

UNO Digital

L'UNO és un joc de cartes en què es competeix contra altres jugadors amb l'objectiu de ser el primer a quedar-se sense cartes. Es juga amb el color i el número de les cartes. Aquest joc s'ha implementat amb Java a través dels mètodes i test següents:

INDEX

MODEL:	2
Carta	2
Jugador	5
Mazo	7
Partida	10
CONTROLLER:	15
VIEW:	19
LOOP TESTING:	22
PAIRWISE TESTING:	23
PATH COVERAGE:	24

MODEL:

Carta

public Carta(String color, String valor)

Funcionalitat: El constructor de la classe. Verifica que el color i el valor de la carta sigui vàlid i comprova tots els possibles casos que es poden donar a l'hora de jugar una carta.

Localització: main/Model/Carta.java, public class Carta.

Test: test/ModelTest/CartaTest.java, public class CartaTest.

Tots els tests fins inclòs el setUp utilitzen el constructor per inicialitzar les proves.

public String getColor()

Funcionalitat: Aquesta funció decideix tornar el valor del *private String color*, el qual pot prendre els diferents valors: "r" = vermell, "b" = blau, "g" = verd, "i" = groc, null = comodí.

Localització: main/Model/Carta.java, public class Carta.

Test: test/ModelTest/CartaTest.java, public class CartaTest.

- **void testCartaColoresValidos()** : Caixa negra i particions equivalents. Caixa blanca i condition coverage. Es divideix en colors permesos i no permesos.
- **void testCartaColorInvalido()** : Caixa negra i particions equivalents. Caixa blanca i condition coverage. Valida restriccions específiques amb assertThrows.
- **void testCartaAccionEspecialValida()** : Caixa negra, particions equivalents i anàlisi de valors limit.
- **void testCartaAccionEspecialInvalida()** : Caixa negra i particions equivalents. Avalua restriccions addicionals per a cartes especials.

public String getValor()

Funcionalitat: Aquesta funció decideix tornar el valor del *private String valor*, el qual pot prendre els diferents valors: "0" a "9" per números, "skip", "reverse", "+2", "wild", "+4" per especials.

Localització: main/Model/Carta.java, public class Carta.

Test: test/ModelTest/CartaTest.java, public class CartaTest.

- **void testCartaNumeroEnRango()** : Caixa negra i particions equivalents. Es valida exclusivament el conjunt permès de valors numèrics.
- **void testCartaNumeroInvalidos()** : Caixa negra i particions equivalents. Caixa blanca, decision coverage i condition coverage. S'inclou un anàlisi de valors límits invàlids i restriccions addicionals com no permetre cartes numèriques sense color.
- **void testCartaAccionEspecialValida()** : Caixa negra, particions equivalents i anàlisi de valors limit.
- **void testCartaAccionEspecialInvalida()** : Caixa negra i particions equivalents. Avalua restriccions addicionals per a cartes especials.

També cal ressaltar que tests como *testCartaNumeroInvalidos* y *testCartaColorInvalido* provenen casos límits i excepcions, verificant certes rutes o condicions del constructor de la classe Carta siguin executades. S'apliquen tècniques com *condition coverage*.

public boolean esCompatible(Carta otraCarta)

Funcionalitat: Aquest mètode verifica si la carta actual es compatible amb la carta que es passa per paràmetre.

Localització: main/Model/Carta.java, public class Carta.

Test: test/ModelTest/CartaTest.java, public class CartaTest.

- **void testCartaCompatibleMismoColor()** : Caixa negra i particions equivalents. Caixa blanca, decision coverage i statement coverage. Valida la compatibilitat basant-se en una característica compartida, en aquest cas, el color.
- **void testCartaCompatibleMismoValor()** : Caixa negra i particions equivalents. Valida la compatibilitat basant-se en una característica compartida, en aquest cas, el valor.
- **void testCompatibilidadCartasEspecialesDiferenteColor()** : Caixa negra i particions equivalents. Valida regles específiques per cartes especials.
- **void testCartaComodinSiempreCompatible()** : Caixa blanca i statement coverage. S'assegura que compleix la seva funcionalitat.
- **void testCartaIncompatible()** : Caixa negra i particions equivalents. Caixa blanca i decision coverage. Verifica que no es compleix ninguna condició de compatibilitat.
- **void testCompatibilidadCartasEspecialesMismoColor()** : Caixa negra i particions equivalents. Es verifica la compatibilitat entre dos cartes especials de colors iguals.
- **void testCompatibilidadCartasDiferentesEspecialesColor()** : Caixa negra i particions equivalents. Es verifica la incompatibilitat entre cartes especials de diferents colors.
- **void testCompatibilidadOtraCartaEsNull()** : Caixa negra i particions equivalents. Es valida que el mètode faci correctament la situació quan la carta es null.
- **void testEsCompatiblePairwise()** : Caixa negra i pairwise testing.














També cal ressaltar que aquests tests exploren el comportament intern de la funció, verificant rutes específiques com: Mateixa condició de color, mateix valor, cas del comodí o cas d'incompatibilitat. S'apliquen tècniques com *Path coverage*.

Nota: En la implementació d'aquest mètode es fan servir mètodes privats booleans als quals no hem assignat proves perquè són un complement per no tenir una línia de codi tan extensa, aquests serien:

- *private boolean isColorValido(String color)*
- *private boolean isValorValido(String valor)*
- *public boolean isValorEspecial(String valor)*

Coverage de Carta:

Carta

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
● Carta(String, String)		100 %		100 %	0	7	0	6	0	1
● esCompatible(Carta)		100 %		100 %	0	6	0	4	0	1
● isValorEspecial(String)		100 %		100 %	0	6	0	2	0	1
● isColorValido(String)		100 %		100 %	0	5	0	1	0	1
● isValorValido(String)		100 %		100 %	0	4	0	3	0	1
● getColor()		100 %		n/a	0	1	0	1	0	1
● getValor()		100 %		n/a	0	1	0	1	0	1
● static {...}		100 %		n/a	0	1	0	1	0	1
Total	0 of 142	100 %	0 of 46	100 %	0	31	0	19	0	8

Nota: Missed Branches es una fusió de decision i condition coverage. No s'ha dit abans en tots els tests, perquè no estàvem segurs.

Jugador

public Jugador(String nom)

Funcionalitat: El constructor de la classe. Verifica que el constructor no agafi valors null pel nombre i inicialitza un nou jugador amb un nombre específic i una mà de cartes vuida.

Localització: main/Model/Jugador.java, public class Jugador.

Test: test/ModelTest/JugadorTest.java, public class JugadorTest.

- **void setUp()**
- **void testConstructor()** : Caixa negra i particions equivalents. Es valida que la classe Jugador manegi correctament els valors vàlids i llenci excepcions per a noms nulls.

public void robarCarta(Mazo mazo)

Funcionalitat: Permet al jugador robar una carta del mall.

Localització: main/Model/Jugador.java, public class Jugador.

Test: test/ModelTest/JugadorTest.java, public class JugadorTest.

Tots els tests (a excepció del setUp i testConstructor) utilitzen aquest mètode per agafar cartes noves. Però hi ha dos que assegurin el seu funcionament:

- **void testRobarCarta()**
- **void testRobarCartaError()**

Ambdós són considerats caixa negra, particions equivalents i valors límit.

public boolean jugarCarta(Carta carta, Mazo mazo)

Funcionalitat: Permet al jugador jugar una carta de la seva mà al mall de cartes.

Localització: main/Model/Jugador.java, public class Jugador.

Test: test/ModelTest/JugadorTest.java, public class JugadorTest.

- **void testJugarCartaValida()** : Caixa negra i particions equivalents. Caixa blanca i decision coverage. Es valida que un jugador pugui jugar una carta valida i compatible amb la del mall.
- **void testJugarCartaInvalida()** : Caixa negra i particions equivalents. Caixa blanca i decision coverage. Es valida que un jugador no pugui jugar una carta que no es compatible.
- **void testJugarCartaError()** : Caixa negra i particions equivalents. Caixa blanca i condition coverage. Es proven diversos casos d'error, com passar una carta null, un mall null, etc

public String getNombre()

Funcionalitat: Obtenir el nom del jugador.

Localització: main/Model/Jugador.java, public class Jugador.

Test: test/ModelTest/JugadorTest.java, public class JugadorTest.

No hi ha tests específics, però es comprova en el *testConstructor()*.

public List<Carta> getMano()

Funcionalitat: Obtenir el mall del jugador.


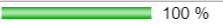







Localització: main/Model/Jugador.java, public class Jugador.

Test: test/ModelTest/JugadorTest.java, public class JugadorTest.

No hi ha tests específics, però s'utilitza en tots els tests.

Coverage de Jugador:

Jugador

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods
● jugarCarta(Carta, Mazo)		100 %		100 %	0 5	0 7	0 1
● robarCarta(Mazo)		100 %		100 %	0 3	0 5	0 1
● Jugador(String)		100 %		100 %	0 2	0 5	0 1
● getMano()		100 %	n/a	n/a	0 1	0 1	0 1
● getNombre()		100 %	n/a	n/a	0 1	0 1	0 1
● static {...}		100 %	n/a	n/a	0 1	0 1	0 1
Total	0 of 92	100 %	0 of 14	100 %	0 13	0 20	0 6

Nota: Missed Branches es una fusió de decision i condition coverage. No s'ha dit abans en tots els tests, perquè no estàvem segurs.

Mazo

public Mazo()

Funcionalitat: El constructor de la classe. Inicialitza el mall amb totes les cartes necessàries.

Localització: main/Model/Mazo.java, public class Mazo.

Test: test/ModelTest/MazoTest.java, public class MazoTest.

- **void setUp()**

Nota: El constructor mazo utilitza una funció complementaria per poder inicialitzar el mall correctament, aquesta es:

Private void inicializar()

Funcionalitat: Inicialitza el mall amb les cartes estandard del joc: cartes numèriques del 0-9 per cada color, cartes especials skip, reverse i +2, i comodins, canvi de color i +4.

Localització: main/Model/Mazo.java, public class Mazo.

Test: test/ModelTest/MazoTest.java, public class MazoTest.

Al ser un mètode privat i complementari del constructor no té ningun tests específic, però s'ha creat un de nou per comprovar que la inicialització es correcte anomenat:

- **void testInicializacionCorrectaDelMazo()** : Caixa negra i particions equivalents. Caixa blanca i statement coverage. Es valida que el mall contingui 108 cartes amb les freqüències esperades per cada tipus de carta.

public void robarCarta()

Funcionalitat: Permet al jugador robar una carta del mall, respectant les probabilitats originals del mall.

Localització: main/Model/Jugador.java, public class Jugador.

Test: test/ModelTest/JugadorTest.java, public class JugadorTest.

- **void testRobarCartaNoDebeSerNull()** : Caixa blanca i decision coverage. Es valida que la funció no retorni null.
- **void testRobarCartaDebeTenerColorCorrecto()** : Caixa negra i particions equivalents. Caixa blanca i condition coverage. Es valida que les cartes tinguin un color vàlid, excepte les cartes comodín.
- **void testRobarCartaDebeTenerValorCorrecto()** : Caixa negra i particions equivalents. Es valida que la cartes sigui un numero, especial o comodí.
- **void testProbabilidadDeCartas()** : Caixa negra i valors límit. Verifica la distribució de probabilitats en el mall mitjançant repetides iteracions.
- **void testVariedadDeCartasRobadas()** : Caixa negra i particions equivalents. Es valida que al menys 10 tipus de cartes robades siguin diferents en 1000 extraccions.

public List<Carta> getCartas()

Funcionalitat: Obté la llista de cartes del mall.

Localització: main/Model/Mazo.java, public class Mazo.

Test: test/ModelTest/MazoTest.java, public class MazoTest.

- **void testInicializacionCorrectaDelMazo()**

public Carta obtenerUltimaCartaJugada()

Funcionalitat: Obté l'última carta que va ser jugada.

Localització: main/Model/Jugador.java, public class Jugador.

Test: test/ModelTest/JugadorTest.java, public class JugadorTest.

- **Void testActualizarUltimaCartaJugada_PrimerCarta()**
- **Void testActualizarUltimaCartaJugada_CartaCompatible()**
- **Void testActualizarUltimaCartaJugada_CartaIncompatible()**
- **Void testActualizarUltimaCartaJugada_ComodinCompatible()**
- **Void testActualizarUltimaCartaJugada_ComodinIncompatible()**
- **Void testEstablecerComodinColorYCondicionarProximaCarta()**
- **Void testCondicionDeColorComodinFallaConColorIncorrecto()**
- **void testRestablecerColorComodinConCartaNormal()**

Aquestes probes utilitzen: Caixa negra i particions equivalents. Caixa blanca, statement coverage, decision coverage i condition coverage. Es verifica que última carta jugada s'actualitzi correctament dependent de les condicions de la carta i les regles del joc.

A més, d'aquesta funció també hi es troba una funció complementaria per poder prosseguir amb el joc i guardar els valors anomenada:

public boolean actualizarUltimaCartaJugada(Carta carta)

Funcionalitat: Si la carta no es null, s'actualitza l'última carta jugada si es compatible amb l'actual, sense tenir en compte els comodins.

Localització: main/Model/Jugador.java, public class Jugador.

Test: test/ModelTest/JugadorTest.java, public class JugadorTest.

Mateixos tests que obtenerUltimaCartaJugada()

Nota: Aquestes funció realitzen tants tests, tots iguals i amb una diferència en què paràmetre de Carta s'avalua que s'ha vist preferible una explicació conjunta que abraça tot allò relacionat amb els tests.

public Carta obtenerComodinColor()

Funcionalitat: Obté el color d'un comodí jugat, si es correspon.

Localització: main/Model/Jugador.java, public class Jugador.

Test: test/ModelTest/JugadorTest.java, public class JugadorTest.

- **Void testActualizarUltimaCartaJugada_ComodinCompatible() :** Caixa negra i particions equivalents. Es verifica que última carta jugada s'actualitzi correctament dependent de les condicions d'un canvi de color o +4.

public void establecerComodinColor(String colorElegido)

Funcionalitat: Mentre el color sigui vàlid, s'estableix el color d'un comodí jugat per condicional la pròxima carta jugada.

Localització: main/Model/Jugador.java, public class Jugador.












Test: test/ModelTest/JugadorTest.java, public class JugadorTest.

- **void testActualizarUltimaCartaJugada_ComodinCompatible()**
- **void testActualizarUltimaCartaJugada_ComodinIncompatible()**
- **void testEstablecerComodinColorYCondicionarProximaCarta()**

- **void testCondicionDeColorComodinFallaConColorIncorrecto()**
- **void testRestablecerColorComodinConCartaNormal()**

Coverage de Mazo:

Mazo

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
● inicializar()		100 %		100 %	0	5	0	14	0	1
● actualizarUltimaCartaJugada(Carta)		100 %		100 %	0	8	0	10	0	1
● establecerComodinColor(String)		100 %		100 %	0	6	0	5	0	1
● Mazo()		100 %		n/a	0	1	0	6	0	1
● robarCarta()		100 %		n/a	0	1	0	3	0	1
● getCartas()		100 %		n/a	0	1	0	1	0	1
● obtenerUltimaCartaJugada()		100 %		n/a	0	1	0	1	0	1
● obtenerComodinColor()		100 %		n/a	0	1	0	1	0	1
● static {...}		100 %		n/a	0	1	0	1	0	1
Total	0 of 252	100 %	0 of 32	100 %	0	25	0	42	0	9

Nota: Missed Branches es una fusió de decision i condition coverage. No s'ha dit abans en tots els tests, perquè no estàvem segurs.

Nota: A diferencia de les classes anteriors, aquestes s'han tingut que posar uns tests complementaris per poder fer coverage:

- **void testActualizarUltimaCartaJugadaConNull()**
- **void testEstablecerComodinColorNull()**
- **void testEstablecerComodinColorIncorrecto()**

Partida

public Partida()

Funcionalitat: El constructor de la classe. Inicialitza la partida amb una llista buida de jugadors, un primer jugador, la direcció del turn i la inicialització correcta del mall.

Localització: main/Model/Partida.java, public class Partida.

Test: test/ModelTest/PartidaTest.java, public class PartidaTest.

- **void setUp()**
- **void testEsFinPartida()**

public void setMazoMock(Mazo mazo)

Funcionalitat: Si el mall no es null, estableix un mock per poder jugar la partida.

Localització: main/Model/Partida.java, public class Partida.

Test: test/ModelTest/PartidaTest.java, public class PartidaTest.

- **void setUp()**
- **void testEsFinPartida()**
- **void testSetMazoMockNull()**

public void setJugadoresMock(List<Jugador> jugadores)

Funcionalitat: Si la llista no es null ni buida, estableix una llista de jugadors mock per la partida.

Localització: main/Model/Partida.java, public class Partida.

Test: test/ModelTest/PartidaTest.java, public class PartidaTest.

- **void setUp()**
- **void testJugarCarta_CartaCompatible()**
- **void testJugarCarta_ComodinConColorElegido()**
- **void testSetJugadoresMockNullYEmpty()**

Per a iniciar partida s'han fet quatre funcions diferents que agafen diferents valors d'entrada:

Funcionalitat:

- **public void iniciarPartida(List<String> nombres, int numCartas) :** Inicia la partida amb un nombre específic de jugadors i cartes.
- **public void iniciarPartida(int num_jugadores) :** Si el mall no es null, estableix un mock per poder jugar la partida.
- **public void iniciarPartida(List<String> nombres) :** Inicia la partida amb uns jugadors específics i 7 cartes per cada jugador.
- **public void iniciarPartida(int num_jugadores, int numCartas) :** Inicia la partida amb un nombre personalitzat de jugadors i cartes.

Localització: main/Model/Partida.java, public class Partida.

Test: test/ModelTest/PartidaTest.java, public class PartidaTest.

- **void setUp()**
- **void testIniciarPartida_NumeroDeJugadores() :** Caixa negra, particions equivalents, valors frontera. Caixa blanca, decision coverage i condition coverage. Es verifica que la partida s'inicialitza correctament segons el nombre de jugadors i es comença amb 7 cartes.
- **void testIniciarPartida_ConNombresYCartas() :** Caixa negra, particions equivalents i valors frontera. Caixa blanca i Loop Testing Niat.
- **void testIniciarPartida_ConNombres() :** Caixa negra, particions equivalents i valors frontera. Caixa blanca i Loop Testing Simple.

[Per més informació llegir els comentaris del codi]

- **void testIniciarPartidaConNumeroDeJugadoresYCartas()** : Caixa negra, particions equivalents i valors límits.
- **void testEsFinPartida()**

public boolean jugarCarta(Carta carta)

Funcionalitat: Permet al jugador actual jugar una carta sense tindre que escollir un color per a un comodí.

Localització: main/Model/Partida.java, public class Partida.

Test: test/ModelTest/PartidaTest.java, public class PartidaTest.

- **void testCicloDeTurnosEnSentidoHorario()** : Caixa negra, Mock, restriccions i particions equivalents. Caixa blanca i decision coverage. Es valida que els torns siguin correctes en sentit horari.
- **void testJugarCarta_CartaCompatible()** : Caixa negra, Mock i particions equivalents. Es comprova que la carta compatible es pot jugar i actualitzar correctament.
- **void testJugarCarta_CartaIncompatible()** : Caixa negra, Mock, restriccions i particions equivalents. Es comprova que la carta no es pot jugar.
Nota: Aquests dos últims testos avaluen el codi intern i realitzen statement coverage.
- **void testAplicarCartaEspecial_CambioSentido()**
- **void testCicloDeTurnosEnSentidoAntihorario** : Caixa negra, Mock i particions equivalents. Caixa blanca i decision coverage. Es valida que els torns siguin correctes en sentit anti-horari.
- **void testAplicarCartaEspecial_SkipTurno** : Caixa negra i Mock. Es valida que la carta skip salta correctament al següent jugador.
- **void testAplicarCartaEspecial_RobarDosCartas** : Caixa negra i Mock. Es valida que la carta +2 fa robar dos cartes al següent jugador.

A més, existeix una altra crida a la funció *jugarCarta* però quan es té que escollir color per un comodí. Aquesta s'anomena:

public boolean jugarCarta(Carta carta, String colorElegido)

Funcionalitat: Permet al jugador actual jugar una carta amb la condició d'escollir un color per a un comodí.

Localització: main/Model/Partida.java, public class Partida.

Test: test/ModelTest/PartidaTest.java, public class PartidaTest.

- **void testJugarCarta_ComodinConColorElegido()** : Caixa negra i Mock. Es valida que un comodí permeti canviar el color i actualitzar-se correctament.
- **void testJugarCarta_ComodinConColorElegidoNull()** : Caixa negra i restriccions. Garanteix que no es pugui establir un color null al jugar un comodí.
- **void testJugarCarta_CartaNull()** : Caixa negra i restriccions. Es valida que no es pugui jugar una carta null.
- **void testAplicarCartaEspecial_RobarCuatroCartas()** : Caixa negra, Mock i restriccions. Es valida que la carta +4 fa robar dos cartes al següent jugador.
- **void testEsFinPartida()**

Aquesta funció utilitza una altra funció complementaria per poder efectuar una acció depenent del tipus de carta especial:

Private void aplicarCartaEspecial()

Funcionalitat: Aplica l'efecte d'una carta especial, incloent l'elecció de color per a comodí.

Localització: main/Model/Mazo.java, public class Mazo.

Test: test/ModelTest/PartidaTest.java, public class PartidaTest.

Al ser un mètode privat i complementari del constructor no té ningun tests específic.

Per concluir amb la funció de jugarCarta, hi ha un mètode a la classe anomenat **robarCartaJugadorActual()** que s'aplica a tots els tests que comproven el funcionament de *jugarCarta()*.

Private void cambiarTurno()

Funcionalitat: Canvia el turn al següent jugador depenent del sentit.

Localització: main/Model/Partida.java, public class Partida.

No té tests, sinó que es una funció complementaria als següents mètodes de la classe:

- **jugarCarta(Carta carta, String colorElegido)**
- **aplicarCartaEspecial(Carta carta, String colorElegido)**

public boolean getSentidoHorario()

Funcionalitat: Retorna la direcció del torn (horari o antihorari)

Localització: main/Model/Partida.java, public class Partida.

Test: test/ModelTest/PartidaTest.java, public class PartidaTest.

- **void testAplicarCartaEspecial_CambioSentido()** : Caixa blanca, Mock i statement coverage. Es valida que la carta reverse canviï el sentit del joc per el cas horari i antihorari.

public boolean esFinPartida()

Funcionalitat: Verifica si la partida ha acabat.

Localització: main/Model/Partida.java, public class Partida.

Test: test/ModelTest/PartidaTest.java, public class PartidaTest.

- **void testEsFinPartida()** : Caixa negra i restriccions. S'assegura que el mètode només es pot utilitzar mentre hi hagi jugadors en la partida i simula una partida de joc.

public List<Jugador> getJugadores()

Funcionalitat: Retorna la llista de jugadors en la partida.

Localització: main/Model/Partida.java, public class Partida.

Test: test/ModelTest/PartidaTest.java, public class PartidaTest.

- **void testIniciarPartida_NumeroDeJugadores**
- **void testAplicarCartaEspecial_RobarDosCartas**
- **void testAplicarCartaEspecial_RobarCuatroCartas**

public Jugador getJugadorActual()

Funcionalitat: Retorna el jugador que té el torn.

Localització: main/Model/Partida.java, public class Partida.

Test: test/ModelTest/PartidaTest.java, public class PartidaTest.

- **void testJugarCarta_CartaCompatible**

public int getNumeroJugadorActual()

Funcionalitat: Retorna el nombre de jugadors en la partida.

Localització: main/Model/Partida.java, public class Partida.

Test: test/ModelTest/PartidaTest.java, public class PartidaTest.

- **void testCicloDeTurnosEnSentidoHorario**
- **void testCicloDeTurnosEnSentidoAntihorario**
- **void testAplicarCartaEspecial_SkipTurno**
- **void testAplicarCartaEspecial_RobarDosCartas**
- **void testAplicarCartaEspecial_RobarCuatroCartas**

public Carta obtenerUltimaCartaJugada()

Funcionalitat: Obtenir l'última carta jugada.

Localització: main/Model/Partida.java, public class Partida.

Test: test/ModelTest/PartidaTest.java, public class PartidaTest.

- **void testJugarCarta_CartaCompatible**
- **void testJugarCarta_CartaIncompatible**
- **void testJugarCarta_ComodinConColorElegido**

public String obtenerComodinColor()

Funcionalitat: Obtenir el color de l'últim comodí jugat.

Localització: main/Model/Partida.java, public class Partida.

Test: test/ModelTest/PartidaTest.java, public class PartidaTest.

- **void testJugarCarta_ComodinConColorElegido**

Coverage de Partida:

Partida

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
● iniciarPartida(List, int)		100 %		100 %	0	11	0	14	0	1
● aplicarCartaEspecial(Carta, String)		100 %		94 %	1	12	0	21	0	1
● iniciarPartida(int)		100 %		100 %	0	4	0	6	0	1
● iniciarPartida(int, int)		100 %		100 %	0	4	0	6	0	1
● jugarCarta(Carta, String)		100 %		100 %	0	4	0	7	0	1
● esFinPartida()		100 %		100 %	0	4	0	6	0	1
● cambiarTurno()		100 %		100 %	0	2	0	4	0	1
● Partida()		100 %		n/a	0	1	0	6	0	1
● setJugadoresMock(List)		100 %		100 %	0	3	0	3	0	1
● setMazoMock(Mazo)		100 %		100 %	0	2	0	3	0	1
● getJugadorActual()		100 %		n/a	0	1	0	1	0	1
● robarCartaJugadorActual()		100 %		n/a	0	1	0	2	0	1
● iniciarPartida(List)		100 %		n/a	0	1	0	2	0	1
● jugarCarta(Carta)		100 %		n/a	0	1	0	1	0	1
● obtenerUltimaCartaJugada()		100 %		n/a	0	1	0	1	0	1
● obtenerComodinColor()		100 %		n/a	0	1	0	1	0	1
● getSentidoHorario()		100 %		n/a	0	1	0	1	0	1
● getJugadores()		100 %		n/a	0	1	0	1	0	1
● getNumeroJugadorActual()		100 %		n/a	0	1	0	1	0	1
● static {...}		100 %		n/a	0	1	0	1	0	1
Total	0 of 417	100 %	1 of 70	98 %	1	57	0	88	0	20

Nota: Missed Branches es una fusió de decision i condition coverage. No s'ha dit abans en tots els tests, perquè no estàvem segurs.

Nota: Com es pot veure a la imatge, ens apareix que aplicarCartaEspecial no recorre totes les possibilitats. Si mirem la imatge de més a baix, comprovem que es tracta d'un switch. El problema recau amb el per defecte (no es per el break, coses de l'editor), que mai s'activa. En el nostre codi ens assegurem que abans d'enviar ninguna carta, aquesta es comprovi que sigui especial. Al no enviar mai una carta no especial, el per defecte no s'activa.

```
switch (carta.getValor()) {  
    case "+4":  
        mazo.establecerComodinColor(colorElegido);  
        cambiarTurno();  
        for (int i = 0; i < 4; i++) {  
            robarCartaJugadorActual();  
        }  
        break;  
    case "wild":  
        mazo.establecerComodinColor(colorElegido);  
        cambiarTurno();  
        break;  
    case "skip":  
        cambiarTurno();  
        cambiarTurno();  
        break;  
    case "reverse":  
        sentidoHorario = !sentidoHorario;  
        cambiarTurno();  
        break;  
    case "+2":  
        cambiarTurno();  
        for (int i = 0; i < 2; i++) {  
            robarCartaJugadorActual();  
        }  
        break;  
}
```

CONTROLLER:

public JuegoController()

Funcionalitat: El constructor de la classe.

Localització: main/Controller/JuegoController.java, public class JuegoController.

Test: test/ControllerTest/ JuegoController.java, public class JuegoControllerTest.

- **void SetUp()**

public void setPartida(Partida partida)

Funcionalitat: Estableix una partida per al joc.

Localització: main/Controller/JuegoController.java, public class JuegoController.

Test: test/ControllerTest/ JuegoController.java, public class JuegoControllerTest.

- **void SetUp()**
- **void testGuardarPartidaYCargarPartida_Exito()**
- **void testGuardarPartida_ErrorGuardado()**

public void setInterfaz(InterfazJuego interfaz)

Funcionalitat: Estableix una interfície per al joc.

Localització: main/Controller/JuegoController.java, public class JuegoController.

Test: test/ControllerTest/ JuegoController.java, public class JuegoControllerTest.

- **void SetUp()**

public void mostrarMenu()

Funcionalitat: Mostra el menú principal al usuari i maneja la selecció d'opcions.

Localització: main/Controller/JuegoController.java, public class JuegoController.

Test: test/ControllerTest/ JuegoController.java, public class JuegoControllerTest.

- **void testMostrarMenu_Opcion3**
- **void testMostrarMenu_Opcion2**
- **void testMostrarMenu_OpcinoNoValido**
- **void testMostrarMenu_Opcion1_Opcion3**
- **void testMostrarMenu_Opcion1_Opcion2**
- **void testMostrarMenu_Opcion1_OpcionNoValido**
- **void testMostrarMenu_Opcion1_Opcion1_Salir**
- **void testMostrarMenu_Opcion1_Opcion1_Introducir0y5**
- **void testMostrarMenu_Opcion1_Opcion1_IntroducirX**
- **void testMostrarMenu_Opcion1_Opcion1_2Jugadores_SalirSinGuardar**
- **void testMostrarMenu_Opcion1_Opcion1_2Jugadores_SalirGuardando**

Tots aquests tests implementen: Caixa negra, particions equivalent i Mockito.

El menú consta d'una opció 1 (mode Offline), opció 2 (mode Online, que no funciona) i opció 3 (sortir). La primera opció està definida com un private en la mateixa classe i es complementaria d'aquesta funció:

private void mostrarMenuOffline()

Funcionalitat: Mostra el menú del modo Offline i maneja la selecció d'opcions.

Localització: main/Controller/JuegoController.java, public class JuegoController.

Que a la vegada, aquesta funció mostrarMenuOffline te definida una funció private per iniciar una nova partida:

private void iniciarNuevaPartidaOffline()

Funcionalitat: Inicia una nova partida offline i que controla el flux de la partida.

Localització: main/Controller/JuegoController.java, public class JuegoController.

private void mostrarMenuCargaPartida(String fileName)

Funcionalitat: Mostra un menú per carregar una partida des d'una carpeta específica.

Localització: main/Controller/JuegoController.java, public class JuegoController.

Test: test/ControllerTest/ JuegoController.java, public class JuegoControllerTest.

- **void testMostrarMenuCargaPartida_CarpetaNoExiste()** : Caixa blanca, Mockito i statement coverage.
- **void testMostrarMenuCargaPartida_NoHayPartidasGuardadas()** : Caixa negra i particions equivalents. Caixa blanca.
- **void testMostrarMenuCargaPartida_PartidasGuardadas()** : Caixa blanca i path coverage.
- **void testMostrarMenuCargaPartida_Salida()** : Caixa negra i particions equivalents. Caixa blanca.
- **void testMostrarMenuCargaPartida_OpcionNoValida0()** : Caixa negra, particions equivalents i valors frontera. Caixa blanca.
- **void testMostrarMenuCargaPartida_OpcionNoValida999()** : Caixa negra, particions equivalents i valors frontera. Caixa blanca.
- **void testMostrarMenuCargaPartida_OpcionNoNumerica()** : Caixa negra i particions equivalents. Caixa blanca.
- **void testMostrarMenuCargaPartida_ConPartidaGuardada()** : Caixa blanca, Mockito i path coverage.

private void cargarPartida(String fileName, String nombreArchivo)

Funcionalitat: Carga una partida guardada a partir del nombre d'un arxiu.

Localització: main/Controller/JuegoController.java, public class JuegoController.

Test: test/ControllerTest/ JuegoController.java, public class JuegoControllerTest.

- **void testCargarPartida_ErrorAlCargar()** : Caixa negra i Mockito. Caixa blanca i statement coverage.
- **void testCargarPartida_NoExisteArchivo()** : Caixa negra, Mockito i particions equivalents.
- **void testGuardarPartidaYCargarPartida_Exito()** : Caixa blanca, Mockito, decision coverage i condition coverage.

private void mostrarMenuGuardarPartida(String fileName)

Funcionalitat: Mostra un menú per guardar la partida amb un nom escollit pel jugador.

Localització: main/Controller/JuegoController.java, public class JuegoController.

Test: test/ControllerTest/ JuegoController.java, public class JuegoControllerTest.

- **void testMostrarMenuGuardarPartida_Cancelar()**
- **void testMostrarMenuGuardarPartida_Guardar()**

Tots aquests testos implementen: Caixa negra, particions equivalent i Mockito.

private void guardarPartida(String fileName, String nombrePartida)

Funcionalitat: Guarda la partida actual amb un nom específic en la carpeta escollida.

Localització: main/Controller/JuegoController.java, public class JuegoController.

Test: test/ControllerTest/ JuegoController.java, public class JuegoControllerTest.

- **void testGuardarPartidaYCargarPartida_Exito()**
- **void testMostrarMenuGuardarPartida_Cancelar()**
- **void testMostrarMenuGuardarPartida_Guardar()**
- **void testGuardarPartida_ErrorGuardado() :** Caixa negra, particions equivalent i Mockito.
- **void testGuardarPartida_ErrorAlGuardar() :** Caixa negra, particions equivalent i Mockito.
- **void testGuardarPartida_ArchivoExistente() :** Caixa blanca.

private boolean jugarTurno()

Funcionalitat: Controla el flux del torn del jugador actual: Guardar partida o verificar si ha dit UNO.

Localització: main/Controller/JuegoController.java, public class JuegoController.

Test: test/ControllerTest/ JuegoController.java, public class JuegoControllerTest.

- **void testJugarTurno_robarCarta()**
- **void testJugarTurno_jugarCartaValida1()**
- **void testJugarTurno_jugarCartaValida2()**
- **void testJugarTurno_jugarCartaValida3()**
- **void testJugarTurno_jugarCartaInvalida()**
- **void testJugarTurno_jugarCartaFueraDeMano1()**
- **void testJugarTurno_jugarCartaFueraDeMano2()**
- **void testJugarTurno_jugarCartaNoNumerica()**
- **void testJugarTurno_jugadorGana()**

Tots aquests testos implementen: Caixa negra, particions equivalent. Caixa blanca, statement coverage, decision coverage i statement coverage. Mockito.

private boolean verificarUno(int tiempoLimite, int cartasPenalizacion)

Funcionalitat: Verifica si s'ha dit UNO dins el temps límit, si no ha fa, roba dos cartes.

Localització: main/Controller/JuegoController.java, public class JuegoController.

Test: test/ControllerTest/ JuegoController.java, public class JuegoControllerTest.

- **void testVerificarUno_DiceUnoCorrectamente()**
- **void testVerificarUno_NoRespondeATiempo()**
- **void testVerificarUno_RespuestaIncorrecta()**

Tots aquests testos implementen: Caixa negra, caixa blanca i mockito.

public boolean verificarFinDelJuego()

Funcionalitat: Verifica si la partida ha finalitzat.

Localització: main/Controller/JuegoController.java, public class JuegoController.

Test: test/ControllerTest/ JuegoController.java, public class JuegoControllerTest.

- **void testVerificarFinDelJuego_ganador() :** Caixa negra, particions equivalent i Mockito.

private void pausar()




























Funcionalitat: Pausa el flux del joc fins que el jugador pressioni la tecla Enter.

Localització: main/Controller/JuegoController.java, public class JuegoController.

Nota: Aquesta funció es complementaria a les funcions anterior, que ajuden a detindre el flux del joc. No te casos de prova propis.

Coverage de Controller:

JuegoController

Element	Missed Instructions	Cov	Missed Branches	Cov	Missed Cxty	Missed Lines	Missed Methods
• verificarUno(int, int)		93 %		100 %	0 3	2 20	0 1
• jugarTurno()		100 %		100 %	0 11	0 47	0 1
• mostrarMenuCargaPartida(String)		100 %		100 %	0 9	0 40	0 1
• mostrarMenu()		100 %		100 %	0 5	0 24	0 1
• iniciarNuevaPartidaOffline()		100 %		100 %	0 5	0 20	0 1
• mostrarMenuOffline()		100 %		100 %	0 5	0 20	0 1
• cargarPartida(String, String)		100 %		100 %	0 3	0 15	0 1
• guardarPartida(String, String)		100 %		100 %	0 2	0 11	0 1
• mostrarMenuGuardarPartida(String)		100 %		100 %	0 3	0 12	0 1
• verificarFinDelJuego()		100 %		100 %	0 2	0 9	0 1
• pausar()		100 %		n/a	0 1	0 4	0 1
• JuegoController()		100 %		n/a	0 1	0 4	0 1
• iniciarPartida(int)		100 %		n/a	0 1	0 2	0 1
• setPartida(Partida)		100 %		n/a	0 1	0 2	0 1
• setInterfaz(InterfazJuego)		100 %		n/a	0 1	0 2	0 1
• lambda\$verificarFinDelJuego\$1(Jugador)		100 %		n/a	0 1	0 1	0 1
• lambda\$mostrarMenuCargaPartida\$0(File, String)		100 %		n/a	0 1	0 1	0 1
Total	5 of 786	99 %	0 of 72	100 %	0 55	2 232	0 17

Nota: Com es pot veure a la imatge, ens apareix que verificarUno no recorre totes les instruccions. Si mirem la imatge de més a baix, comprovem que es tracta d'un catch per quan el nostre test no pot processar la nostra resposta. No es que el test no falli, sinó que, el try especifica un catch per quan no es pugui tractar la nostra resposta. Al no poder tallar l'execució i que la resposta l'agafa ja que només permet strings, aquesta instrucció mai s'executa.

```
try {
    // Esperar respuesta dentro del tiempo límite
    String respuesta = future.get(tiempoLimite, TimeUnit.SECONDS);

    // Verificar si la respuesta es UNO
    if (respuesta.trim().equalsIgnoreCase("UNO")) {
        interfaz.mostrarMensaje(";Correcto! Has dicho \"UNO\" a tiempo.");
        pausar();
        return true;
    } else {
        interfaz.mostrarMensaje(";Respuesta incorrecta! No dijiste \"UNO\".");
    }
} catch (TimeoutException e) {
    interfaz.mostrarMensaje(";Se acabó el tiempo! No dijiste \"UNO\" a tiempo.");
} catch (InterruptedException | ExecutionException e) {
    interfaz.mostrarMensaje("Error al procesar tu respuesta.");
} finally {
    executor.shutdownNow(); // Liberar recursos
}
```

[Per més informació llegir els comentaris del codi]

VIEW:

Per View no era necessari realitzar ningun test, però per estar segurs ens hem tomat el temps de testejarlo:

public ConsolaView()

Funcionalitat: El constructor de la classe.

Localització: main/View/ConsolaView.java, public class ConsolaView.

Test: test/ControllerTest/ ConsolaViewTest.java, public class ConsolaViewTest.

- **void SetUp()**

De normal utilitza el constructor per defecte, però també hi ha un altre constructor:

public ConsolaView()

Funcionalitat: Constructor que permet configurar un flux d'entrada alternatiu.

Localització: main/View/ConsolaView.java, public class ConsolaView.

Test: test/ControllerTest/ ConsolaViewTest.java, public class ConsolaViewTest.

- **void testSolicitarColorComodin_EntradaValida()**
- **void testSolicitarColorComodin_EntradaInvalida()**
- **void testSolicitarColorComodin_EntradaInvalidaVariasVeces()**
- **void testSolicitarAccion()**
- **void testMostrarCarta()**

A continuació, tots els tests son @override, es a dir, forcen al compilar a comprovar en temps d'execució que s'està sobreescrivint correctament un mètode.

public void mostrarEstadoPartida (List<Jugador> jugadores, Carta ultimaCarta, String colorComodin, Jugador jugadorActual)

Funcionalitat: Es simula l'estat dels jugadors per a la partida amb diferents conbinacions de cartes i colors.

Localització: main/View/ConsolaView.java, public class ConsolaView.

Test: test/ControllerTest/ ConsolaViewTest.java, public class ConsolaViewTest.

- **void testMostrarEstadoPartida() :** Caixa negra i particions equivalents. Caixa blanca, statement coverage, decision i condition coverage. Mockito.

public void mostrarEstado(List<Carta> cartasMano, String color, String valor)

Funcionalitat: Es simula la correcta visualització de l'estat de les cartes pels diversos colors i valors.

Localització: main/View/ConsolaView.java, public class ConsolaView.

Test: test/ControllerTest/ ConsolaViewTest.java, public class ConsolaViewTest.

- **void testMostrarEstado() :** Caixa negra i particions equivalents. Caixa blanca, statement coverage, decision i condition coverage. Mockito.

public void mostrarGanador(Jugador jugador)

Funcionalitat: Valida que el missatge del guanyador es mostri correctament.

Localització: main/View/ConsolaView.java, public class ConsolaView.

Test: test/ControllerTest/ ConsolaViewTest.java, public class ConsolaViewTest.

- **void testMostrarGanador() :** Caixa negra i particions equivalents. Caixa blanca, statement coverage, decision i condition coverage. Mockito.

public void mostrarMensaje(String mensaje)

Funcionalitat: Valida que un missatge personalitzat es mostri correctament per consola.

Localització: main/View/ConsolaView.java, public class ConsolaView.

Test: test/ControllerTest/ ConsolaViewTest.java, public class ConsolaViewTest.

- **void testMostrarMensaje()** : Caixa negra i particions equivalents. Caixa blanca, statement coverage, decision i condition coverage. Mockito.

public String solicitarAccion()

Funcionalitat: Simula una entrada de joc.

Localització: main/View/ConsolaView.java, public class ConsolaView.

Test: test/ControllerTest/ ConsolaViewTest.java, public class ConsolaViewTest.

- **void testSolicitarAccion()** : Caixa negra i particions equivalents. Caixa blanca, statement coverage, decision i condition coverage. Mockito.

public String solicitarColorComodin()

Funcionalitat: Simula diverses entrades de color

Localització: main/View/ConsolaView.java, public class ConsolaView.

Test: test/ControllerTest/ ConsolaViewTest.java, public class ConsolaViewTest.

- **void testSolicitarColorComodin_EntradaValida()**
- **void testSolicitarColorComodin_EntradaInvalida()**
- **void testSolicitarColorComodin_EntradaInvalidaVariasVeces()**

Tots aquests tests implementen: Caixa negra i particions equivalents. Caixa blanca, statement coverage, decision i condition coverage. Mockito.

public void clearScreen()

Funcionalitat: Neteja la consola de missatges.

Localització: main/View/ConsolaView.java, public class ConsolaView.

Nota: No te ningun test asociat, però s'utilitza en dos funcions:

- *mostrarEstadoPartida*
- *solicitarColorComodin.*

public void mostrarCarta(Carta carta)

Funcionalitat: Valida la correcta visualització d'una carta.

Localització: main/View/ConsolaView.java, public class ConsolaView.

Test: test/ControllerTest/ ConsolaViewTest.java, public class ConsolaViewTest.

- **void testMostrarCarta()** : Caixa negra i particions equivalents. Caixa blanca, statement coverage, decision i condition coverage. Mockito.

public void mostrarCarta(Carta carta)

Funcionalitat: Metode per convertir els colors abreuiats a un nom complet.



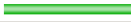
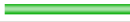

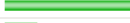
















Localització: main/View/ConsolaView.java, public class ConsolaView.

Nota: No te ningun test asociat, però s'utilitza en diverses funcions:

- *mostrarCarta*
- *mostrarEstado*
- *mostrarEstadoPartida*

Coverage de View:

ConsolaView

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
● clearScreen()		86 %		n/a	0	1	2	6	0	1
● mostrarEstadoPartida(List_Carta_String_Jugador)		100 %		100 %	0	5	0	25	0	1
● solicitarColorComodin()		100 %		100 %	0	5	0	14	0	1
● mostrarEstado(List_String_String)		100 %		100 %	0	2	0	6	0	1
● mostrarCarta(Carta)		100 %		100 %	0	2	0	6	0	1
● obtenerColorCompleto(String)		100 %		100 %	0	6	0	8	0	1
● ConsolaView()		100 %		n/a	0	1	0	3	0	1
● ConsolaView(InputStream)		100 %		n/a	0	1	0	3	0	1
● mostrarGanador(Jugador)		100 %		n/a	0	1	0	2	0	1
● solicitarAccion()		100 %		n/a	0	1	0	1	0	1
● mostrarMensaje(String)		100 %		n/a	0	1	0	2	0	1
Total	4 of 322	98 %	0 of 27	100 %	0	26	2	76	0	11

Nota: Com es pot veure a la imatge, ens apareix que clearScreen no recorre totes les instruccions. Si mirem la imatge de més a baix, comprovem que es tracta d'un catch, el mateix error que trobem al coverage de Controller.

```

@Override
public void clearScreen() {
    try {
        // Windows
        ProcessBuilder pb = new ProcessBuilder("cmd", "/c", "cls");
        pb.inheritIO().start().waitFor();
    } catch (IOException | InterruptedException ex) {
        // Si ocorre algún error al ejecutar el comando
        System.err.println("No se pudo limpiar la pantalla.");
    }
}

```

LOOP TESTING:

El nostre codi inicialment no comptava amb un bucle on hi poder aplicar *loop testing*, però després de programar unes noves funcions i posar uns quants tests, el nostre únic loop testing, ubicat en els mètodes d'iniciarPartida, s'executa per als diferents casos ja sigui quan la llista està buida, la llista es null, casos vàlids, més de 4 jugadors,...

Hem implementat dos tipus:

- Simple, per aquells mètodes amb un paràmetre.
- Aniat, per aquells mètodes amb dos paràmetres.

Els casos de tests on això s'aplica son en **testIniciarPartida_ConNombresYCartas()** i **testIniciarPartida_ConNombres()**, aquests dos codis es poden trobar a test/ModelTest/PartidaTest.java, concretament a public class PartidaTest.

Aquí es pot veure el codi després d'aplicar els tests.

```
public void iniciarPartida(List<String> nombres, int numCartas) {
    // Precondición: El número de cartas debe ser mayor que 0 pero menor a 101 (sino la partida no acaba nunca)
    ◆ assert (numCartas > 0 && numCartas < 101) : "El número de cartas debe ser mayor que 0 pero menor a 101";

    // Precondición: La lista de nombres no debe ser null
    ◆ assert (nombres != null) : "La lista de nombres no debe ser null";

    // Precondición: La lista de nombres no debe estar vacía
    ◆ assert (!nombres.isEmpty()) : "La lista de nombres no debe estar vacía";

    // Precondición: El número de jugadores debe estar entre 2 y 4
    ◆ assert (nombres.size() >= 2 && nombres.size() <= 4) : "El número de jugadores debe ser entre 2 y 4";

    // Implementado para poder reutilizar la misma partida (borrando los jugadores anteriores antes de comenzar)
    ◆ if (!jugadores.isEmpty()) {
        jugadores.clear();
    }

    ◆ for (String nombre : nombres) {
        jugadores.add(new Jugador(nombre));
    }

    // Repartir las cartas iniciales a cada jugador
    ◆ for (Jugador jugador : jugadores) {
    ◆     for (int i = 0; i < numCartas; i++) {
        jugador.robarCarta(mazo);
    }
    }

    public void iniciarPartida(int num_jugadores) {
        // Precondición: El número de jugadores debe estar entre 2 y 4
        ◆ assert (num_jugadores >= 2 && num_jugadores <= 4) : "El número de jugadores debe ser entre 2 y 4";

        List<String> nombres = new ArrayList<>();

        // Crear lista de nombres para jugadores predeterminados
        ◆ for (int i = 0; i < num_jugadores; i++) {
            nombres.add("Jugador" + (i + 1));
        }

        this.iniciarPartida(nombres, 7); // Llamada con los valores por defecto (7 cartas)
    }

    public void iniciarPartida(List<String> nombres) {
        // Llamada al método iniciarPartida con 7 cartas por defecto
        this.iniciarPartida(nombres, 7); // Llamada con los valores por defecto (7 cartas)
    }

    public void iniciarPartida(int num_jugadores, int numCartas) {
        // Precondición: El número de jugadores debe estar entre 2 y 4
        ◆ assert (num_jugadores >= 2 && num_jugadores <= 4) : "El número de jugadores debe estar entre 2 y 4";

        List<String> nombres = new ArrayList<>();

        // Crear lista de nombres para jugadores predeterminados
        ◆ for (int i = 0; i < num_jugadores; i++) {
            nombres.add("Jugador" + (i + 1));
        }

        // Llamada al método iniciarPartida con los nombres generados y el número de cartas especificado
        this.iniciarPartida(nombres, numCartas);
    }
}
```

[Per més informació llegir els comentaris del codi]

PAIRWISE TESTING:

El Pairwise Testing és un mètode combinatori de prova que, per a cada parell de paràmetres d'entrada a un sistema prova totes les possibles combinacions discretes d'aquests paràmetres. Utilitzant vectors de prova escollits amb cura, això es pot fer molt més ràpid que una cerca exhaustiva de totes les combinacions de tots els paràmetres paral·lelitzant les proves de parells de paràmetres.

En aquest context, els paràmetres són:

- `this.color`: Color de la carta actual (r, g, b, y o null).
- `this.valor`: Valor de la carta actual (1, 5, skip, +2, wild, +4).
- `otraCarta.color`: Color de la otra carta (r, g, b, y o null).
- `otraCarta.valor`: Valor de la otra carta (1, 5, skip, +2, wild, +4).

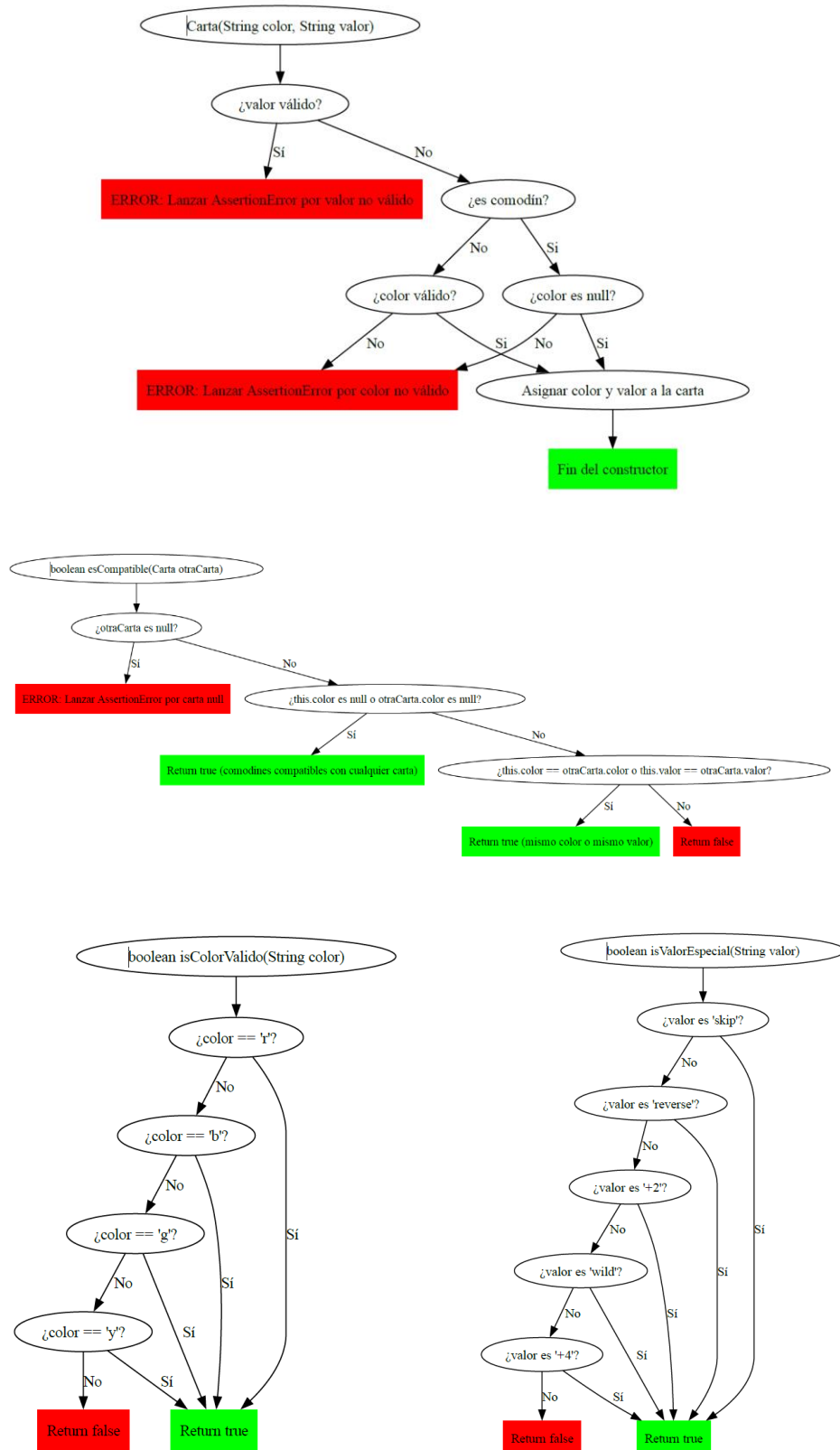
Per al nostre codi hem dissenyat la taula amb una extensió de **Visual PICT**, una eina que automatitza la generació de combinacions pairwise. Aquesta eina assegura que cada par de valors possibles dels paràmetres mencionats sigui avaluat al menys, una vegada.

Un problema amb el que tenir compte es que, els comodins només poder tenir el color nulls i la extensió genera comodins amb color, per la qual cosa hi ha que modificar-ho manualment tenint que comprovar que després de la modificació segueix complint les normes de pairwise testing.

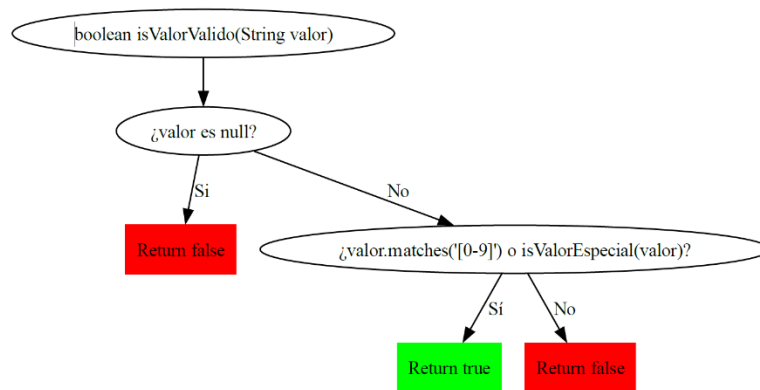
Els casos de tests on això s'aplica son en **testEsCompatiblePairwise()** aquests codi es pot trobar a `test/ModelTest/CartaTest.java`, concretament a public class `CartaTest`.

PATH COVERAGE:

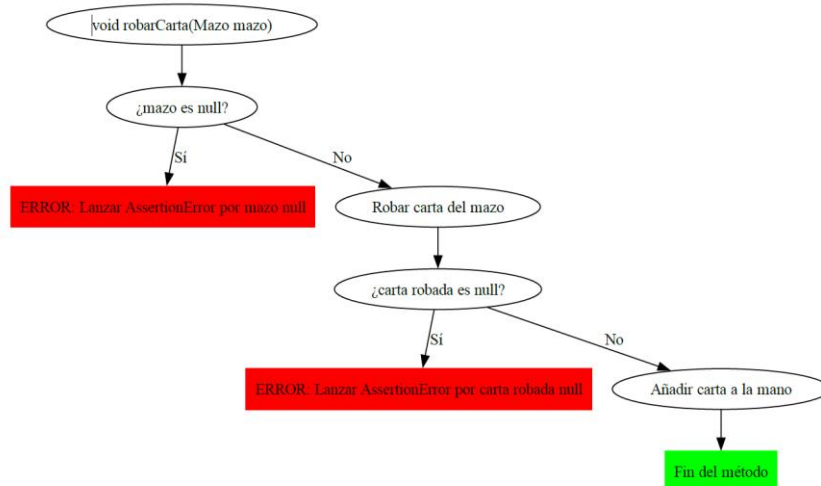
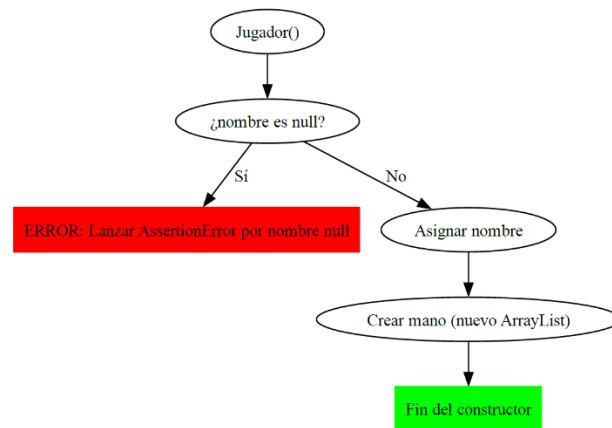
Carta:



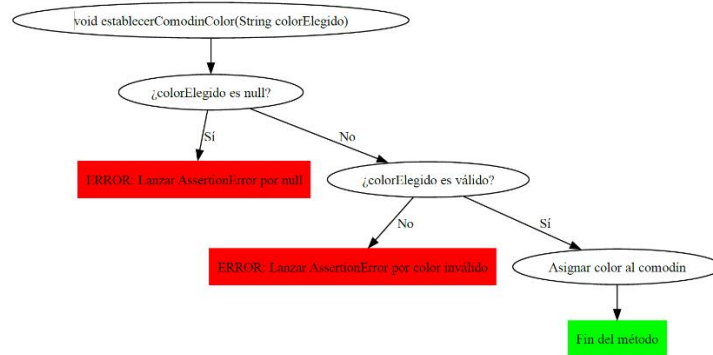
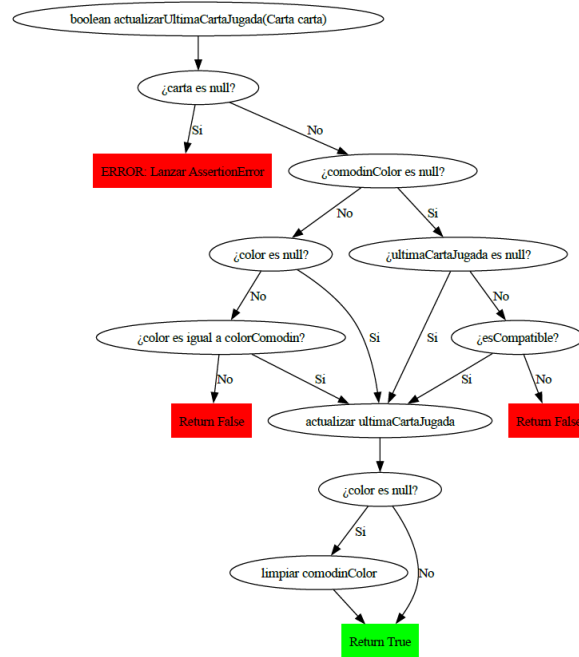
[Per més informació llegir els comentaris del codi]



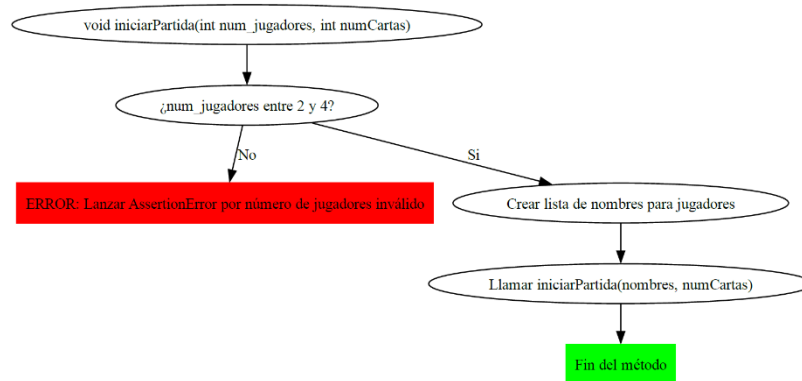
Jugador:



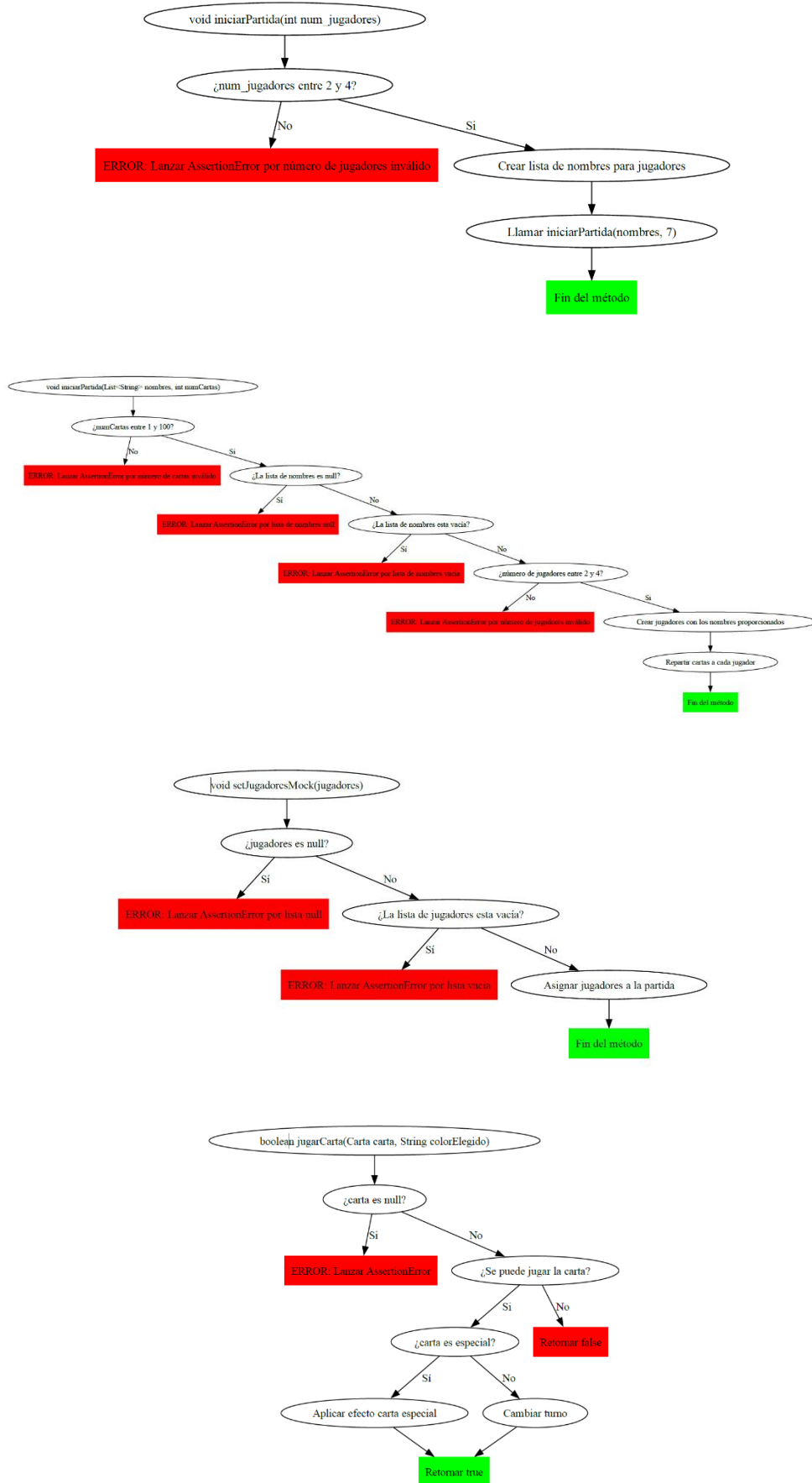
Mazo:



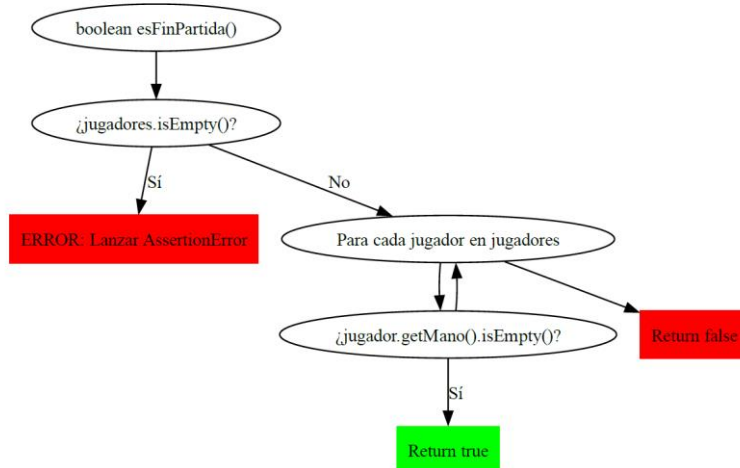
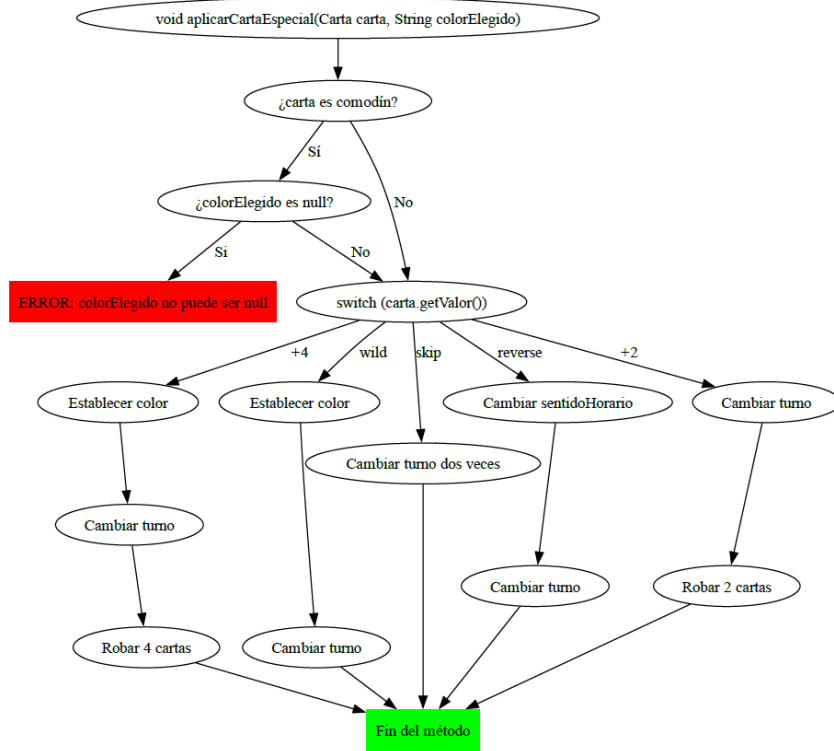
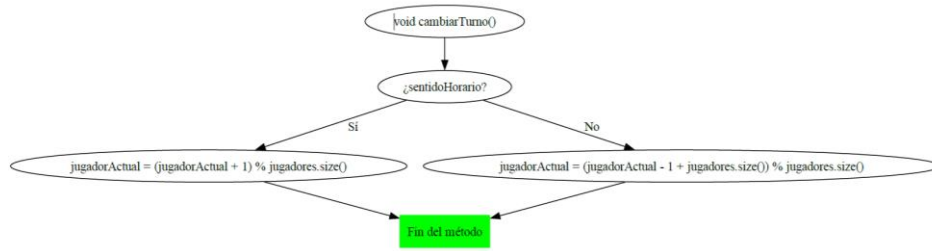
Partida:



[Per més informació llegir els comentaris del codi]



[Per més informació llegir els comentaris del codi]



[Per més informació llegir els comentaris del codi]