Code :

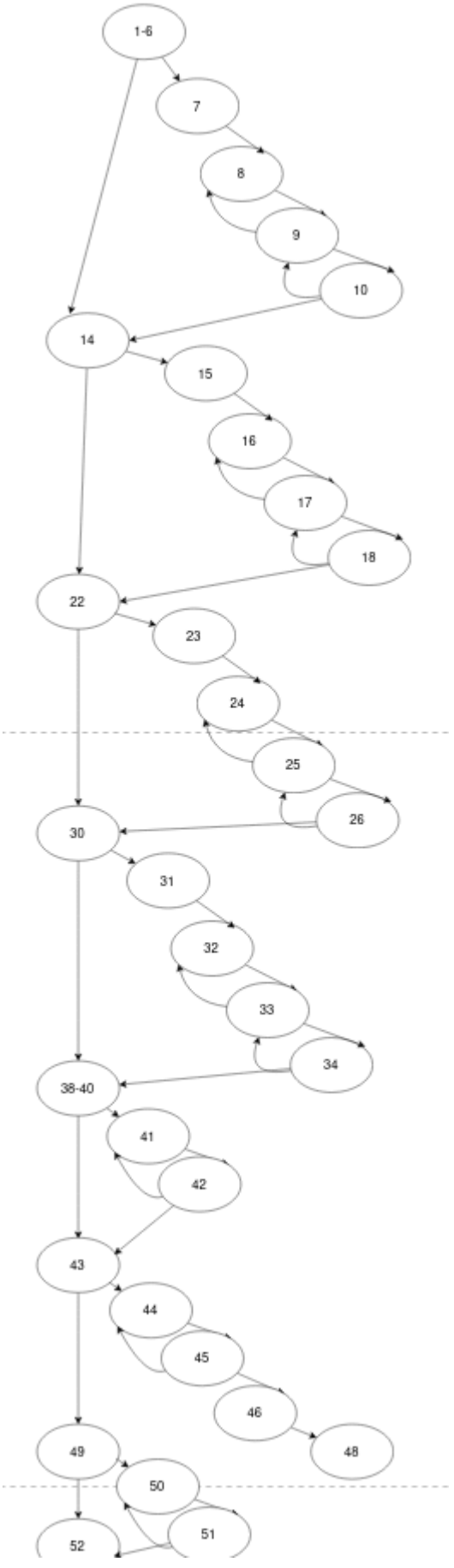
```
public int[][] lesDeplacements(Animal e) {
int v= e.getVitesse();
int x= e.getX();
int y= e.getY();
int[][] res =new int[v*(2+v*2)][2];
if (v>=1){
int[][] res1 = lesVoisins(x,y);
for (int i=0; i<4; i++){
        for (int j=0; j<2; j++){
        res[i][j] = res1[i][j];
        }
}
}
if (v>=2){
int[][] res2 = lesVoisin2(x,y);
for (int i= 0; i <8; i++) {
        for (int j= 0; j <2; j++) {
        res[i+4][j] = res2[i][j];
        }
}
}
if (v>=3){
int[][] res3 = lesVoisin3(x,y);
for (int i= 0; i <12; i++) {
        for (int j= 0; j <2; j++) {
        res[i+12][j] = res3[i][j];
        }
}
}
if (v>=4){
int[][] res4 = lesVoisin4(x,y);
for (int i= 0; i <16; i++) {
        for (int j= 0; j <2; j++) {
        res[i+24][j] = res4[i][j];
        }
}

}

ArrayList<int[]> a= new ArrayList<>();
ArrayList<int[]> b= new ArrayList<>();
for (int i= 0; i <v*(2+v*2); i++) {
a.add(res[i]);
}
for (int i= 0; i <v*(2+v*2); i++) {
```

```
        if ((!(a.get(i)[0]>=nbColonnes || a.get(i)[1] >= nbLignes || a.get(i)[0] < 0 || a.get(i)[1] <
0)) && (elements[a.get(i)[0]][a.get(i)[1]].isAccessible() )){
                b.add(a.get(i));;
        }
        }
        int[][] res2= new int[b.size()][2];
        for (int i= 0; i <b.size(); i++) {
        res2[i]=b.get(i);
        }
        return res2;
```

Graph de flot de contrôle correspondant :



Définition du jeu de test :
Couverture de branche car présence de beaucoup de if sans else.
Étant donné que l'intégralité des données utilisées pour les conditions et les calcules sont toutes issues de animal, afin de simplifier nos tests, nous allons les considérer comme paramètres.

Valeur X	Valeur Y	Valeur V	Résultat attendu	Résultat obtenu
1	1	0	[]	
1	1	4	[[1, 2], [2, 1], [1, 3], [3, 1],[2, 2], [1, 4], [2, 3], [3, 2], [4, 1], [1, 5], [2, 4], [3, 3], [4, 2], [5, 1]]	

Code de test :

```
package com.example.jeuduloup2;

import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

class GrilleTest {
   private Grille grille;

   @BeforeEach
   void setUp() {
       grille=new Grille(10,10);
       grille.miseEnPlace();

   }
   @AfterEach
   void tearDown() {
       grille=null;
   }

   @Test
   void lesDeplacements() {
       Mouton mouton=new Mouton(1,1);
       mouton.setVitesse(4);
       int[][] deplacements= grille.lesDeplacements(mouton);
```

```
        int[][] attendu={{1, 2}, {2, 1}, {1, 3}, {3, 1}, {2, 2},
{1, 4}, {2, 3},  {3, 2},  {4, 1}, {1, 5}, {2, 4},{3, 3},{4, 2},
{5, 1}};
        assertEquals(attendu.length,deplacements.length);

    }
    @Test
    void lesDeplacements2() {
        Mouton mouton=new Mouton(1,1);
        mouton.setVitesse(0);
        int[][] deplacements=grille.lesDeplacements(mouton);
        int[][] attendu={};
        assertEquals(attendu.length,deplacements.length);
    }
}
```

Résultat des tests :