INFORME PRÀCTICA 3: TEST I QUALITAT DE SOFTWARE

Marc Martí Asensio: 1424859

Alejandro Garcia Carballo: 1423957

Índex:

TDD i proves de caixa negra

Durant el desenvolupament del joc, hem estat seguint la manera de procedir que marca el Test Driven Development. Per tant, la majoria de proves de caixa negra s'han realitzat abans de conèixer com seria amb exactitud el codi de la funció en qüestió.

En aquestes proves, el que busquem és comprovar que la funció desenvolupada funciona correctament, buscant aquells casos on pot no comportar-se de la manera esperada o fins i tot donar errors.

1. Classe Tauler

Funció solapa:

La funció solapa s'encarrega de comprovar si la figura que s'ha de col·locar solapa amb els límits (murs) del tauler o alguna figura de les que ja estan col·locades.

La funció solapa rep tres paràmetres: la figura que s'està movent, el moviment horitzontal i el moviment vertical que se li vol aplicar. Els moviments es poden no aplicar, prenent el valor 0 en ambdós casos.

Per provar aquesta funció, utilitzarem els següents valors:

- Valors frontera: aquells valors vàlids que es troben molt propers als valors límits que donarien errors. Provarem aquelles posicions que limiten amb els murs del tauler (inferior, esquerra i dret) i amb les figures que ja es troben col·locades.
- Valors límit: com a valors límits no vàlids, cal provar aquells casos on les figures solapen amb els murs del tauler i en aquells casos on no (però no són valors frontera).

Funció comprovarFilesCompletades():

La funció comprovarFilesCompletades s'encarrega de mirar si en el tauler hi ha alguna fila completada. En cas afirmatiu, les borrarà i desplaçarà les altres files cap a posicions inferiors.

Els casos generals que provarem que fan que el programa es comporti de manera diferent són:

- La fila inferior està completada.
- Hi ha una fila completada dins del tauler (per exemple, la fila del mig).
- La fila superior està completada.

Per aquests casos, podem afegir algunes variacions que ampliaran més els casos de prova:

- Només hi ha una fila completada.
- Hi ha unes quantes files seguides completades.
- Hi ha dues files completades, però no estan seguides (separades per una fila no completada).
- Quan s'han comprovat i eliminat les files completades, encara queden figures encaixades en el tauler (de files que no estan completes).

Funció eliminarFilaCompletada():

La funció eliminarFilaCompletada s'encarrega d'eliminar una fila que ja ha estat completada. Quan la elimina, desplaça les files que es trobaven a sobre d'ella una posició cap a baix. En el cas de que es completi la fila superior, aquesta quedarà buida.

El mètode rep per paràmetre el nombre de la fila que ha estat completada, indicada per la funció comprovarFilesCompletades.

Els casos de prova que hem utilitzat són molt semblants als de comprovarFilesCompletades, ja que les dues funcions estan relacionades de manera directa.

Funció posicioValida():

La funció posicioValida s'encarrega de comprovar si la posició donada per paràmetre existeix dins del Tauler. La seva utilitat dins del codi del joc és evitar que s'intenti accedir a posicions del Tauler que estan fora del rang de la matriu i que sorgeixin errors.

Per aquest mètode, ens interessa provar aquells valors que poden donar problemes a l'hora de fer les comprovacions:

- Les quatre cantonades del tauler.
- Posicions als límits del tauler (a dalt, a baix, dreta i esquerra).

Funció encaixarFigura():

La funció encaixarFigura s'utiltiza per encaixar una figura en el tauler en la posició on es troba actualment. Aquesta funció es cridarà quan una figura no pugui baixar més posicions, tant si ha arribat al final del tauler com si ha sota seu es troba una figura ja col·locada.

La funció rep per paràmetre la figura que s'ha d'encaixar al tauler. La resta de dades que necessita per funcionar, com son la seva posició actual o la màscara de la seva rotació actual, les pot obtenir a partir de la figura.

Per aquesta funció, provarem d'encaixar cada una de les figures que tenim en el joc.

2. Classe Figura

Funció moure():

La funció moure s'encarrega de gestionar el moviments horitzontals de la figura. Aquests moviments són els que indica el jugador mitjançant les tecles de moviment esquerra o dret. Quan es vol moure una figura, primer es comprova si amb el moviment a realitzar solaparà d'alguna manera amb el tauler i el seu contingut. En cas de que no solapi, es realitzarà el moviment.

Es rep com a paràmetre els moviment a realitzar i el tauler per fer les comprovacions necessàries.

Les proves que farem per aquesta funció amb un tauler buit són:

- Posició inicial al centre del tauler, es pot moure cap a esquerra i dreta.
- Posició inicial al límit esquerra del tauler, només podem moure cap a la dreta.
- Posició inicial al límit dret del tauler, només podem moure cap a l'esquerra.

També cal provar les interaccions amb figures ja encaixades del tauler.

Funció rotar():

La funció rotar gestiona les rotacions de la figura que s'està movent. Quan el jugador clica la tecla de rotar, es crida aquesta funció, que comprova si amb la següent rotació de la figura aquesta solaparà amb algun element del tauler. En cas de que no solapi amb res, realitza el gir.

Es rep com a paràmetre el tauler de joc per a verificar si es pot realitzar o no el gir.

Per aquest mètode farem primer comprovacions amb el tauler buit i després amb un tauler amb figures ja encaixades. Depenent de la figura, aquesta pot tenir només 1 rotació, 2 o 4. Hem buscat tots els casos per cada una de les rotacions i figures on es pugui rotar i on no.

Funció caure():

La funció caure fa que la figura que s'estigui movent baixi una posició respecte a la actual. S'utilitzarà aquesta funció en cada torn i quan el jugador cliqui la tecla de baixar. Abans de realitzar el moviment, es comprova si és possible fer-ho perquè no solapa amb cap element.

S'envia el tauler de joc com a paràmetre per comprovar el solapament.

A l'hora de gestionar la caiguda, cal tenir en compte la situació del tauler. Per tant, haurem de fer probes tant per un taulell buit com un tauler amb figures ja encaixades. S'ha de tenir en compte l'alçada del tipus de peça que estem provant.

3. Classe GenaradorFiguraAleatoria

Funció novaFigura():

La funció novaFigura s'encarrega de donar al sistema l'índex de la figura que toca generar. Tot i que la classe es diu "Aleatoria", no ho és del tot. L'elecció de la figura es fa mitjançant una Collection on es guarden tots els índex de els figures de manera desordenada de manera aleatòria. Cada cop que es necessita una peça nova se li demana a aquesta funció, que treu el primer índex que troba dins de la Collection. Quan es vol treure un nou índex i la Collection es troba buida, es tornen a posar tots els índex i es barregen tots.

Aquest sistema fa que el jugador no pugui saber quina figura li sortirà a continuació, però també evita que una figura surti molts pocs cops.

Per provar aquesta funció, el que hem fet ha sigut demanar-li 6 vegades seguides (tants cops com figures diferents existeixen), i mitjançant un array de booleans de la mateixa mida, hem mirat quines figures han sortit o no. Cal que, després de haver executat la funció 6 vegades, hagin sortit totes les peces del joc.

Aquesta prova l'hem realitzat 40 vegades per comprovar que funcioni correctament.

Proves de caixa blanca

Hem utilitzat les tècniques de caixa blanca en les parts del codi on teniem funcions amb una certa complexitat i podiem analitzar l'algoritme més detingudament. Les funcions que hem considerat testejar son les de solapa(),comprovarFilesCompletades() i eliminarFilaCompletada().

Hem controlat el control flow amb Eclemma:

Coverage a la clase Figura:

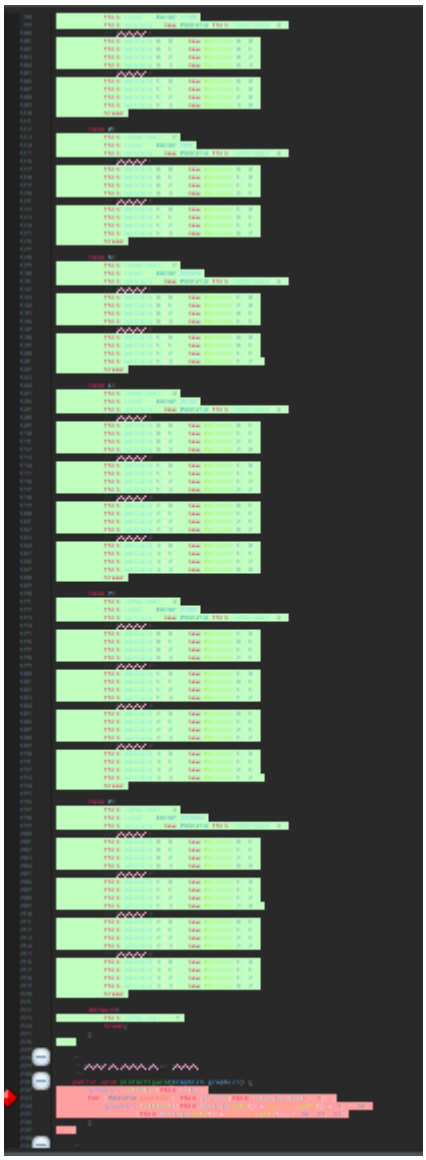


Figura.java	94,9 %	1.036	56	1.092

El coverage que aconseguim amb Eclemma a la clase figura executant el test de figura, es de 94,9% , pero en realitat el percentatge hauría de ser més gran ja que hi ha dues funcions que utilitzem pero que no s'executen en el test de figura, sinó en el test de tauler, que son les següents: (setPosicio i getColor)

```
    public void setPosicio(Posicio posicio) {
        this.posicio = posicio;
    }

    /*
     * Màscara amb les posicions de la Figura segons les rotacions que
     * pot tenir.
     */
    private Posicio[][] mascara;

    // Retorna la màscara corresponent a la rotació actual.
    public Posicio[] getMascara() {
        return this.mascara[this.rotacioActual - 1];
    }

    /*
     * Color de la Figura.
     */
    private Color color;

    public Color getColor() {
        return this.color;
    }
```

Quan executem el test de tauler, comprovem que realment si que passem per aquestes funcions:

```
    public void setPosicio(Posicio posicio) {
        this.posicio = posicio;
    }

    /*
     * Màscara amb les posicions de la Figura segons les rotacions que
     * pot tenir.
     */
    private Posicio[][] mascara;

    // Retorna la màscara corresponent a la rotació actual.
    public Posicio[] getMascara() {
        return this.mascara[this.rotacioActual - 1];
    }

    /*
     * Color de la Figura.
     */
    private Color color;

    public Color getColor() {
        return this.color;
    }
```

Només hi ha una funció que no hem pogut testejar, ja que aquesta funció és una funció que no podem cridar nosaltres, sinó que és una funció que criden les llibreries que hem utilitzat per a fer el programa, de forma interna. Aquelles funcions que son per pintar, no les podem cridar de forma implícita, sino que son les llibreries les que les criden:

```
    public void pintarFigura(Graphics graphics) {
        graphics.setColor(this.color);
        for (Posicio posicio : this.mascara[this.rotacioActual - 1]) {
            graphics.fillRect((this.posicio.getX() + posicio.getX() + 1) *
                    (this.posicio.getY() + posicio.getY()) * 26, 25, 25);
        }
    }
```

Coverage a la clase Tauler:

Tauler.java	70,2 %	268	114	382

A la classe de Tauler, hem pogut testejar totes les funcions excepte dues funcions de pintar en el tauler (pintarTauler() i pintarPuntuacio()), les quals com hem explicat préviament, son

funcions que s'utilitzen internament per les llibrerias, i son cridades per aquestes a l'hora de l'execució:

```java
public void pintarPuntuacio(Graphics graphics, int puntuacio) {
    graphics.setColor(Color.WHITE);
    graphics.drawString("Puntuacio: " + puntuacio, 25, 600);
}
```

```java
public void pintarTauler(Graphics graphics) {
    // Pintar fons del tauler (buït)
    graphics.setColor(Color.BLACK);
    graphics.fillRect(this.posicio.getX(), this.posicio.getY(),
            this.amplada, this.alcada);

    // Pintar limits tauler i el seu contingut
    for (int fila = 0; fila < nFiles; fila++) {
        graphics.setColor(Color.GRAY);
        graphics.fillRect(0, fila * 26, this.midaCasella, this.midaCase
```

Totes les demés funcions están testejades amb el coverage covert.

Coverage a la clase GeneradorFiguraAleatoria:

GeneradorFiguraAleatoria.java	100,0 %	70

```java
package tetris;

import java.util.ArrayList;

public class GeneradorFiguraAleatoria {

    private ArrayList<Integer> figuraSeguent = new ArrayList<Integer>();

    /*
     * Genera un nou índex de figura de manera "aleatoria".
     * Aquest sistema fa que el jugador no pugui saber quina figura li sortirà a
     * continuació, però també evita que una figura surti molts pocs cops.
     */
    public int novaFigura() {
        if (this.figuraSeguent.isEmpty()) {
            Collections.addAll(this.figuraSeguent, Figura.O, Figura.I, Figura.Z, Figura.S,
                    Figura.L, Figura.T, Figura.P);
            Collections.shuffle(this.figuraSeguent);

        }
        int novaFigura = this.figuraSeguent.get(0);
        this.figuraSeguent.remove(0);
        return novaFigura;
    }


}

```

A la clase GeneradorFiguraAleatoria aconseguim un coverage del 100%.

1. Classe Tauler

Funció solapa:

```
73      public boolean solapa(Figura figura, int movimentX, int movimentY) {
74          boolean solapaBool = false;
75          for (Posicio posicioMascara : figura.getMascara()) {
76              Posicio posicioAComprovar =
77                      new Posicio(posicioMascara.getX() + figura.getPosicio().getX() + movimentX,
78                      posicioMascara.getY() + figura.getPosicio().getY() + movimentY);
79              if (posicioValida(posicioAComprovar)) {
80                  if (tauler[posicioAComprovar.getY()]
81                          [posicioAComprovar.getX()] != Color.BLACK) {
82                      solapaBool = true;
83                  }
84              } else {
85                  solapaBool = true;
86              }
87          }
88          return solapaBool;
89      }
90
```

-*Statement coverage:* Totes les sentències s'estan executant com a mínim un cop.

-*Decision coverage:* Totes les decisions prenen els valors true i false.

-*Condition coverage:* Com en aquest codi no tenim cap condició amb més d'una decisió, el fet de que es compleixi el decision coverage ja ens garantitza que s'està complint el condition coverage.

-*Loop testing:* Hem realitzat loop testing per al bucle for de la línea 74. Com que aquest bucle depén del nombre de posicions, i el Tetris només té figures amb 1, 2 o 4 rotacions, només podem probar el bucle per aquests tres casos.

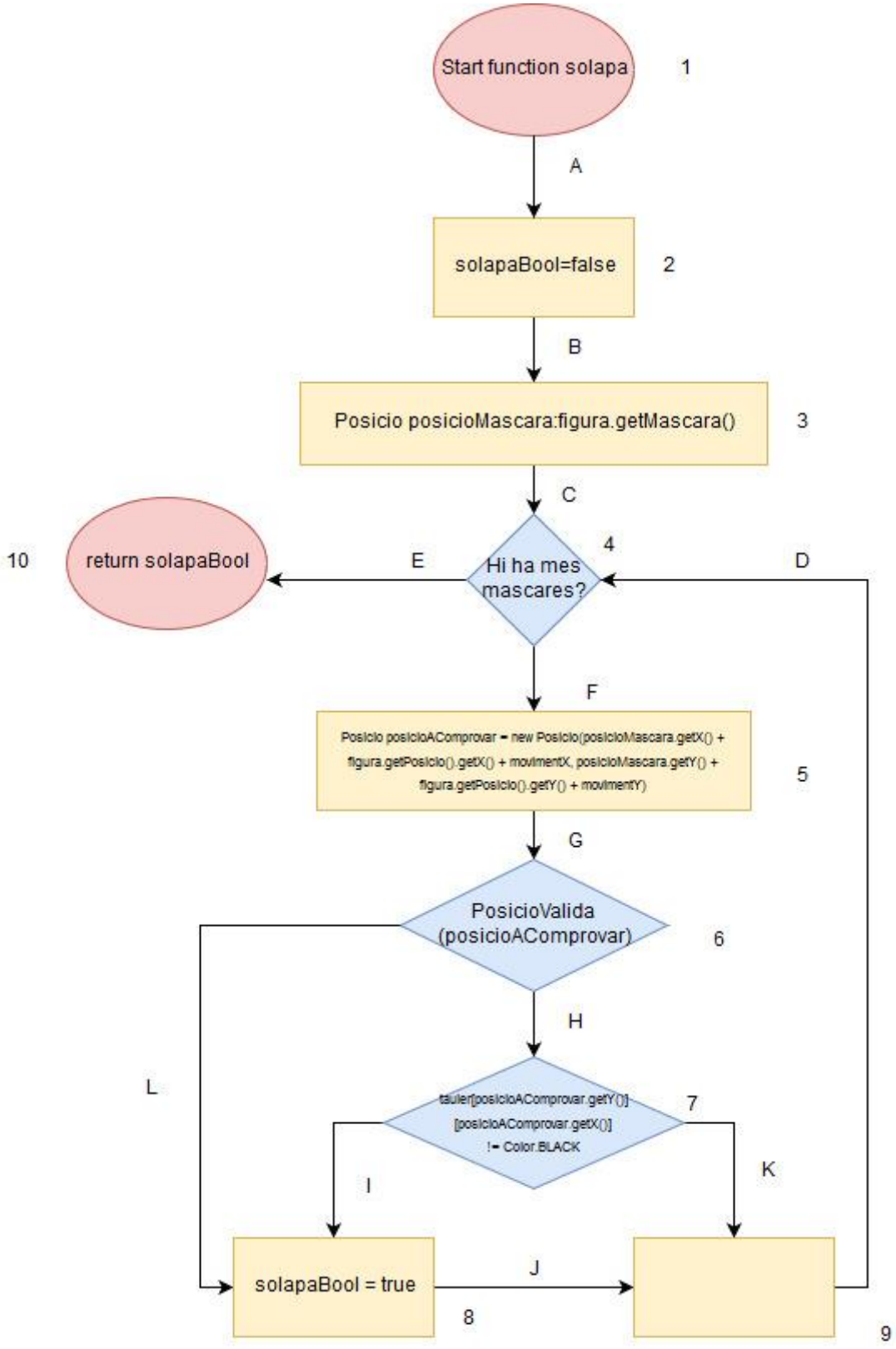
-*Path coverage:* Les proves realitzades compleixen tots els paths independents de la funció.

Complexitat ciclomàtica:

M = num. de condicions + 1 = 3 + 1 = 4

O també:

M = num.arestes - num.nodes + 2 = 12 - 10 + 2 = 4



1. (1A, 2B, 3C, 4E, 10)

2. (1A, 2B, 3C, 4F, 5G, 6H, 7I, 8J, 9D, 4E, 10)

3. (1A, 2B, 3C, 4F, 5G, 6L, 8J, 9D, 4E, 10)

4. (1A, 2B, 3C, 4F, 5G, 6H, 7K, 9D, 4E, 10)

Funció comprovarFilesCompletades():

```
117     public int comprovarFilesCompletades() {
118         int filesCompletades = 0;
119         boolean forat;
120         for (int fila = nFiles - 1; fila >= 0; fila--) {
121             forat = false;
122             for (int columna = 0; columna < nColumnes; columna++) {
123                 if (this.tauler[fila][columna] == Color.BLACK) {
124                     forat = true;
125                     break;
126                 }
127             }
128             if (!forat) {
129                 eliminarFilaCompletada(fila);
130                 filesCompletades++;
131                 fila++;
132             }
133         }
134         return filesCompletades;
135     }
136
```

-*Statement coverage:* Totes les sentències s'estan executant com a mínim un cop.

-*Decision coverage:* Totes les decisions prenen els valors true i false.

-*Condition coverage:* Com en aquest codi no tenim cap condició amb més d'una decisió, el fet de que es compleixi el decision coverage ja ens garantitza que s'està complint el condition coverage.

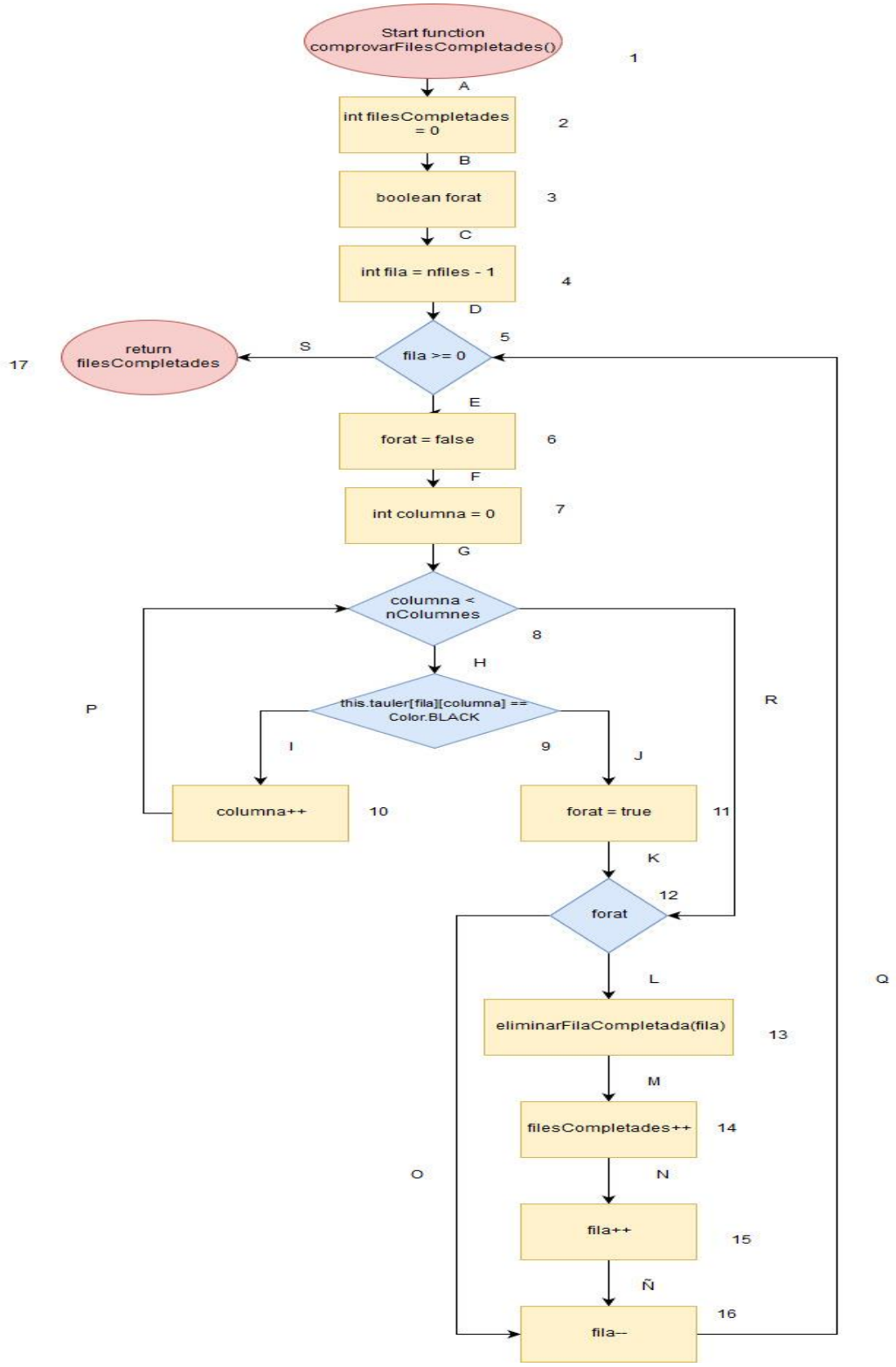
-*Path coverage:* Les proves realitzades compleixen tots els paths independents de la funció.

Complexitat ciclomàtica:

M = num. de condicions + 1 = 4 + 1 = 5

O també:

M num.arestes - num.nodes + 2 = 20 - 17 + 2 = 3 + 2 = 5



1. (1A, 2B, 3C, 4D, 5S, 17)

2. (1A, 2B, 3C, 4D, 5E, 6F, 7G, 8H, 9J, 11K, 12L, 13M, 14N, 15Ñ, 16Q, 5S, 17)

3. (1A, 2B, 3C, 4D, 5E, 6F, 7G, 8R, 12L, 13M, 14N, 15Ñ, 16Q, 5S, 17)

4. (1A, 2B, 3C, 4D, 5E, 6F, 7G, 8H, 9J, 11K, 12O, 16Q, 5S, 17)

5. (1A, 2B, 3C, 4D, 5E, 6F, 7G, 8H, 9I, 10P, 8H, 9J, 11K, 120, 16Q, 5S, 17)

Funció eliminarFilaCompletada():

```
public void eliminarFilaCompletada(int filaAEliminar) {
    if (filaAEliminar != 0) {
        for (int fila = filaAEliminar; fila > 0; fila--) {
            for (int columna = 0; columna < nColumnes; columna++) {
                this.tauler[fila][columna] = this.tauler[fila - 1][columna];
            }
        }
        for (int columna = 0; columna < nColumnes; columna++) {
            this.tauler[0][columna] = Color.BLACK;
        }
    } else {
        for (int columna = 0; columna < nColumnes; columna++) {
            this.tauler[0][columna] = Color.BLACK;
        }
    }
}
```

-*Statement coverage:* Totes les sentències s'estan executant com a mínim un cop.

-*Decision coverage:* Totes les decisions prenen els valors true i false.

-*Condition coverage:* Com en aquest codi no tenim cap condició amb més d'una decisió, el fet de que es compleixi el decision coverage ja ens garantitza que s'està complint el condition coverage.

-*Loop testing:* Testejarem el primer bucle de tots (que depén de filaAEliminar). Primer no entrarem al bucle, després farem una passada, dues, 10 passades i nFiles – 1 passades.

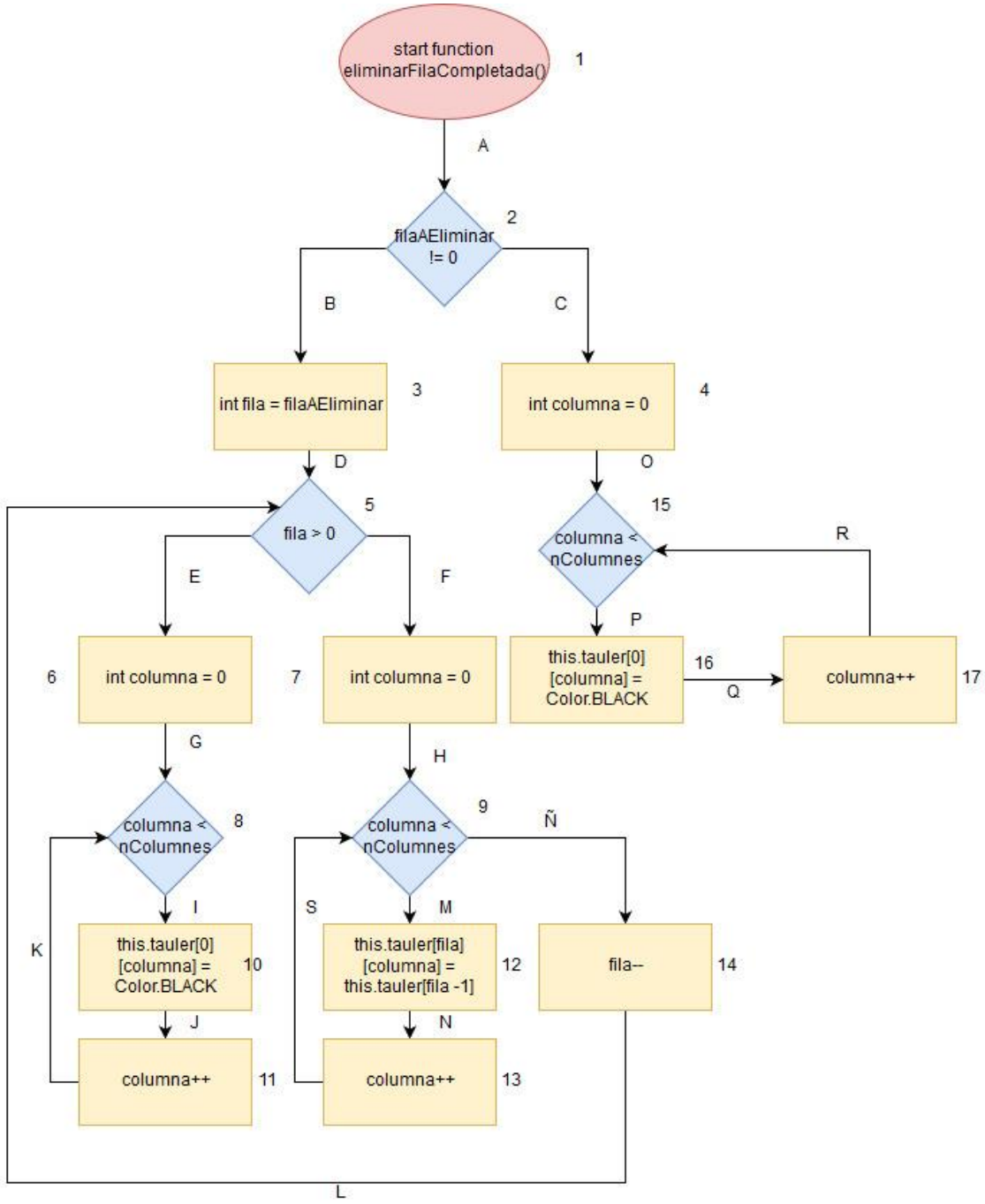
-*Path coverage:* Les proves realitzades compleixen tots els paths independents de la funció.

Complexitat ciclomàtica:

M = num. de condicions + 1 = 5 + 1 = 6

O també:

M num.arestes - num.nodes + 2 = 21 - 17 + 2 = 4 + 2 = 6



1. (1A, 2C, 4O, 15P, 16Q, 17R)

2. (1A, 2B, 3D, 5F, 7H)

3. (1A, 2B, 3D, 5E, 6G)

4. (1A, 2B, 3D, 5E, 6G, 8I, 10J, 11K, 8)

5. (1A, 2B, 3D, 5F, 7H, 9M, 12N, 13

6. (1A, 2B, 3D, 5F, 7H, 9Ñ, 14L, 5F, 7H, 9M, 12N, 13S, 9)

Funció posicioValida():

```
public boolean posicioValida(Posicio posicio) {
    if (posicio.getX() >= 0 && posicio.getX() <= (nColumnes - 1) &&
            posicio.getY() >= 0 && posicio.getY() <= (nFiles - 1)) {
        return true;
    } else {
        return false;
    }
}
```

-*Statement coverage:* Totes les sentències s'estan executant com a mínim un cop.

-*Decision coverage:* Totes les decisions prenen els valors true i false.

-*Condition coverage:* Com en aquest codi no tenim cap condició amb més d'una decisió, el fet de que es compleixi el decision coverage ja ens garantitza que s'està complint el condition coverage.

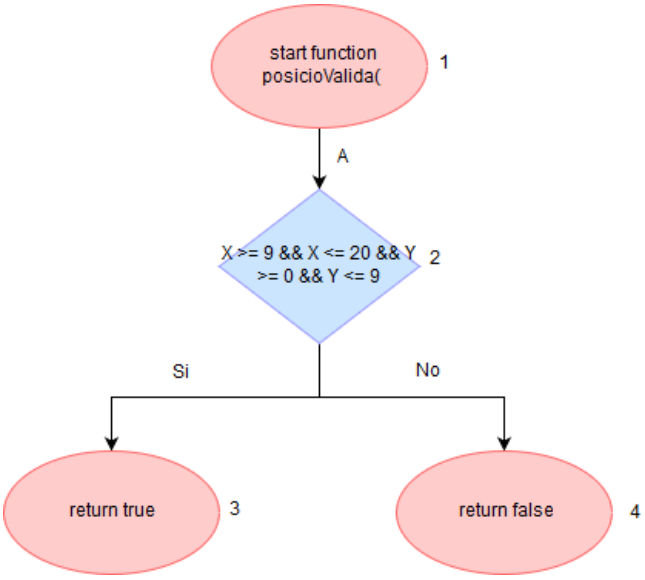
-*Path coverage:* Les proves realitzades compleixen tots els paths independents de la funció.

Complexitat ciclomàtica:

M = num. de condicions + 1 = 1 + 1 = 2

O també:

M num.arestes - num.nodes + 2 = 3 - 3 + 2 = 0 + 2 = 2



1. (1A, 2-Si, 3)
2. (1A, 2-No, 4)

Funció encaixarFigura():

```
public void encaixarFigura(Figura figura) {
    for (Posicio posicio : figura.getMascara()) {
        this.tauler[figura.getPosicio().getY() + posicio.getY()]
                [figura.getPosicio().getX() + posicio.getX()] = figura.getColor();
    }
}
```

-*Statement coverage:* Totes les sentències s'estan executant com a mínim un cop.

-*Decision coverage:* Com que en aquest codi no hi ha cap condició, el decision coverage no es pot complir.

-*Condition coverage:* Com en aquest codi no hi ha cap condició, el decision coverage no es pot complir.

-*Loop testing:* Com que el bucle de la funció depén del nombre de posicions, i el Tetris només té figures amb 1, 2 o 4 rotacions, només podem probar el bucle per aquests tres casos.

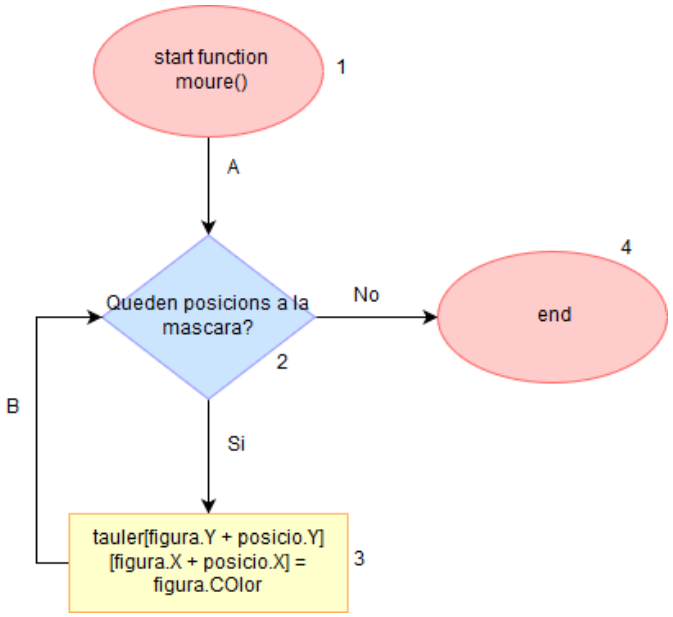
-*Path coverage:* Les proves realitzades compleixen tots els paths independents de la funció.

Complexitat ciclomàtica:

M = num. de condicions + 1 = 1 + 1 = 2

O també:

M num.arestes - num.nodes + 2 = 4 - 4 + 2 = 0 + 2 = 2



1. (1A, 2-No, 4)

2. (1A, 2-Si, 3-B, 2-No, 4)

2. Classe Figura

Funció moure():

```
public void moure(int movimentHoritzontal, Tauler tauler) {
    if (!tauler.solapa(this, movimentHoritzontal, 0)) {
        this.posicio.setX(this.posicio.getX() + movimentHoritzontal);
    }
}
```

-*Statement coverage:* Totes les sentències s'estan executant com a mínim un cop.

-*Decision coverage:* Totes les decisions prenen els valors true i false.

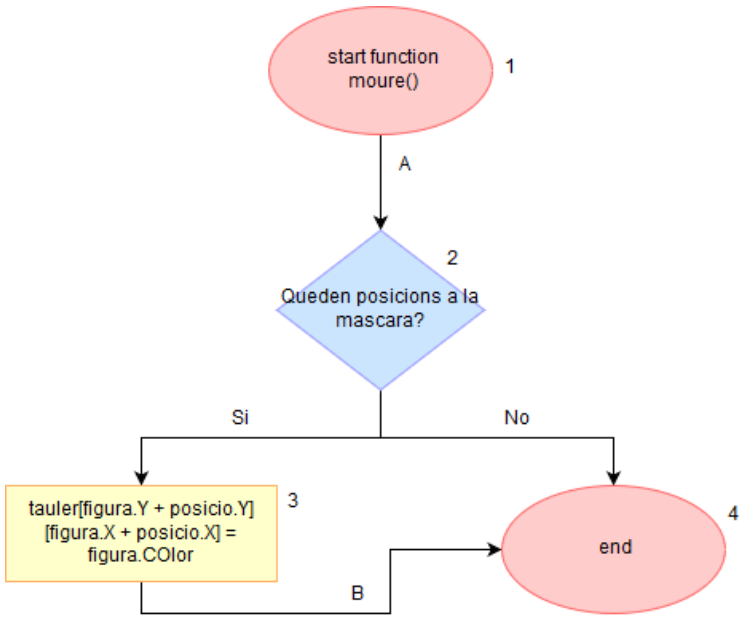
-*Condition coverage:* Com en aquest codi no tenim cap condició amb més d'una decisió, el fet de que es compleixi el decision coverage ja ens garantitza que s'està complint el condition coverage.

-*Path coverage:* Les proves realitzades compleixen tots els paths independents de la funció.

Complexitat ciclomàtica:
M = num. de condicions + 1 = 1 + 1 = 2

O també:

M num.arestes - num.nodes + 2 = 4 - 4 + 2 = 0 + 2 = 2



1. (1A, 2-No, 4)

2. (1A, 2-Si, 3-B, 4)

Funció rotar():

```
public void rotar(Tauler tauler) {
    int possibleRotacio = ((this.rotacioActual) % this.rotacions) + 1;
    Figura figuraGirada = new Figura(this.indexFigura, this.posicio);
    figuraGirada.setRotacioActual(possibleRotacio);
    if (!tauler.solapa(figuraGirada, 0, 0)) {
        this.rotacioActual = possibleRotacio;
    }
}
```

-*Statement coverage:* Totes les sentències s'estan executant com a mínim un cop.

-*Decision coverage:* Totes les decisions prenen els valors true i false.

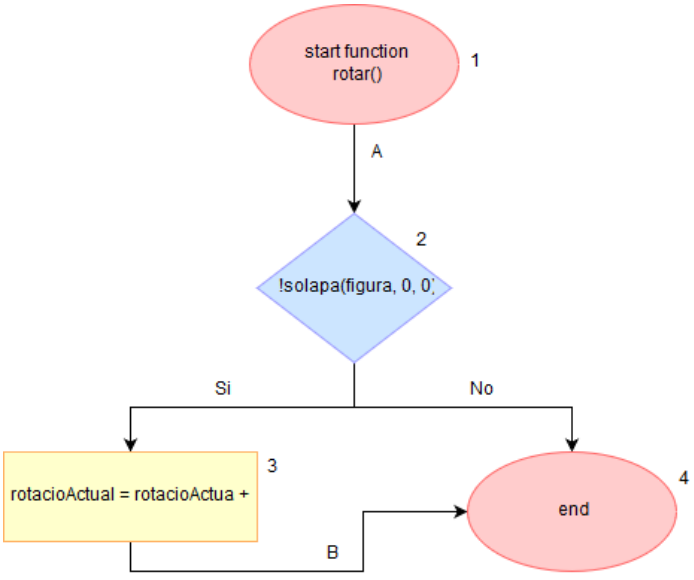
-*Condition coverage:* Com en aquest codi no tenim cap condició amb més d'una decisió, el fet de que es compleixi el decision coverage ja ens garantitza que s'està complint el condition coverage.

-*Path coverage:* Les proves realitzades compleixen tots els paths independents de la funció.

Complexitat ciclomàtica:
M = num. de condicions + 1 = 1 + 1 = 2

O també:

M num.arestes - num.nodes + 2 = 4 - 4 + 2 = 0 + 2 = 2



1. (1A, 2-No, 4)

2. (1A, 2-Si, 3-B, 4)

Funció caure():

```
public boolean caure(Tauler tauler) {
    if (this.posicio.getY() + 1 <= Tauler.nFiles - 1) {
        if (!tauler.solapa(this, 0, 1)) {
            this.posicio.setY(this.posicio.getY() + 1);
            return true;
        }
    }
    return false;
}
```

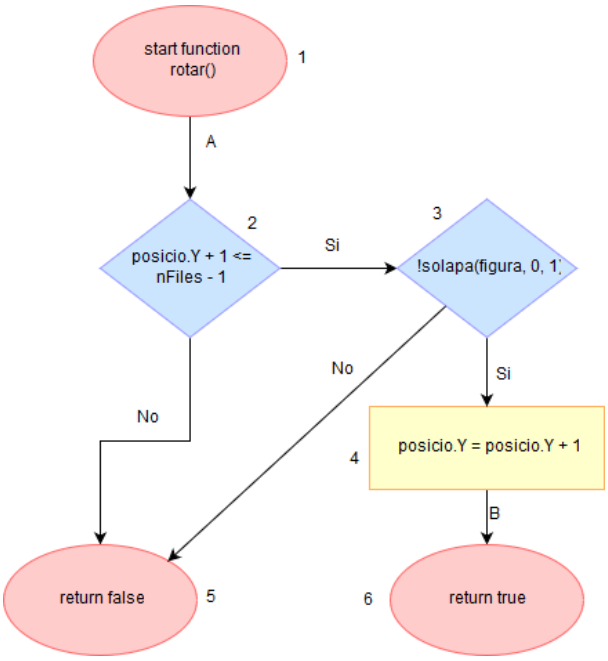
-*Statement coverage:* Totes les sentències s'estan executant com a mínim un cop.

-*Decision coverage:* Totes les decisions prenen els valors true i false.

-*Condition coverage:* Com en aquest codi no tenim cap condició amb més d'una decisió, el fet de que es compleixi el decision coverage ja ens garantitza que s'està complint el condition coverage.

-*Path coverage:* Les proves realitzades compleixen tots els paths independents de la funció.

Complexitat ciclomàtica:
M = num. de condicions + 1 = 2 + 1 = 3



1. (1A, 2-No, 5)

2. (1A, 2-Si, 3-No, 5)

3. (1A, 2-Si, 3-Si, 4-B, 6)

3. Classe GeneradorFiguraAleatoria

Funció novaFigura():

```
public int novaFigura() {
    if (this.figuraSeguent.isEmpty()) {
        Collections.addAll(this.figuraSeguent, Figura.O, Figura.I, Figura.Z, Figura.S,
                Figura.L, Figura.T, Figura.P);
        Collections.shuffle(this.figuraSeguent);

    }
    int novaFigura = this.figuraSeguent.get(0);
    this.figuraSeguent.remove(0);
    return novaFigura;
}
```

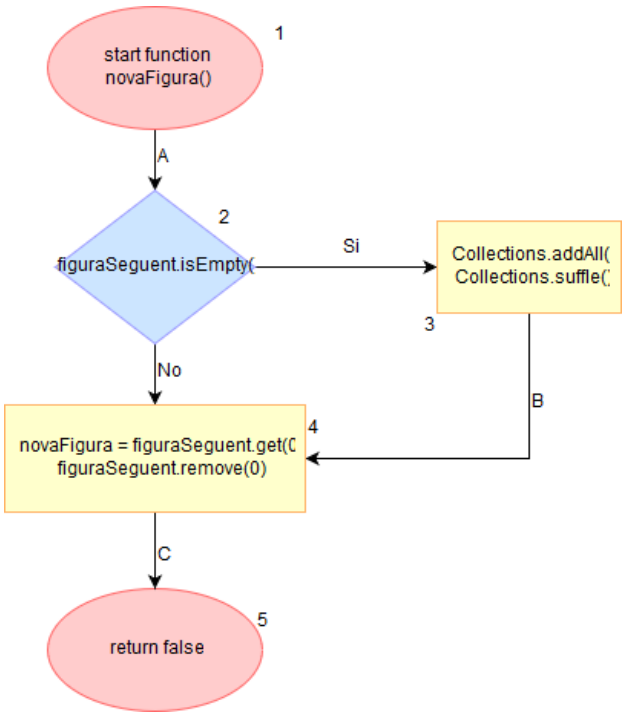
-*Statement coverage:* Totes les sentències s'estan executant com a mínim un cop.

-*Decision coverage:* Totes les decisions prenen els valors true i false.

-*Condition coverage:* Com en aquest codi no tenim cap condició amb més d'una decisió, el fet de que es compleixi el decision coverage ja ens garantitza que s'està complint el condition coverage.

-*Path coverage:* Les proves realitzades compleixen tots els paths independents de la funció.

Complexitat ciclomàtica:
M = num. de condicions + 1 = 1 + 1 = 2

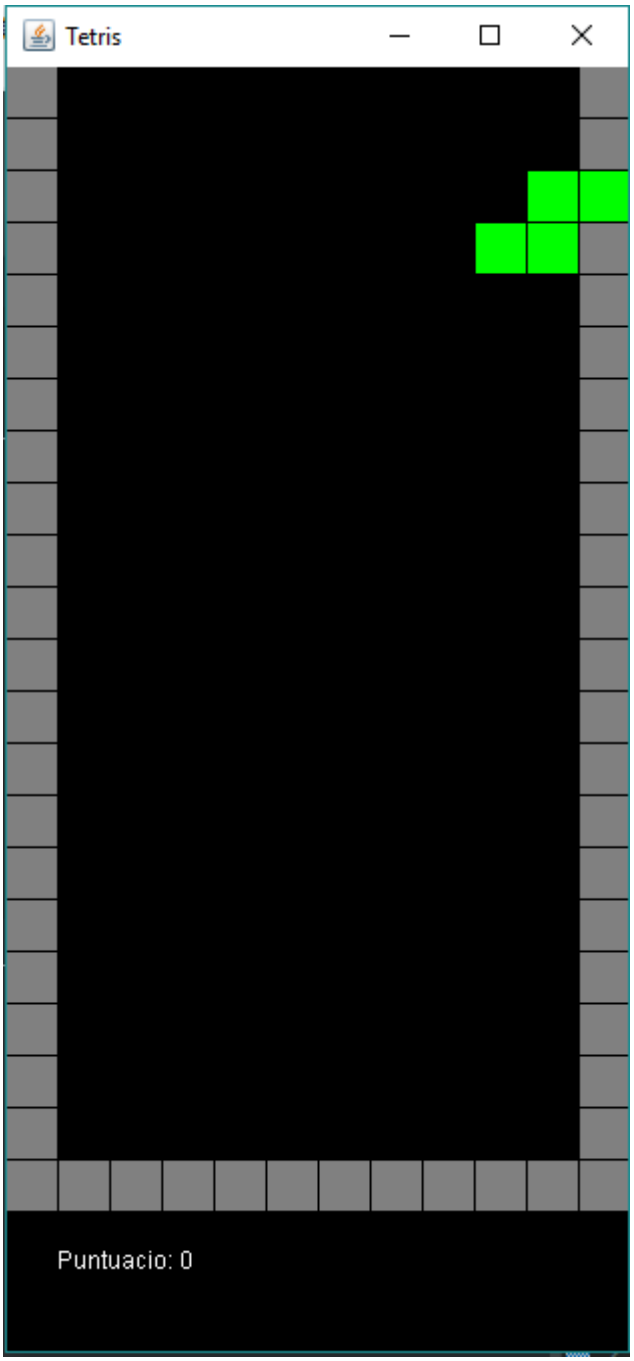


1. (1A, 2-No, 4-C, 5)

2. (1A, 2-Si, 3-B, 4-C, 5)

Exploratory testing

Error 1:




```
Joc [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\javaw.exe (11 ene. 2018 16:27:29)
Exception in thread "Thread-2" java.lang.ArrayIndexOutOfBoundsException: 10
        at tetris.Tauler.encaixarFigura(Tauler.java:108)
        at tetris.Joc$1.run(Joc.java:42)
        at java.lang.Thread.run(Unknown Source)
```

Passos seguits per arribar a aquest estat:

Posicionem la figura al límit dret polsant la tecla de direccionament dreta, i fem totes les rotacions possibles polsant la tecla de direcció cap amunt, si alguna de les figures al rotar solapa amb la paret, aquesta figura traspassarà la paret i sortirà una excepció indicant que ens hem sortit del array, ja que les posicions de les parets no estan permeses i intentarà accedir a una posició no permesa.

Nota: Aquest error s'ha solucionat a la versió final del joc. Adjuntem el codi que amb l'error i el corregit a les imatges.

A que es degut?

Aquest error es degut a que en la funció de rotar no s'estava actualitzant correctament la rotació actual que tenia la figura.

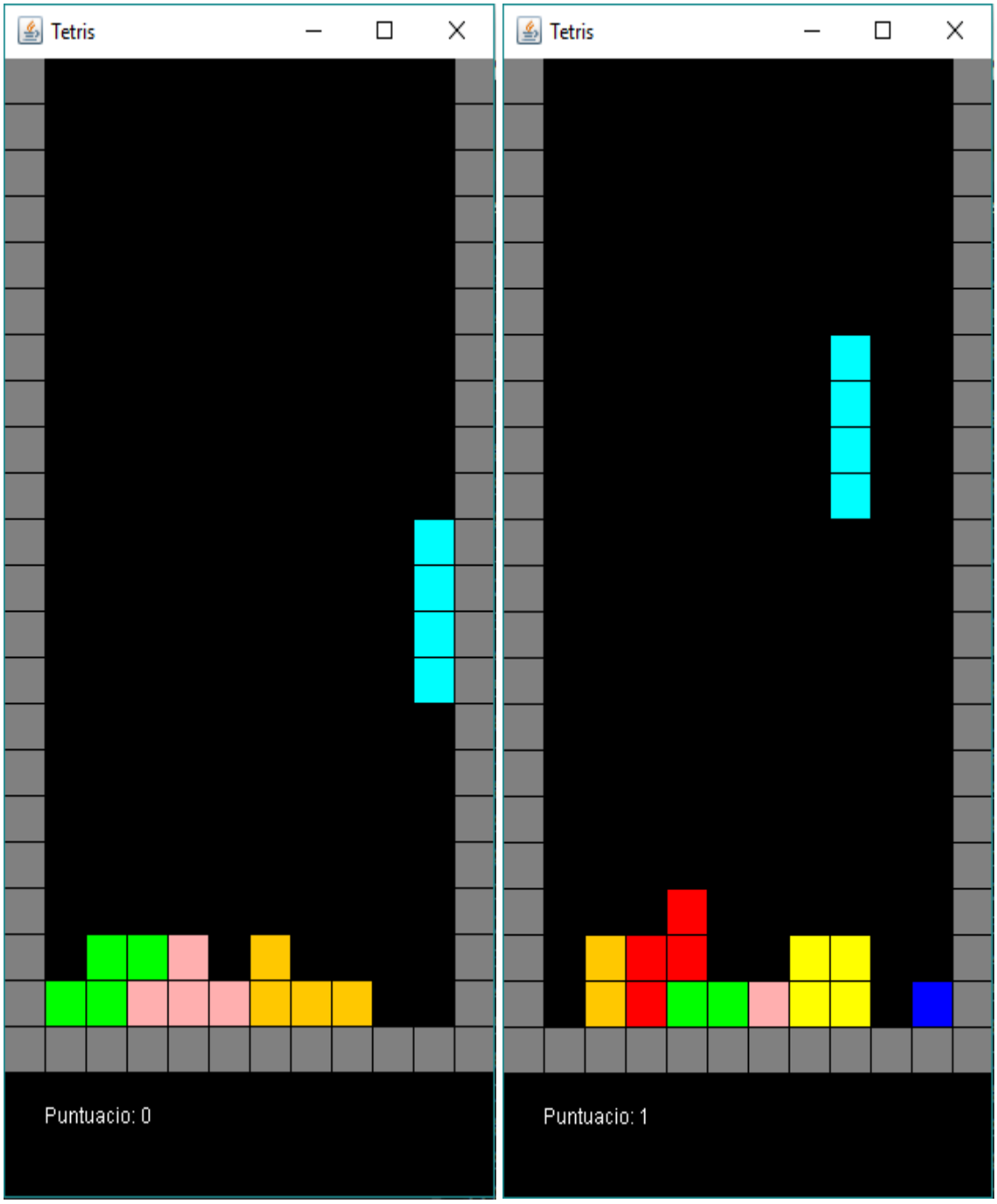
Versió de la funció amb l'error:

```
public void rotar(Tauler tauler) {
    int possibleRotacio = ((this.rotacioActual) % this.rotacions) + 1;
    if (!tauler.solapa(this, 0, 0)) {
        this.rotacioActual = possibleRotacio;
    }
}
```

Versió de la funció amb error corregit:

```
public void rotar(Tauler tauler) {
    int possibleRotacio = ((this.rotacioActual) % this.rotacions) + 1;
    Figura figuraGirada = new Figura(this.indexFigura, this.posicio);
    figuraGirada.setRotacioActual(possibleRotacio);
    if (!tauler.solapa(figuraGirada, 0, 0)) {
        this.rotacioActual = possibleRotacio;
    }
}
```

Error 2:



Passos a seguir per arribar a aquest estat:

Col·loquem la figura al límit dret polsant la tecla de direccionalment dreta, o a dos posicions del límit i quan intentem rotar veiem que no la figura no rota.

A que es degut?

Aquest error es degut a que segurament el centre de la màscara de la figura no està ben programat.

Mock objects

Hem fet un mock object per a la classe de generació de figures aleatòries, ja que necessitàvem fer proves amb jubula, i amb un generador aleatori, no podíem aconseguir fer una partida, ja que cada partida era diferent.

El codi del mock object es el següent:

```java
package tetris;
import java.util.ArrayList;


public class MockGeneradorFiguraAleatoria {

    private ArrayList<Integer> figuraSeguent = new ArrayList<Integer>();
    /*
     * Genera un array amb les figures sense barrejarles, per tal de poder
     * fer una partida al jubula amb unes figures ple-establertes
     */
    public int novaFigura() {
        if (this.figuraSeguent.isEmpty()) {
            Collections.addAll(this.figuraSeguent, Figura.O, Figura.I, Figura.Z, Figura.S,
                    Figura.L, Figura.T, Figura.P);
        }
        int novaFigura = this.figuraSeguent.get(0);
        this.figuraSeguent.remove(0);
        return novaFigura;
    }
}
```

Ens estalviem la línia de codi on barregem les figures inserides en la col·lecció i d'aquesta manera sempre utilitzarem les mateixes peces.

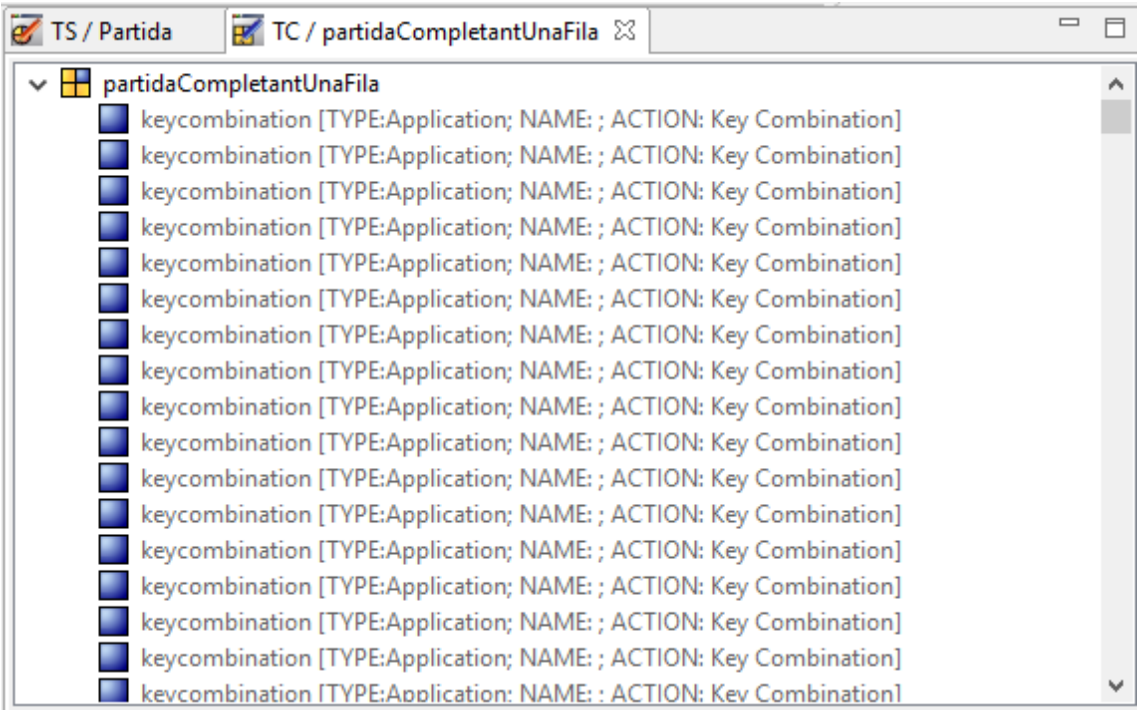
```java
import java.awt.Graphics;

public class Joc extends JPanel {

    private static final long serialVersionUID = 1L;

    private static boolean esPotMoure;
    private static Figura figura;
    private static Tauler tauler = new Tauler();
    private static MockGeneradorFiguraAleatoria generador = new MockGeneradorFiguraAleatoria();
    private static int puntuacio = 0;
    private static boolean perdut = false;
```

En la classe joc, substituïm el generador per el mock object, crearem un Jar amb el mock object per fer proves al jubula, i després tornarem a utilitzar el generador original.

Test Automation

Hem fet una automatització de partida amb jubula, on aconseguim que es faci una fila, per comprovar que la puntuació funciona correctament, i que les files al ser completades son eliminades, i un cop feta la fila, fem moviments al atzar aproximant les figures als límits per comprovar que la solapació i rotació funciona correctament i després simplement deixem caure les figures fins detecti una derrota i acabi la partida.

El test case per a fer la partida no s'ha fet en el mode de gravació, sinó que s'han posat els moviments manualment per tal de aconseguir fer una fila.



En el següent link es pot veure l'execució de la automatització de jubula:

https://drive.google.com/file/d/198ndo6qWpIJ7aUuIE-J__SztaCg933qf/view

Inspection Issue Log

Project: Tetris	**Origin:**	**R**equirements, **D**esign, **C**onstruction, **T**esting
Inspection ID: 001	**Type:**	**M**issing, **W**rong, **E**xtra, **U**sability, **P**erformance, **S**tyle, **C**larity, **Q**uestion
Meeting Date: **Recorder:** 11/01/2018	**Severity:**	**M**ajor, **m**inor

Defects Found: _____ , Major _____ , minor _____ .

#	Origin	Type	Severity	Location	Description
1	D	C	m	Figura.java, 52	El nom de la variable no és gaire clar. S'hauria d'especificar millor que es tracta del nombre total de rotacions que pot fer la Figura.
2	D	C	m	Figura.java, 33	El nom de la variable genera confusió, sembla que sigui una única màscara, però en realitat guarda un conjunt.
3	C	S	m	Figura.java, 270	S'efectua un return dins d'una condició if. Mala pràctica.
4	D	C	m	Tauler.java, 11	El nom de la variable no distingeix si es tracta d'un atribut relacionat amb la matriu del Tauler o dels gràfics del Tauler.
5	D	C	m	Tauler.java, 20	El nom "tauler" es pot confondre amb un objecte de la classe Tauler.java, quan en realitat es tracta de una matriu de Colors.

Inspection Summary Report

Inspection Identification:

Project: Tetris
Inspection ID: 001
Meeting Date: 11/01/2018

Work Product Description: The product to be inspected is the game Tetris. It has been developed in Java and tested with JUnit.

	Inspectors	**Preparation Time**
Author:	Alex and Marc	2 hours
Moderator:	Alex	1 hours
Recorder:	Alex	1 hours
Reader:	Marc	1 hours
Inspector:	Marc	2 hours
Inspector:	Alex	2 hours

Inspection Data

Pages or Lines of Code:	Lines	Meeting Time:	3 hours
Planned for Inspection:	1570	Total Planning Effort:	3 labor hours
Actually Inspected:	1570	Total Overview Effort:	3 labor hours
		Total Preparation Effort:	2 labor hours
		Actual Rework Effort:	1 labor hours

Product Appraisal

ACCEPTED
X as is
___ conditionally upon verification
Verifier Alex, Marc

NOT ACCEPTED
___ reinspect following rework
___ inspection not completed

Projected Rework Completion Date: 11/01/2018