



Asignatura: Test Qualitat i Software  
Grado: Ingeniería Informática

## **Práctica 1: Mastermind**

Integrantes del equipo de prácticas:

**Emil Carvajal 1529821 | Lluís Galante 1535722**

15/11/2021

NOTA: <TODAS las capturas de Statement Coverage se encuentran en el apartado "Statement Coverage">

## // HINT

**Funcionalidad:** <Creación de un objeto Hint a partir de un objeto Code y un objeto SecretCode.>

**Localización:** <mastermind/Model/, Hint, Hint() >

**Test:** <test/, HintTest, TestConstructorHint(>.

<Este test garantiza la correcta construcción de la clase Hint inicializando la clase con diversos códigos de usuarios y códigos secretos.

Tipo: Caja blanca , Técnica: Statement Coverage>

**Funcionalidad:** <Ordena las pistas de la clase Hint del tipo : "xo -" (Mayor a menor) >

**Localización:** <mastermind/Model/, Hint, SortHint() >

**Test:** <test/, HintTest, TestSortHint(>.

<Este test ordena diversos tipos de combinaciones y comprueba que las salidas estén correctamente ordenadas en el formato "x o -".

Tipo: Caja blanca , Técnica: Statement Coverage>

## // CODE

**Funcionalidad:** <Creación de un objeto Code a partir de un String>

**Localización:** <mastermind/Model/, Code, Code() >

**Test:** <test/, CodeTest, TestConstructorCode(>.

<Garantiza la correcta construcción de un objeto Code a partir de un String.

Tipo: Caja blanca , Técnica: Statement Coverage>

**Funcionalidad:** <Creación de un objeto Code a partir de un String>

**Localización:** <mastermind/Model/, Code, Code() >

**Test:** <test/, CodeTest, TestConstructorCode(>.

<Garantiza la correcta construcción del objeto a partir de un String.

Tipo: Caja blanca , Técnica: Statement Coverage>

**Funcionalidad:** <Comprueba que dos Codes sean iguales>

**Localización:** <mastermind/Model/, Code, checkCode() >

**Test:** <test/, CodeTest, TestCheckCode(>.

<Se comprueba que si se pasan 2 códigos iguales sean iguales y si se pasan 2 códigos diferentes, que sean diferentes.

Tipo: Caja blanca , Técnica: Statement Coverage>

## // PLAYER

**Funcionalidad:** <Hace que el usuario introduzca un código y lo devuelve>

**Localización:** <mastermind/Model/, Player, IntroduceCode() >

**Test:** <test/, PlayerTest, TestIntroduceCode()>.

<Comprueba que el código introducido devuelve un string si es correcto, si el string introducido por el usuario no es correcto comprueba que la función devuelva null.

Tipo: Caja blanca , Técnica: Statement Coverage>

**Funcionalidad:** <Esta función comprueba que el código introducido por el usuario se encuentre en los límites del juego y sea correcto ( longitud de tamaño cuatro y usando los caracteres disponibles que representan a colores)>

**Localización:** <mastermind/Model/, Player, correct\_code() >

**Test:** <test/, PlayerTest, correct\_codeTest()>.

<Se comprueba que la función devuelva true o false correctamente si los códigos introducidos son realmente correctos o no probando con diversos tipos de posibles códigos.

Tipo: Caja blanca , Técnica: Statement Coverage>

**Funcionalidad:** <Esta función comprueba que el código introducido por el usuario se encuentre en los límites del juego y sea correcto ( longitud de tamaño cuatro y usando los caracteres disponibles que representan a colores)>

**Localización:** <mastermind/Model/, Player, correct\_code() >

**Test:** <test/, PlayerTest, Test\_Particio\_Equivalent\_CSV()>.

<Se ha creado un dataset de posibles códigos y se ha dividido en clases equivalentes en función de valores límite de longitud de los códigos. Para cada clase existen valores válidos (los colores que acepta el juego) e inválidos (cualquier otro tipo de carácter). Este test comprueba que el método correct\_code() sólo devuelve true para la clase “frontera4v” que tienen longitud válida y caracteres válidos. También comprueba que todas las posibles decisiones y condiciones se ejecutan en el método.

Tipo: Caja Negra , Técnicas: Partición equivalente, Valores límite.

Tipo: Caja Blanca , Técnicas: Decision Coverage, Condition Coverage.>

<Dataset>

Nulos	limite2	limite3	frontera4v	frontera4nv	limite5	limite6	limite20
null	YG	YYY	RBYG	ZRBY	OGRBY	OGRBY	OGRBYOGRBYOGRBYOGRBY
null	GR	OGR	GYOP	BGVE	BRY#P	BRYGP	YPVOBYPVOBYPVOBYPVOB
null	RP	OGP	BGVO	OUBG	1G2B!	1G2BY	12345OGRBYOGRBYOGRBY
null	RL	PGT	BYPO	GBYQ	12HRT	12HRT	WQNMZYPVOBYPVOBYPVOB
null	1F	12W	GBYV	LPVO	12345	12345	WQNMZYPVOBYPVOBYP123
null	12	324	RPVO	ORB5	38293	38293	1,23457E+19

```

28 public Boolean correct_code(String code){
29     if( code == null || code.length() != MasterMindGame.CODE_LENGTH ){
30         System.out.println("Codigo introducido inexistente o de largaria inadecuada");
31         return false;
32     }
33     for( int index = 0; index < MasterMindGame.CODE_LENGTH; index++){
34         char characterCode = code.charAt(index);
35         boolean correct_character = false;
36         for(int i = 0; i < MasterMindGame.COLORS.size(); i++){
37             String aux = MasterMindGame.COLORS.get(i);
38             if(characterCode == aux.charAt(0)){
39                 correct_character = true;
40             }
41         }
42         if(!correct_character){
43             //El usuario ha introducido un caracter que no representa ha ningún color.
44             System.out.println("El caracter '" + characterCode + "' no coincide con ninguno de " +
45                 "los colores disponibles. Su codigo introducido queda anulado.");
46             System.out.println("El codigo ha de ser una conbinación de los colores:" + MasterMindGame.COLORS);
47             return false;
48         }
49     }
50     return true;

```

## //SECRET CODE

**Funcionalidad:** <Creación de un objeto Code a partir de un String>

**Localización:** <mastermind/Model/, SecretCode, SecretCode() >

**Test:** <test/, PlayerTest, TestConstructorSecretCode()>.

<Garantiza la correcta construcción del objeto SecretCode a partir de un generador aleatorio.

Tipo: Caja blanca , Técnica: Statement Coverage>

**Funcionalidad:** <Genera el código secreto de colores de longitud 4 de forma aleatoria>

**Localización:** <mastermind/Model/, SecretCode, generateSecretcode() >

**Test:** <test/, PlayerTest, TestGenerateSecretcode()>.

<Comprueba que el la función haya generado un código correcto que se adapta a las normas del juego.

Tipo: Caja blanca , Técnica: Statement Coverage>

**Funcionalidad:** <Simula la obtención de un código secreto utilizando una clase interfaz y un MockObject. Devuelve un código secreto determinado por nosotros >

**Localización:** <mastermind/Model/, SecretCodeInterface, generateSecretcode() >

**Test:** <test/, PlayerTest, TestConstructorSecretCode()>.

<Comprueba la inicialización de Secret Codes sin necesidad de una clase SecretCode, utilizando una SecretCodeInterface. La prueba consiste en determinar un código secreto determinado, llamar a una función del main del juego que utiliza SecretCodes pero adaptada para SecretCodeInterface "introduceCode\_Mock\_SecretCode" y comprobar que el resultado de pistas sea el esperado. También se simula con la vista.

Tipo: Caja Negra , Técnica: MockObejcts, Mock: MockSecretCode>

## // MASTERMIND GAME

**Funcionalidad:** < Constructor de Mastermind. Inicializa el código secreto, el tablero, el jugador, la lista de Intentos, el número de intentos usados y define a las variables de ganador y acabar juego en estado false >

**Localización:** < mastermind/Controlador/, MastermindGameMasterMindGame()>

**Test:** < test/, MastermindGameTest, testConstructor()>.

<El test comprueba que las variables se han inicializado de la manera correcta con los valores esperados.

Tipo: Caja blanca , Técnica: Statement Coverage>

**Funcionalidad:** < Añade el código introducido por el usuario al Tablero, además si el código del usuario coincide con el código secreto restablece las variables de ganador (win) y acabar juego (isOver) a True>

**Localización:** < mastermind/Controlador/, MastermindGame, introduceCode() >

**Test:** < test/, MastermindGameTest, TestIntroduceCode() >.

<Comprueba que los strings de códigos que se pasan a la función IntroduceCode() por parámetro se agreguen a la lista de códigos del Tablero. Además también se encarga de comprobar que realmente se modifica el contenido de las variables de ganador y juego acabado si el string pasado por parámetro coincide con el string equivalente del código secreto.

Tipo: Caja blanca , Técnica: Statement Coverage>

**Funcionalidad:** < Añade el código introducido por el usuario al Tablero, además si el código del usuario coincide con el código secreto restablece las variables de ganador (win) y acabar juego (isOver) a True. Se ha adaptado a la versión original para que funcione con Mocks.>

**Localización:** < mastermind/Controlador/MastermindGame, introduceCode\_Mock()>

**Test:** < test/, MastermindGameTest, Test\_PairwiseCSV()>.

<Este test comprueba que todas las parejas discretas de colores sean probadas en el juego. Para ello hemos tenido que utilizar la interfaz SecredCode\_Interface ya que la clase general genera un código secreto aleatorio y las pistas resultantes serian siempre diferentes.

Tipo: Caja Negra , Técnica: Pairwise, Mock: MockSecretCode>

**Funcionalidad:** <Actualiza el código del jugador en el menú del juego >

**Localización:** <mastermind/Controlador/, update\_code(), >

**Test:** < test/, MastermindGameTest, Test\_Update\_Code()>.

<Comprueba que el código actualizado es el esperado, ya que este ha sido inicializado en un MockObject de la clase Player. También comprueba el correcto funcionamiento del Mock Player.

Tipo: Caja Negra , Técnica: MockObejcts, Mock: MockPlayer1>

**Funcionalidad:** < Es el main del juego normal pero adaptada para poder utilizar las clases PlayerInterface y SecretCode\_Interface. De esta manera podemos simular el juego completo sin conocer el código secreto aleatorio ni las entradas del usuario.>

**Localización:** < mastermind/Controlador/MastermindGame, mainGame\_Mock()>

**Test:** < test/, MastermindGameTest, Test\_Lista\_Codigos()>.

<Este test simula una partida completa del juego mediante un código secreto determinado por un MockObject y una lista de códigos como entradas que se encuentran en el mock de MockPlayer2. Cada vez que se pide al usuario un código, MockPlayer2 devuelve el siguiente código de una lista de códigos y se va actualizando con un contador. También se comprueba que se cumplan todas las decisiones y condiciones.

Tipo: Caja Negra , Mocks: MockSecretCode, MockPlayer2

Tipo: Caja Blanca , Técnicas: Decision Coverage, Condition Coverage>

```
104 public void mainGame_Mock(){
105     this.intentos = 0;
106
107     while(!isOver){
108         //this.isOver=true;
109         if(this.intentos < MAX_OPPORTUNITIES){
110             String code_answer = this.playerInterface.IntroduceCode();
111             this.listaIntentos.add(code_answer);
112
113             System.out.println("l intentos: "+this.listaIntentos.get(0));
114             //Si el codigo introducido no es valido se devuelve null.
115             while ( code_answer == "null"){
116                 System.out.println("is null");
117                 //Volvemos a pedir que introduzca el código si el código
118                 // introducido por el usuario no preseta el formato valido.
119                 code_answer = this.playerInterface.IntroduceCode();
120             }
121             introduceCode_Mock_SecretCode(code_answer);
122             this.intentos++;
123         }
124         else{
125             this.isOver = true;
126             System.out.println("NO MORE OPPORTUNITIES LEFT. END OF THE GAME. ");
127         }
128     }
129 }
```

**Funcionalidad:** <Utilizamos esta función para poder testear un tercer caso de Decision i Condition Coverage>

**Localización:** < mastermind/Controlador/MastermindGame, DecConCoverage3()>

**Test:** < test/, MastermindGameTest, TestDecConCoverage3()>.

< Utiliza unos posibles inputs para poder comprobar que se ejecutan todas las decisiones y condiciones.

Tipo: Caja Blanca , Técnicas: Decision Coverage, Condition Coverage>

```

243
244 public int DecConCoverage3(int input1, int input2){
245     int contador = 0;
246     if (input1 >4 || input1<2 ) { //no -inf a 2
247         if (input2 <0)
248             contador = -2;
249         else
250             contador = 100;
251     }
252     else
253         if (input2 >0)
254             contador = 10;
255         else
256             contador = 80;
257     return contador;
258 }
259 }

```

**Funcionalidad:** <Utilizamos esta función modificada de mainGame() para poder testar mediante un test el path coverage del método mainGame() >

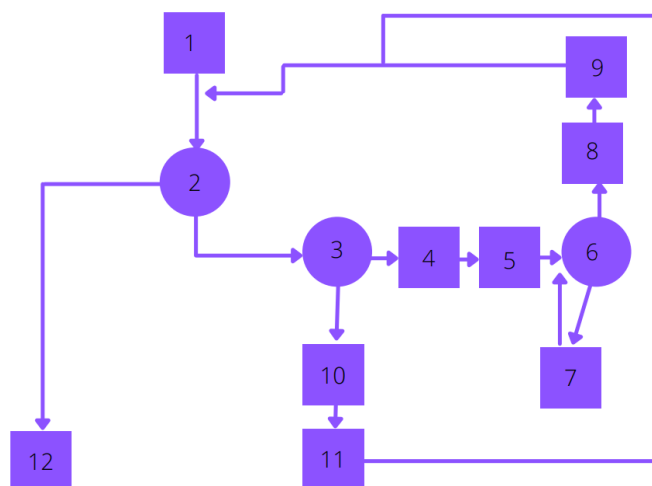
**Localización:** < mastermind/Controlador/, MastermindGame, mainGame\_Paths >

**Test:** < test/, MastermindGameTest, Test\_mainGame\_Paths()>.

<Una vez detectados todos los posibles paths que se pueden generar en la función mainGame(), mediante tests cases definidos ejecutamos todos los paths (mirar diagrama de flujo) para hacer el Path coverage.

Tipo: Caja Blanca , Técnicas: Path Coverage>

\*Diagrama de Flujo de mainGame():



**Funcionalidad:** <Utilizamos esta función simplificada de introduce\_code() para poder realizar un Path Coverage que cubra los posibles casos >

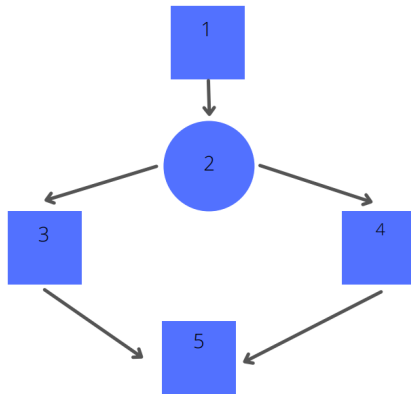
**Localización:** <mastermind/Controlador/, MastermindGame, introduceCode\_Paths >

**Test:** < test/, MastermindGameTest, Test\_introduceCode\_Paths()>.

<Una vez detectados todos los posibles paths que se pueden generar en la función Test\_introduceCode\_Paths() , mediante tests cases definidos ejecutamos todos los paths (mirar diagrama de flujo) para hacer el Path coverage

Tipo: Caja Blanca , Técnicas: Path Coverage>

\*Diagrama de Flujo de introduceCode\_Paths():



#### LOOP TESTING:

**Funcionalidad:** <Usamos esta función para testar el primer caso de Loop simple>

**Localización:** <mastermind/Controlador/ , MastermindGame , TestSimpleLoop1()>

**Test:** < test/, MastermindGameTest, TestSimpleLoop1() >.

<Este test está hecho con la finalidad de hacer loop testing simple. Como debida a la manera que está implementado nuestro código no hemos podido encontrar loops para hacer loop testing, hemos recreado en Mastermind unos bucles extra para poder realizar sus correspondientes loops testings.

Tipo: Caja Blanca , Técnicas: Loop Testing >

**Funcionalidad:** <Usamos esta función para testar el segundo caso de Loop simple >

**Localización:** <mastermind/Controlador/ , MastermindGame , TestSimpleLoop2() >

**Test:** < test/, MastermindGameTest, TestSimpleLoop2() >.

<Este test está hecho con la finalidad de hacer loop testing simple. Como debida a la manera que está implementado nuestro código no hemos podido encontrar loops para hacer loop testing, hemos recreado en Mastermind unos bucles extra para poder realizar sus correspondientes loops testing.

Tipo: Caja Blanca , Técnicas: Loop Testing >



**Funcionalidad:** <Usamos esta función para testar el primer caso de Loop Anidado >

**Localización:** <mastermind/Controlador/ , MastermindGame , TestAniuatLoop1() >

**Test:** < test/, MastermindGameTest, TestAniuatLoop1() >.

< Este test está hecho con la finalidad de hacer loop testing anidado. Como debida a la manera que está implementado nuestro código no hemos podido encontrar loops para hacer loop testing, hemos recreado en Mastermind unos bucles extra para poder realizar sus correspondientes loops testing.

Tipo: Caja Blanca , Técnicas: Loop Testing >

**Funcionalidad:** < Usamos esta función para testar el segundo caso de Loop Anidado>

**Localización:** < mastermind/Controlador/, MastermindGame , TestAniuatLoop2() >

**Test:** < test/, MastermindGameTest, TestAniuatLoop2() >.

< Este test está hecho con la finalidad de hacer loop testing anidado. Como debida a la manera que está implementado nuestro código no hemos podido encontrar loops para hacer loop testing, hemos recreado en Mastermind unos bucles extra para poder realizar sus correspondientes loops testing.

Tipo: Caja Blanca , Técnicas: Loop Testing >

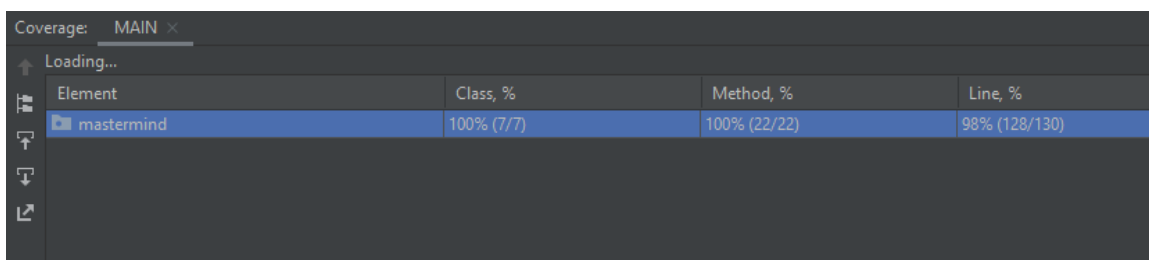
### Statement coverage:

Capturas del statement coverage en las que se aprecia como todas las sentencias de nuestro código (una vez excluidas las funciones que usábamos únicamente para los tests) se ejecutan como mínimo una vez .

Cabe destacar que el porcentaje de líneas ejecutadas es del 98% ya que para estos resultados de las capturas hemos simulado un juego real y también hemos quitando los Mock Objects y tests de forma temporal solo para realizar el coverage.

Las dos líneas que no se ejecutan son las marcadas en rojo en la última captura, estas no se ejecutan ya que el jugador de la ejecución ganó sin llegar al límite de intentos sin haber acertado, de ahí que las últimas dos líneas para el caso que no hubiera ganado y se hubiera quedado sin intentos disponibles y no se ejecuten.

De todas formas al haber hecho TDD hemos asegurado el Statement Coverage de todo el código.



The screenshot shows a 'Coverage: MAIN' window with a table of coverage data. The table has four columns: 'Element', 'Class, %', 'Method, %', and 'Line, %'. The 'mastermind' class is highlighted with a blue background and shows 100% coverage for both classes and methods, and 98% coverage for lines (128 out of 130). The window also includes a 'Loading...' status bar and a sidebar with icons for navigating through the project structure.

Element	Class, %	Method, %	Line, %
mastermind	100% (7/7)	100% (22/22)	98% (128/130)

Coverage: MAIN x

100% classes, 98% lines covered in package 'mastermind'

Element	Class, %	Method, %	Line, %
Controlador	100% (1/1)	100% (4/4)	94% (34/36)
Model	100% (4/4)	100% (12/12)	100% (61/61)
Vista	100% (1/1)	100% (5/5)	100% (31/31)
MAIN	100% (1/1)	100% (1/1)	100% (2/2)

Coverage: MAIN x

100% classes, 94% lines covered in package 'mastermind.Controlador'

Element	Class, %	Method, %	Line, %
MasterMindGame	100% (1/1)	100% (4/4)	94% (34/36)

Coverage: MAIN x

100% classes, 100% lines covered in package 'mastermind.Model'

Element	Class, %	Method, %	Line, %
Code	100% (1/1)	100% (3/3)	100% (4/4)
Hint	100% (1/1)	100% (3/3)	100% (21/21)
Player	100% (1/1)	100% (3/3)	100% (26/26)
SecretCode	100% (1/1)	100% (3/3)	100% (10/10)

Coverage: MAIN x

100% classes, 100% lines covered in package 'mastermind.Vista'

Element	Class, %	Method, %	Line, %
Board	100% (1/1)	100% (5/5)	100% (31/31)

```

31
32 public void mainGame(){
33     while(!isOver){
34         if(this.intentos < MAX_OPPORTUNITIES){
35             String code_answer = this.player.IntroduceCode();
36             this.listaIntentos.add(code_answer);
37             //Si el código introducido no es válido se devuelve null.
38             while (code_answer == null){
39                 //Volvemos a pedir que introduzca el código si el código introducido por el usuario no preseta el formato válido.
40                 code_answer = this.player.IntroduceCode();
41             }
42             introduceCode(code_answer);
43             this.intentos++;
44         }
45         else{
46             this.isOver = true;
47             System.out.println("NO MORE OPPORTUNITIES LEFT. END OF THE GAME. ");
48         }
49     }
50 }

```

Líneas no ejecutadas porque el jugador no ha acertado en este test.

## MOCKS

**MockHint:** Simula clase Hint.

**MockPlayer1:** Simula clase Player.

**MockPlayer2:** Simula clase Player pero utiliza una lista determinada de códigos.

**MockSecretCode:** Simula clase SecretCode.