gameboard3, true);
      int[] gameboard4 = {2,1,2,0};
      testCaseIsPlayable(gameboard4, false);
      }
```

Execution :

```
******* Test de la méthode isPlayable() *********
******* Test
```

```
[7, 3, 0, 0] Attentes : true  OK
 *******  Test
[1, 2, 7, 0] Attentes : true  OK
 *******  Test
[2, 2, 3, 7] Attentes : true  OK
 *******  Test
[2, 1, 2, 0] Attentes : false  OK
```

Méthode n°3

Méthode : display

```
    /**
     * Affiche avec des bâtons l'état du jeu
     * @param gameboard le tableau d'entier du jeu
     */
    void display(int[] gameboard){
        int i = 0;
        while( i < gameboard.length && gameboard[i] != 0 ){
            System.out.print(i + "\t : ");
            for (int j = 0 ; j < gameboard[i] ; j++){
                System.out.print("| ");
            }
            System.out.println();
            i++;
        }
    }
```

Code méthode de test

```
/**
    * Test un cas de la méthode display
    * La vérification doit se faire à l'oeil
    * @param gameboard
    */
   void testCaseDisplay(int[] gameboard){
      System.out.println(" *******  Test ");
      System.out.println(Arrays.toString(gameboard));
      display(gameboard);
   }

   /**
    * Teste en batterie la méthode display()
    * La vérification doit se faire à la main
    */
   void testDisplay(){
      System.out.println(" *******  Test de la méthode display *********");
      int[] gameboard1 = {7,3,0,0};
      testCaseDisplay(gameboard1);
      int[] gameboard2 = {3,3,3};
      testCaseDisplay(gameboard2);
   }
```

Execution :

```
 *******  Test de la méthode display *********
 *******  Test
[7, 3, 0, 0]
0       : | | | | | | |
1       : | | |
```

```
******* Test
[3, 3, 3]
0     :|||
1     :|||
2     :|||
```

Méthode n°4

Méthode : playableLine

```
    /**
     * @param gameboard
     * @param divideQuantity
     * @return l'index de la seule ligne jouable s'il y en a une
     * S'il en a plusieurs ou aucune, renvoie -1
     */
    int playableLine(int[] gameboard, int divideQuantity){
        int index = -1;
        if (divideQuantity == 0) {
            index = 0;
        }
        else {
            int playableLigneQuantity = 0;
            int k = 0;
            while (playableLigneQuantity < 2 && k < (divideQuantity+1)
              && k < gameboard.length){
                if (gameboard[k] > 2 ){
                    index = k;
                    playableLigneQuantity++;
                }
                k++;
            }
            if (playableLigneQuantity != 1){
                index = -1;
            }
        }
        return index;
    }
```

Code méthode de test

```
    /**
     * Teste un cas unique de la méthode playableLine()
     * @param gameboard
     * @param divideQuantity
     * @param expected
     */
    void testCasPlayableLineQuantity(int[] gameboard, int divideQuantity, int expected){
        System.out.println("*****  Test ");
        display(gameboard);
        if (playableLine(gameboard, divideQuantity) == expected){
            System.out.println("Nombre de divisions effectuées : " + divideQuantity +" | Attentes : "
            + expected + " | réponse : " + playableLine(gameboard, divideQuantity) + " : OK ");
        } else {
            System.err.println("Nombre de divisions effectuées : " + divideQuantity +" | Attentes : "
            + expected + " | réponse : " + playableLine(gameboard, divideQuantity) + " : ERROR ");
        }
    }

    /**
     * Teste en batterie la méthode playableLine()
     */
    void testPlayableLine(){
        System.out.println(" *******  Test de la méthode playableLine *********");
```

```
    int[] gameboard = {7,0,0,0};
    testCasPlayableLineQuantity(gameboard, 0,0);
    int[] gameboard2 = {3,4,0,0};
    testCasPlayableLineQuantity(gameboard2, 1,-1);
    int[] gameboard3 = {3,2,2,0};
    testCasPlayableLineQuantity(gameboard3, 2,0);
    int[] gameboard4 = {2,2,2,1};
    testCasPlayableLineQuantity(gameboard4, 3,-1);
  }
```

Execution :

```
 *******  Test de la méthode playableLine **********
*****  Test
0      : | | | | | | | |
Nombre de divisions effectuées : 0 | Attentes : 0 | réponse : 0 : OK
*****  Test
0      : | | |
1      : | | | |
Nombre de divisions effectuées : 1 | Attentes : -1 | réponse : -1 : OK
*****  Test
0      : | | |
1      : | |
2      : | |
Nombre de divisions effectuées : 2 | Attentes : 0 | réponse : 0 : OK
*****  Test
0      : | |
1      : | |
2      : | |
3      : |
Nombre de divisions effectuées : 3 | Attentes : -1 | réponse : -1 : OK
```

Méthode n°5

Méthode : possible

```
    /**
     * @param gameboard le tableau d'entier du jeu
     * @param lineNB le numero de la ligne du tableau souhaitée
     * @param stickQuantity la quantité de batons à séparer
     * @return true s'il est possible de separer StickQuantity bâtons de la ligne LineNB
     * du tableau de jeu
     * false sinon
     */
    boolean possible(int[] gameboard, int lineNB, int stickNB){
        boolean possible = false;
        if (gameboard[lineNB] > 2){
            if (gameboard[lineNB] == 3){
                if (stickNB == 1 || stickNB == 2){
                    possible = true;
                }
            }
            else if (stickNB >= 1  && stickNB < gameboard[lineNB]
                && gameboard[lineNB]- stickNB != stickNB) {
                possible = true;
            }
        }
         return possible;
     }
```

Code méthode de test

```
  void testCasePossible(int[] gameboard, int lineNB, int stickNB, boolean expected){
```

```
    System.out.println(" *******  Test");
    display(gameboard);
    System.out.print("Ligne choisie : " + lineNB + " | nombre d'allumette à séparer : " + stickNB + " : ");
    if (possible(gameboard, lineNB, stickNB) == expected){
       System.out.println(" OK ");
    } else {
       System.err.println(" ERROR ");
    }
 }

 /**
  * test en batterie la m"thode possible()
  * On ne prend ici que des index existants car la vérification aura déjà eu lieu
  */
 void testPossible(){
    System.out.println(" *******  Test de la méthode possible() **********");
    int[] gameboard = {7,0,0,0};
    testCasePossible(gameboard, 0,3, true);
    int[] gameboard2 = {3,4,0,0};
    testCasePossible(gameboard2, 1,2,false);
    int[] gameboard3 = {3,2,2,0};
    testCasePossible(gameboard3, 2,2,false);
    int[] gameboard4 = {2,2,2,1};
    testCasePossible(gameboard4, 3,1, false);
 }
```

Execution :

```
 *******  Test de la méthode possible() **********
 *******  Test
0      : | | | | | | |
Ligne choisie : 0 | nombre d'allumette à séparer : 3 :  OK
 *******  Test
0      : | | |
1      : | | | |
Ligne choisie : 1 | nombre d'allumette à séparer : 2 :  OK
 *******  Test
0      : | | |
1      : | |
2      : | |
Ligne choisie : 2 | nombre d'allumette à séparer : 2 :  OK
 *******  Test
0      : | |
1      : | |
2      : | |
3      : |
Ligne choisie : 3 | nombre d'allumette à séparer : 1 :  OK
```

Méthode n°6

Méthode : split

```
    /**
     * sépare strickQuantity bâtons de la ligne LineNB du jeu directement
     * dans le gameboard
     * @param gameboard le tableau d'entier du jeu
     * @param lineNB le numero de la ligne du tableau souhaitée
     * @param stickQuantity la quantité de batons à séparer
     */
    void split(int[] gameboard, int lineNB, int stickQuantity){
```

```
        int i =0;
        while(gameboard[i] != 0){
            i++;
        }
        gameboard[i] = stickQuantity;
        gameboard[lineNB] = gameboard[lineNB] - stickQuantity;
        System.out.println();
    }
```

Code méthode de test

```
void testCaseSplit(int[] gameboard, int lineNB, int stickQuantity, int[] expected){
      System.out.println("********* Test");
      display(gameboard);
      split(gameboard, lineNB, stickQuantity);
      System.out.print("Numéro de ligne : " + lineNB + "| Quantité d'allumettes : " + stickQuantity + " | ");
      System.out.println("Attente : ");
      display(expected);
      if (Arrays.equals(gameboard, expected)){
         System.out.println(" OK ");
      } else {
         System.out.println(" ERROR ");
      }
   }


   /**
    * Teste en batterie la méthode split() en batterie
    * On donnera des arguments valide car la vérification se fera avant
    */
   void testSplit(){
      System.out.println(" *******  Test de la méthode split *********");
      int[] gameboard = {7,0,0,0};
      int[] expected1 = {4,3,0,0};
      testCaseSplit(gameboard, 0,3, expected1);
      int[] gameboard2 = {3,4,0,0};
      int[] expected2 = {3,1,3,0};
      testCaseSplit(gameboard2, 1,3,expected2);
      int[] gameboard3 = {3,2,2,0};
      int[] expected3 = {2,2,2,1};
      testCaseSplit(gameboard3, 0,1,expected3);
      int[] gameboard4 = {2,2,3,0};
      int[] expected4 = {2,2,2,1};
      testCaseSplit(gameboard4, 2,1, expected4);
   }
```

Execution :

```
 *******  Test de la méthode split *********
********* Test
0       : | | | | | | |

Numéro de ligne : 0| Quantité d'allumettes : 3 | Attente :
0       : | | | |
1       : | | |
 OK
********* Test
0       : | | |
1       : | | | |

Numéro de ligne : 1| Quantité d'allumettes : 3 | Attente :
0       : | | |
```

```
1     :|
2     :|||
 OK
********* Test
0     :|||
1     :||
2     :||

Numéro de ligne : 0| Quantité d'allumettes : 1 | Attente :
0     :||
1     :||
2     :||
3     :|
 OK
********* Test
0     :||
1     :||
2     :|||

Numéro de ligne : 2| Quantité d'allumettes : 1 | Attente :
0     :||
1     :||
2     :||
3     :|
 OK
```

Méthode n°7

Méthode : moveDefiner

```
/**
     * Cette méthode est le "cerveau" de l'ordinateur, c'est
     * elle qui renvoie le meilleur coup à jouer par l'ordinateur
     * @param gameboard
     * @return Les coordonnées {numéro de ligne, quantité d'allumette} du coup à jouer
     */
    int[] moveDefiner(int[] gameboard){
        int[] move = {0,0};
        int i = 0;
        boolean found = false;
        int j = 1;
        while (i < gameboard.length && gameboard[i] != 0 && !found){
            if(gameboard[i] > 2){
                while(j < gameboard[i] && !found){
                    if (gameboard[i] - j != j &&
                      isALoosingGameboard(deepSplit(gameboard, i, j))){
                        found = true;
                        move[0] = i;
                        move[1] = j;
                    }
                    j++;
                }
            }
            i++;
        }
        // si la situation est perdante alors on joue aléatoirement dans la
        // premiere ligne disponible en espérant que le joueur fasse une erreur
        int[] nullArray = {0,0};
        if (Arrays.equals(move, nullArray)){

            while (gameboard[move[0]] <= 2){
                move[0]++;
            }
            while(move[1] < 1 || move[1] > gameboard[move[0]]-1 || gameboard[move[0]]-
```

```
            move[1] == move[1]){
               move[1] = (int) (Math.random()*gameboard[move[0]]-1);
            }

        }
        return move;
    }
```

Code méthode de test

```
/**
    * Le mouvement est correct si la plateau apres le mouvement est perdant
    * @param gameboard
    * @param expected boolean attendu de isALoosingGameboard
    */
   void testCaseMoveDefiner(int[] gameboard, boolean expected){
      System.out.println("****** test");
      display(gameboard);
      int[] move = moveDefiner(gameboard);
      System.out.print("Ligne : " + move[0] + " | nombre d'allumette : " + move[1]
         + " | Attente plateau perdant  : " + expected );
      if ( isALoosingGameboard(deepSplit(gameboard, move[0], move[1])) == expected){
         System.out.println(" OK ");
      } else {
         System.err.println(" ERROR ");
      }
   }

   /**
    * Teste en batterie la méythode MoveDefiner
    * Le tableau doit être jouable, la vérification se fait avant l'appel de la méthode
    */
   void testMoveDefiner(){
      System.out.println("**** Tests MoveDefiner ******") ;
      int[] gameboard1 = {3,0,0};
      testCaseMoveDefiner(gameboard1, true);
      int[] gameboard2 = {4,0,0,0};
      testCaseMoveDefiner(gameboard2, false);
      int[] gameboard3 = {5,0,0,0,0};
      testCaseMoveDefiner(gameboard3, true);
      int[] gameboard4 = {2,2,2,3,0};
      testCaseMoveDefiner(gameboard4, true);
   }
```

Execution :

```
**** Tests MoveDefiner ******
****** test
0       : | | |
Ligne : 0 | nombre d'allumette : 1 | Attente plateau perdant  : true OK
****** test
0       : | | | |
Ligne : 0 | nombre d'allumette : 1 | Attente plateau perdant  : false OK
****** test
0       : | | | | |
Ligne : 0 | nombre d'allumette : 1 | Attente plateau perdant  : true OK
****** test
0       : | |
1       : | |
2       : | |
3       : | | |
Ligne : 3 | nombre d'allumette : 1 | Attente plateau perdant  : true OK
```

Méthode n°8

Méthode : deepSplit

```
    /**
     * @param gameboard
     * @param ligne l'index de la ligne à diviser
     * @param quantity la quantité d'allumette à séparer
     * @return une copie du gameboard avec la bonne séparation
     * En faisant une copie dans une autre adresse, on peut stocker les
     * différentes positions sans créer de problèmes
     */
   int[] deepSplit(int[] gameboard, int ligne, int quantity){
       int[] newGameboard = new int[gameboard.length];
       int i = 0;
       while (i < newGameboard.length && gameboard[i] != 0){
           if (i == ligne){
               newGameboard[i] = gameboard[i]-quantity;
           } else {
               newGameboard[i] = gameboard[i];
           }
           i = i +1;
           }
       newGameboard[i] = quantity;
       return newGameboard;
       }
```

Code méthode de test

```
  void testCaseDeepSplit(int[] gameboard, int ligne, int quantity, int[] expected){
     System.out.println("****** Test");
     System.out.println(Arrays.toString(gameboard));
     System.out.println("Numéro de ligne : "+ ligne);
     System.out.println("Quantité d'allumettes : " + quantity);
     System.out.println("Attente : " + Arrays.toString(expected));
     System.out.println("Resultat : " + Arrays.toString(deepSplit(gameboard, ligne, quantity)));
     if (Arrays.equals(deepSplit(gameboard, ligne, quantity), expected)){
        System.out.println("OK ");
     } else {
        System.err.println(" ERROR ");
     }
  }

  void testDeepSplit(){
     System.out.println(" *******  Test de la méthode deepSplit *********");
     int[] gameboard1 = {4,0,0,0};
     int[] expected1 = {1,3,0,0};
     testCaseDeepSplit(gameboard1, 0, 3, expected1);
     int[] gameboard2 = {4,3,0,0,0,0,0};
     int[] expected2 = {1,3,3,0,0,0,0};
     testCaseDeepSplit(gameboard2, 0, 3, expected2);
     int[] gameboard3 = {4,5,0,0,0,0,0,0,0};
     int[] expected3 = {4,2,3,0,0,0,0,0,0};
     testCaseDeepSplit(gameboard3, 1, 3, expected3);

  }
```

Execution :

```
*******  Test de la méthode deepSplit *********
****** Test
```

```
[4, 0, 0, 0]
Numéro de ligne : 0
Quantité d'allumettes : 3
Attente : [1, 3, 0, 0]
Resultat : [1, 3, 0, 0]
OK
****** Test
[4, 3, 0, 0, 0, 0, 0]
Numéro de ligne : 0
Quantité d'allumettes : 3
Attente : [1, 3, 3, 0, 0, 0, 0]
Resultat : [1, 3, 3, 0, 0, 0, 0]
OK
****** Test
[4, 5, 0, 0, 0, 0, 0, 0, 0]
Numéro de ligne : 1
Quantité d'allumettes : 3
Attente : [4, 2, 3, 0, 0, 0, 0, 0, 0]
Resultat : [4, 2, 3, 0, 0, 0, 0, 0, 0]
OK
```

Méthode n°9

Méthode : isALoosingGameboard

```
/**
     * @param gameboard le tableau de jeu
     * @return true si la disposition est perdante.
     * i.e. toutes les dispositions filles sont gagnantes
     * Une situation perdante est une situation où le joueur
     * qui est en train de jouer ne peut que perdre la partie
     * Cette méthode parcours toutes les dispositions filles
     * possibles en partant d'une disposition mère
     * Si une disposition fille est perdante alors la disposition
     * mère est gagnante donc la méthode renvoie false
     */
    boolean isALoosingGameboard(int[] gameboard){
        if (!isPlayable(gameboard)){
            return true;
        }
        boolean tousLesFilsSontgagnants = true;
        int i = 0;
        int j = 1;
        //Parcours des lignes du gameboard
        while(i < gameboard.length && tousLesFilsSontgagnants){
            //On verifie que la quantité d'allumette est supérieur à 2
            // pour limiter les tests car on ne peut pas séparer sinon
            if(gameboard[i] > 2){
               // On parcours toutes les séparations possibles
               while( j < (gameboard[i]/2 + gameboard[i]%2) && tousLesFilsSontgagnants){
                    // Si la séparation est valide i.e. les 2 tas sont inégaux on sépare
                    // et on demande s'il est perdant
                    if(gameboard[i]-j != j){
                        int[] splitedGameboard = deepSplit(Arrays.copyOf(gameboard,
                            gameboard.length), i, j);
                        //Si une des dispositions filles est perdante alors on arrête
                        // et on renvoie false : la situation est gagnante
                        if (isALoosingGameboard(splitedGameboard)){
                            tousLesFilsSontgagnants = false;
                        }
                    }
                    j++;
                }
            }
            i++;
```

```
        }
        return tousLesFilsSontgagnants;
    }
```

Code méthode de test

```
  /**
   * @param gameboard
   * @param expected boolean reponse attendue
   */
  void testCasisALoosingGameboard(int[] gameboard, boolean expected){
      System.out.println(" *******  Test ");
      System.out.print("Gameboard : "+ Arrays.toString(gameboard) + " expected = " + expected);
      if (isALoosingGameboard(gameboard)== expected){
          System.out.println(" OK ");
      }
      else {
          System.out.println(" ERROR ");
      }
  }

  void testisALoosingGameboard(){
      System.out.println(" *******  Test de la méthode isALoosingGameboard *********");
      int[] gameboard1 = {2,0};
      testCasisALoosingGameboard(gameboard1, true);
      int[] gameboard2 = {3,0,0,0};
      testCasisALoosingGameboard(gameboard2, false);
      int[] gameboard3 = {4,0,0,0};
      testCasisALoosingGameboard(gameboard3, true);
      int[] gameboard4 = {5,0,0,0,0};
      testCasisALoosingGameboard(gameboard4, false);
      int[] gameboard5 = {7,0,0,0,0,0,0};
      testCasisALoosingGameboard(gameboard5, true);
      int[] gameboard6 = {10,0,0,0,0,0,0,0,0,0};
      testCasisALoosingGameboard(gameboard6, true);
      int[] gameboard7 = {8,0,0,0,0,0,0,0,0};
      testCasisALoosingGameboard(gameboard7, false);
      int[] gameboard9 = {14,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
      testCasisALoosingGameboard(gameboard9, false);
      int[] gameboard10 = {12,3,0,0,0,0,0,0,0,0,0,0,0,0,0};
      testCasisALoosingGameboard(gameboard10, true);
  }
```

Execution :

```
******* Test de la méthode isALoosingGameboard *********
 *******  Test
Gameboard : [2, 0] expected = true OK
 *******  Test
Gameboard : [3, 0, 0, 0] expected = false OK
 *******  Test
Gameboard : [4, 0, 0, 0] expected = true OK
 *******  Test
Gameboard : [5, 0, 0, 0, 0] expected = false OK
 *******  Test
Gameboard : [7, 0, 0, 0, 0, 0, 0] expected = true OK
 *******  Test
Gameboard : [10, 0, 0, 0, 0, 0, 0, 0, 0, 0] expected = true OK
 *******  Test
Gameboard : [8, 0, 0, 0, 0, 0, 0, 0, 0] expected = false OK
 *******  Test
Gameboard : [14, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] expected = false OK
```

```
******* Test
Gameboard : [12, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] expected = true OK
```