<Rubén Simó>

<Alejandro Sabater >

SnakeGame

ÍNDEX

Arxiu SnakePart

Funcionalitat: Constructor de SnakePart.

Localització: SnakePart.java, clase SnakePart i snakePart()

Funcionalitat: Constructor paràmetric de la SnakePart.

Localització: SnakePart.java, clase SnakePart i , SnakePart(xCoord, yCoord, tileSize).

Test: BoardTest.java, BoardTest, testInitialSnake(). Es realitza un test de caixa blanca amb la técnica statement coverage que comprova que el tamany inicial de la serp es 5.

Funcionalitat: Funció que s'encarrega de dibuixar la serp.

Localització: SnakePart.java, clase SnakePart i draw()

Arxiu Board

Funcionalitat: Constructor per defecte de Board que l'inicialitza.

Localització: Board.java, classe Board, Board().

Test: BoardTest.java, BoardTest, testInitialBoard(). Es realitza un test de caixa blanca amb la técnica statement coverage que comprova que el board creat pel constructor no sigui NULL.

Funcionalitat: La funció **paint**() s'encarrega de dibuixar el taulell on es troben la serp, les pomes, la puntuació i el nom de l'usuari.

Localització: Board.java, classe Board, paint(Graphics g).

Funcionalitat: La funció **start**() s'encarrega de començar la partida creant un thread.

Localització: Board.java, classe Board, start().

Test: BoardTest.java, BoardTest, testgetStartedGame(). Es realitza un test de caixa blanca amb la técnica statement coverage que comprova que el joc es troba en execució mitjançant la variable running, que ha de ser true després d'executar la función start().

Funcionalitat: La funció **stop**() s'encarrega de parar la partida

Localització: Board.java, classe Board, stop().

Test: BoardTest.java, BoardTest, testgetStartedGame(). Es realitza un test de caixa blanca amb la técnica statement coverage que comprova que el joc es troba en parat mitjançant la variable running, que ha de ser false després d'executar la funció stop().

```
@Test
public void testgetStartedGame() throws InterruptedException, IOException {
Board board = new Board();
assertEquals(board.running,  expected: true);
board.stop();
assertEquals(board.running,  expected: false);

}
```

Funcionalitat: La funció **update**() s'encarrega d'anar actualitzant el taulell constantment per a que el joc continui en funcionament.

Localització: Board.java, classe Board, update().

Test: BoardTest.java, BoardTest, testgetUpdateFunction(). Es realitza un test de caixa blanca amb la técnica statement coverage que després d'executar la funció update() comprova que els ticks vagin augmentant de valor fins arribar a 800000 mil·lisegons

```
@Test
public void testUpdateFunction() throws InterruptedException, IOException {
    Board board = new Board();
    int ticks_before = board.ticks;
    board.update();
    assertNotEquals(ticks_before, board.ticks);
}
```

Funcionalitat: La funció **updateSnake**(boolean right, boolean left, boolean down, boolean up) s'encarrega del moviment de la serp, es va actualitzant constantment calculant les noves coordenades de la posició en la que es troba.

Localització: Board.java, classe Board, updateSnake(boolean right, boolean left, boolean down, boolean up).

Funcionalitat: La funció **moveSnake**() s'encarrega d'anar actualitzant la serp eliminant l'ultima casella d'aquesta cada cop que avança una casella sense augmentar el seu tamany.

Localització: Board.java, classe Board, moveSnake().

Funcionalitat: La funció comproveLimits() s'encarrega de indicar si la serp esta fora dels límits del taulell.

Localització: Board.java, classe Board, comproveLimits().

Test: BoardTest.java, BoardTest, testgetComproveLimits(). Es realitza un test de caixa blanca amb la técnica statement coverage que comprova mitjançant 6 casos que les coordenades de la posició de la serp siguin dins del taulell. En el primer i el segon cas comprovem que les coordenades x i y siguin vàlides, en el tercer i el quart cas que les dues coordenades x i y estiguin fora del límits, en el cinquè cas la coordenada x es troba dins dels límits pero la y es troba fora i a l'últim cas pasa el contrari que l'anterior, la coordenada y es troba dins dels límits i la coordenada x es troba fora. BoardTest.java, BoardTest, testgetComproveLimitsStartingGame(). Test utilitzarparal·lelament per realitzar més proves sobre la mateixa funcionalitat.

Funcionalitat: La funció **comproveApplesInBoard**() s'encarrega de generar una poma en el cas de que al taulell no hi hagi cap.

Localització: Board.java, classe Board, comproveApplesInBoard().

Test: BoardTest.java, BoardTest, testComproveApplesInBoard(). Es realitza un test de caixa blanca amb la técnica statement coverage que comprova que una poma creada es trobi dins del taulell.

```
@Test
public void testComproveApplesInBoard() throws InterruptedException {
    Board board = new Board();
    boolean exists = board.comproveApplesInBoard();
    assertEquals(exists,  expected: true);
}
```

Funcionalitat: La funció **eat**() s'encarrega de recorrer el bucle del tamany de l'Snake i fer una comprovació de si la primera posició es situa a la mateixa on hi ha una poma. En cas que aquest sigui afirmatiu, fa una crida a la funció grow().

Localització: Board.java, classe Board, eat().

Test:BoardTest.java, BoardTest, testEat().Es realitza un test de caixa blanca amb la técnica statement coverage que comprova que el tamany del Snake augmentin en cas de trobar-se sobre una poma. En aquest cas, no fixem la posició de la poma i per això no augmenta, ja que no es troba a sobre de la poma.

```
@Test //False case.
public void testEat() throws InterruptedException
{
    Board board = new Board();
    int size = board.SnakeSize;
    board.eat();
    assertEquals(size,board.SnakeSize);
}
```

Funcionalitat: La funció **grow**() s'encarrega d'augmentar el tamany de la serp i la puntuació en 1, de reduir la seva velocitat i de eliminar la poma menjada.

Localització: Board.java, classe Board, grow().

Test:BoardTest.java, BoardTest, testGrowSnake(). Es realitza un test de caixa blanca amb la técnica statement coverage que comprova que després d'executar la funció grow(), el tamany de la serp incrementi en 1. BoardTest.java, BoardTest, testGrownPuntuation(). Paral·lelament hem realitzat aquest test que comprova que despres d'executar la comanda grow() comprova que la puntuació s'incrementi en 1.

```
@Test
public void testGrowSnake() throws InterruptedException {
    Board board = new Board();
    board.grow();
    assertEquals(board.SnakeSize, expected: 6);
}
@Test
public void testGrownPuntuation() throws InterruptedException {
    Board board = new Board();
    board.grow();
    assertEquals(board.puntuation, expected: 1);
}
```

Funcionalitat: La funció **lose_hitHimself**() s'encarrega de recorrer el bucle del array del tamany de l'Snake realitzant la comprovació en tot moment de si es xoca contra ella mateixa. En cas positiu, retorna true i perd el joc. En cas negatiu es retorna false i continua el joc.

Localització: Board.java, classe Board, lose_hitHimself().

Test:BoardTest.java, BoardTest, testLose_hitHimself(). Es realitza un test de caixa blanca amb la técnica statement coverage que comprova que indicant les coordenades de la serp per a que no estigui en contacte amb si mateixa i que després d'executar la funció lose_hitHimself() indiqui que no s'esta tocant a si mateixa.

```
@Test
public void testLose_hitHimself() throws InterruptedException {
    Board board = new Board();
    board.setCoordSnake_X(10);
    board.setCoordSnake_Y(10);
    boolean exists = board.lose_hitHimself();
    assertEquals(exists,  expected: false);

}
```

Funcionalitat: La funció **keyPressed**(KeyEvent e) s'encarrega d'indicar quina tecla ha estat pressionada comprovant les quatre opcions possibles, que hagi pressionat la flecha cap avall, la flecha cap amunt, la flecha cap a la dreta i la flecha cap a l'esquerra .

Localització: Board.java, classe Board, keyPressed(KeyEvent e).

Test:BoardTest.java, BoardTest, testKeyPressedDown(). Es realitza un test de caixa blanca amb la técnica statement coverage que comprova que la tecla pressionada es la flecha cap avall després d'executar la funció keyPressed(KeyPressed e) sent e la flecha inferior.

Arxiu JSON

Funcionalitat: La funcionalitat **loadJSONFile**() s'encarrega d'obrir l'arxiu database.json i transformar-lo en un objecte JSON. Poteriorment retorna l'objecte.

Localització: JSON.java, classe JSON, loadJSONFile().

Test:BoardTest.java, BoardTest, testLoadJSONFile(). Es realitza un test de caixa blanca amb la técnica statement coverage que carregam a memòria el fitxer "database.json". Posteriorment mitjançant l'assertEqual comprovem els tipus d'Objecte carregat per comprovar que es un Objecte JSON. Mitjançant una impressió per pantalla comprovem que el resultat és l'esperat.

```java
@Test
public void testLoadJSONFile() throws InterruptedException, IOException {

    JSON json = new JSON();
    JSONObject JSONLoaded = json.loadJSONFile();
    JSONObject jsonType = new JSONObject();
    System.out.println(JSONLoaded);
    assertEquals(JSONLoaded.getClass(), jsonType.getClass());
}
```

```
{"":0,"User2":2,"User1":15,"User3":3}

================================================
Default Suite
Total tests run: 1, Passes: 1, Failures: 0, Skips: 0
================================================
```

Funcionalitat: La funcionalitat **saveJSONFile**(user, puntuation) s'encarrega de carregar l'arxiu JSON (degut a que pot ser hi ha un mateix usuari el qual hagi jugat) a través de l'anterior funció esmentada, loadJSON(), en un objecte JSON. Una vegada carregat, revisa si el nom d'usuari existeix ja, en cas afirmatiu, es revisa la seva puntuació anterior per substituir-la en cas de que aquesta sigui major. En cas de que l'usuari no existeixi, es carrega tant el seu usuari com la seva puntuació. Finalment, es converteix l'objecte JSON una altra vegada en el fitxer anterior: "database.json".

Localització: JSON.java, classe JSON, saveJSONFile().

Test:BoardTest.java, BoardTest, testSaveJSONFile(). Es realitza un test de caixa blanca amb la técnica statement coverage que s'encarrega de testejar la funció saveJSONFile() que inserta 3 usuaris a l'arxiu JSON. Per després comprovar que els tres usuaris han estat creats correctament ho comprovem mitjançant un system.out.printl de la informació continguda a l'arxiu JSON, que coincideix amb els usuaris creats previament.

```
@Test
public void testSaveJSONFile() throws InterruptedException, IOException {

    JSON json = new JSON();
    json.saveJSONFile( username: "User1", puntuation: 15);
    json.saveJSONFile( username: "User2", puntuation: 2);
    json.saveJSONFile( username: "User3", puntuation: 3);
    String json_information ="";
    BufferedReader file = new BufferedReader(new FileReader( fileName: "database.json"));
    json_information = file.readLine();
    file.close();
    System.out.println(json_information);
}
```

```
{"":0,"User2":2,"User1":15,"User3":3}

================================================
Default Suite
Total tests run: 1, Passes: 1, Failures: 0, Skips: 0
================================================
```

Arxiu Apple

Funcionalitat: Constructor de Apple que inicialitza la poma.

Localització: Apple.java, classe Apple, Apple().

Test:BoardTest.java, BoardTest, testCreateApple(). Es realitza un test de caixa blanca amb la técnica statement coverage que comprova que la poma creada pel constructor no sigui NULL.

Funcionalitat: La funció **draw**(Graphics apple) s'encarrega de dibuixar la poma al taulell.

Localització: Apple.java, classe Apple, draw(Graphics apple).

Arxiu Main

Funcionalitat: Constructor Main. Encarregat de generar i inicialitzar tots els paràmetres necessaris per poder jugar. Crea la finestra del joc, el fa visible, inicialitza el nom de l'usuari que està jugant…

Localització: Main.java, classe Main, Main(String name).

Test: Main.java, classe Main, testLoopingEvitar_una_passada(), testLoopingEvitar_dues_passades(), testLoopingEvitar_m_passades(), testLoopingEvitar_n_passades .S'han realitzat 5 diferents tipus de test per a aquest cas degut a que tenim un bucle i realitzem Loop testing. Les funcions de Test son les seguents: testLoopingEvitar(), on evitem el bucle fixant que ja s'han utilitzat tots els intents per a ficar una opció al menú i per tant no accedeix al bucle de cap manera. Com podem fixar-nos a la segona imatge, ens mostra el menú però no ens permet seleccionar cap opció.

```
@Test
public void testLoopingEvitar() throws InterruptedException, FileNotFoundException {
    //Evitar el loop
    Main main = new Main( name: "Test looping");
    main.tries_to_use = 5;
    assertEquals((main.maximum_tries - main.tries_to_use), expected: 0);
    main.showMenu();
}
```

```
--------MENU-------

Select one option:
1. Play
2. Show Ranking

==========================================
Default Suite
Total tests run: 1, Passes: 1, Failures: 0,
```

testLoopingEvitar_una_passada() en aquest test, fixem que els intents que té per seleccionar una opció son 4. De tal manera que només entrará 1 vegada al bucle.

testLoopingEvitar_dues_passades() en aquest test, fixem que els intents que té per seleccionar una opció son 3. De tal manera que només entrará 2 vegades al bucle.

testLoopingEvitar_m_passades() en aquest test, fixem que els intents que té per seleccionar una opció son 2. De tal manera que només entrará 3 vegades al bucle.

testLoopingEvitar_n_passades() en aquest test, fixem que els intents que té per seleccionar una opció és 1.. De tal manera que entrará 4 vegades al bucle.

Funcionalitat: Funció encarregada de mostrar el menú per pantalla i realitzar la crida a la funció Play() o a ShowRanking() segons pertoqui. Disposa d'un bucle en el qual té per defecte 5 oportunitats per triar una opció correcta..

Localització: Main.java, classe Main, ShowMenu(String name).

Funcionalitat: Funció encarregada de printar per pantalla el ranking guardat a la base de dades.

Localització: Main.java, classe Main, ShowRanking(String name).

Test: Mitjançant el test testLoadJSONMOCK() ens assegurem que la funcionalitat showRanking funciona correctament. En primer lloc creem un objecte JSON, al que mitjançant el Mock Object MockJSON, afegim un usuari previament definit i el mostrem a pantalla a través de la funció showRanking().

```
@Test
public void testLoadJSONMOCK() throws InterruptedException, FileNotFoundException {
    JSONObject JsonToTest = new JSONObject();
    JsonToTest = MockJSON.addUserPuntuation();
    Main main = new Main( name: "Testing Mock Object JSON - LOAD");
    main.showRanking(JsonToTest);
}
```

Arxiu MockJSON

Funcionalitat: La funcionalitat de **addUserPuntuation**() és crear un Objecte JSON amb la informació d'un usuari i la seva puntuació i retornar-la. Utilitzant aquest Mock Object, podem simular una base de dades pel moment en que no estava encara implementada.

Localització: MockJSON.java, classe MockJSON, addUserPuntuation().

Funcionalitat: Mitjançant la funcionalitat **getUserPuntuations**() creem un objecte JSON el qual retorna. Intentem simular una càrrega de dades des d'una base de dades a memòria.

Localització: MockJSON.java, classe MockJSON, getUserPuntuations().

Test: Mitjançant el test testSaveJSONMOCK() ens assegurem que podem simular bases de dades per a obtenir aquestes una vegada encara no han estat creades.

```java
@Test
public void testSaveJSONMOCK() throws InterruptedException, FileNotFoundException {
    JSONObject JsonToTest = new JSONObject();
    JsonToTest = MockJSON.getUsersPuntuations();
    String json_information = "{\"Test_Get\":10}";
    JSONObject Actual = new JSONObject(json_information);
    System.out.println(Actual);
    System.out.println(JsonToTest);
}
```

```
{"Test_Get":10}
{"Test_Get":10}
```