Memoria Práctica 3

Realizado por:
Antonio Quirante Hernandez
Jordi Conde Molina
Pablo Borrego Megías
Manuel Contreras Orge

Índice

Diseño de pruebas de unidad y de componentes

En este grupo de tests se va a comprobar que, al crear el tablero todas la variables de las que se compone nuestro juego se crea, tal y como queremos y esperamos. En total se van a hacer 5 tests.

Test rondas

Al crear el tablero de la partida el número de rondas de esta inicialmente debe de ser 0, por lo que vamos a comprobarlo en este test.

```
group('tablero', () {
    test('valor 0', () {
      final tablero1 = new tablero.vacio();
      expect(tablero1.numeroRonda, 0);
});
```

Test array jugadores

Otras de las cosas que hay que comprobar es que los jugadores que existentes al crear el tablero donde se va a jugar es 0 (al ser un array el valor va a ser []), ya que aunque ahora mismo siempre va a haber solo 2 jugadores por partida (jugador persona y jugador IA), en un futuro podrá haber una cantidad mayor de estos, por lo que no vamos a inicializarlos al crear el tablero, si no que se van a meter dichos jugadores a la hora de crear la partida.

```
test('valor []', () {
      final tablero1 = new tablero.vacio();
      expect(tablero1.jugadores, []);
});
```

Test número jugadores

Como se ha explicado antes, al crear el tablero no va a existir ninguna cantidad predeterminada de jugadores en dicho tablero, por lo que el valor de la variable *numJugadores* debe de ser 0.

```
test('valor 0', () {
      final tablero1 = new tablero.vacio();
      expect(tablero1.numJugadores, 0);
});
```

Test ultima carta

Ahora, en este test, vamos a comprobar que a la hora de crear el tablero, se le asigna una carta del mazo a la carta existente en el tablero, es decir, que la variable de tipo *carta* con el nombre *ultimaCarta* no es una carta sin valor (ya que al crear una carta con el constructor de esta clase se le asigna el color blanco y el número -1).

```
test('valor color distinto de blanco ', () {
      final tablero1 = new tablero.vacio();
      expect(tablero1.ultimaCarta.color != 'blanco', true);
});
```

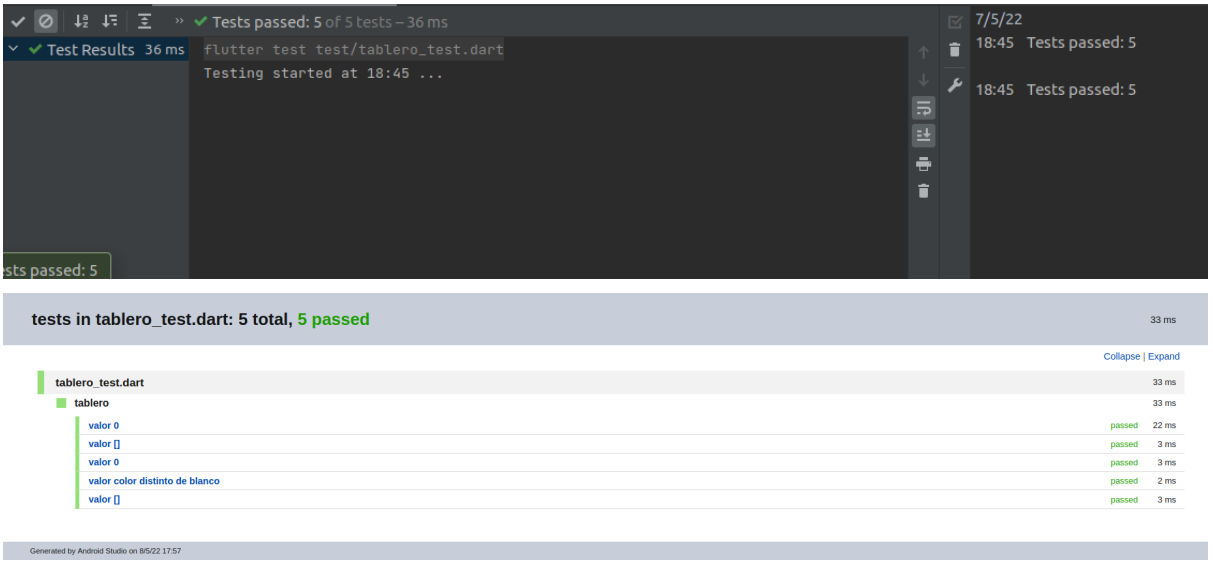
Test mazo

Por último, a la hora de crear el tablero, solo nos falta comprobar que el mazo de la partida se crea correctamente. Para ello, vamos a comprobar que el mazo de las cartas en este constructor está vacío, ya que este se llena a la hora de empezar la partida.

```
test('valor []', () {
      final tablero1 = new tablero.vacio();
      expect(tablero1.mazo, []);
});
```

Resultados

El resultado de los distintos tests explicados anteriormente, que se pasan correctamente, se puede observar en las imágenes de abajo.



Pruebas de unidad (grupo)

En estos tests se van a evaluar distintas funcionalidades de nuestro programa que son esenciales para su correcto funcionamiento, todas estas funcionalidades se encuentran dentro de las clases *tablero* y *jugador* (y sus clases derivadas *jugadorPersona* y *jugadorIA*).

Valores iniciales del constructor con parámetros

En este test se ha comprobado que al iniciar una instancia de la clase tablero con su constructor con parámetros, ésta se inicializa correctamente así como sus parámetros. Para hacer este test simplemente, hemos creado unos objetos de tipo jugador y los añadimos a un array, luego este array se pasa por parámetro del constructor de tablero. Se espera que el tamaño del mazo sea de 39, al array de jugadores sea de tamaño 2 y el numero de ronda= 0.

```
List<jugador> jugadores = [];
      jugadores.add(player1);
      jugadores.add(player2);


      final tablero1 = new tablero(jugadores);
      expect(tablero1.mazo.length, 39); //Se asigna una a ultimaCarta
una, por tanto debe haber 40 - 1 cartas en el mazo
      expect(tablero1.jugadores.length, 2);
      expect(tablero1.numeroRonda, 0);
```

Correcto reparto de cartas

Aquí vamos a comprobar que el método *asignarCartas* de la clase *tablero* funciona correctamente, dicho método lo que hace es asignar las 7 cartas iniciales a cada uno de los distintos jugadores que van a jugar la partida, por lo que en este test vamos a crear 2 jugadores, añadirlos al array de jugadores que se meterá como parámetro a la hora de crear el tablero, y una vez hecho esto habrá que asignar las cartas mediante la función explicada antes.

```
List<jugador> jugadores = [];
jugadores.add(player1);
jugadores.add(player2);

final tablero1 = new tablero(jugadores);
tablero1.asignarCartas(0);
tablero1.asignarCartas(1);
```

```
expect(tablero1.jugadores[0].mano.length, 7);
expect(tablero1.jugadores[1].mano.length, 7);
```

Test jugar

En este test se va a comprobar que el método *jugar* del jugador IA funciona correctamente, en dicha función la IA, dependiendo de la carta que está en el tablero (variable que se llama *ultimaCarta)*, comprueba si tiene una carta que pueda echar o, en caso de que no tenga ninguna que se corresponda, robar una carta nueva para su mazo. Para hacer esto, creamos el tablero con los jugadores y hacemos que el jugador con id igual a 1 (la IA) juegue, por lo que después de esa acción la cantidad de cartas en su mazo debe de ser distinta a la inicial, que es 7; esa es la acción que vamos a evaluar en este test.

```
final tablero1 = new tablero(jugadores);
tablero1.asignarCartas(1);

tablero1.jugar();

expect(tablero1.jugadores[1].mano.length != 7, true);
```

Test robar carta

En el siguiente test se va a evaluar el correcto funcionamiento de la acción de robar carta, que lo hacen los dos tipos de jugadores (la persona y la IA). Esta función se encarga de coger una carta aleatoria del mazo de cartas, la elimina de dicho mazo para que no la pueda tener a la vez el otro jugador, y se la asigna al jugador que ha invocado a dicho método; por lo que vamos a comprobar que el tamaño de la mano del jugador que va a llamar a esta función (que es en este caso es el jugador IA) se de 8, ya que vamos a llamar a la función *robar* nada más empezar la partida.

```
List<jugador> jugadores = [];
jugadores.add(player1);
jugadores.add(player2);

final tablero1 = new tablero(jugadores);
tablero1.asignarCartas(1);

tablero1.jugadores[1].robar_carta(tablero1.mazo);

expect(tablero1.jugadores[1].mano.length, 8);
```

Elección de carta válida

En este test se comprueba que la función de elegir carta elige correctamente una carta válida según las normas del juego. Para ello hemos forzado una situación en la que la carta que hay en en tablero es de color rojo y el jugador tiene en su mazo una carta de color rojo también. Se comprueba que la carta que se ha elegido es del mismo color que la que hay en el tablero.

```
final tablero2 = new tablero(jugadores);
tablero2.ultimaCarta.color='rojo';
tablero2.ultimaCarta.numero=4;

tablero2.jugadores[1].mano.add(tablero2.mazo[22]);
tablero2.jugadores[1].cartas_restantes++;

carta c=tablero2.jugadores[1].elegir_carta(tablero2.ultimaCarta);

expect(tablero2.ultimaCarta.color == c.color, true);
```

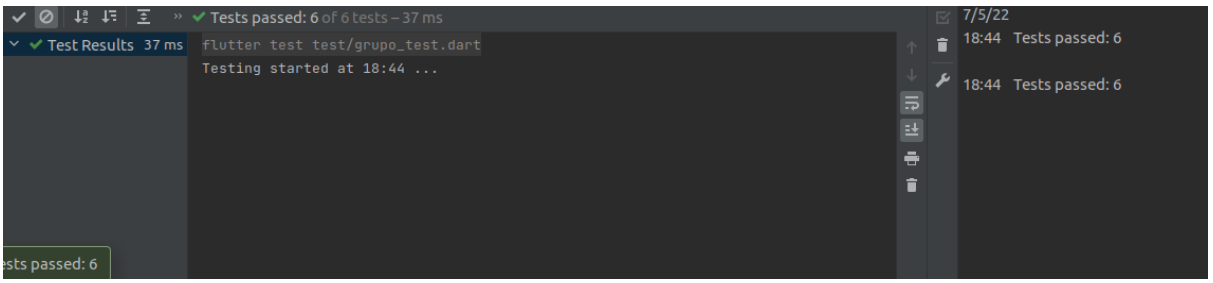
Test crear carta

En este sencillo test se comprueba que el constructor de la clase carta genera cartas de color y número aleatorio y correcto. Para ello se comprueba si el color es distinto de blanco (valor de carta inválido) y el número es distinto de -1(valor incorrecto).

```
carta card = new carta();

expect(card.color != 'blanco' && card.numero != -1, true);
```

Resultados

El resultado de estos tests descritos anteriormente se pueden observar en las imágenes de abajo, en las que se puede observar que el resultado de la ejecución de los tests es correcta, tal y como esperábamos.



Pruebas de componentes (Widgets)

En estos test se van a comprobar que funcionen correctamente distintos widgets que encontramos en nuestro programa, dentro de las clases *home* y *juego*. El primer test se hará en en la clase *home* y los otros dos en la clase *juego*.

Test 1

En este primer test vamos a comprobar que el botón de la pantalla principal (clase *home*), que nos lleva a la pantalla de juego (clase *juego*) existe; para esto vamos a comprobar que existe un texto en el que ponga ‘Iniciar Nueva Partida’, que es el que contiene el botón que estamos comprobando.

```
testWidgets('Boton iniciar Partida', (WidgetTester tester) async {
    //Creamos los widgets
      await tester.pumpWidget(MaterialApp(home: Home()));
      expect (find.text('Iniciar Nueva Partida'), findsOneWidget);
});
```

Test 2

En el segundo test se comprueba que al pulsar el botón de robar en la pantalla de juego se incrementa en 1 la cantidad de cartas que dispone el jugador persona en su mano. Para hacer esto se comprueba que al principio la mano del jugador

contiene 7 cartas, que son las iniciales del juego y que al pulsar el botón de robar esta cantidad aumenta a 8.

```
testWidgets('Incremento mazo jugador', (WidgetTester tester) async
{
    //Creamos los widgets
    await tester.pumpWidget(MaterialApp(home: Juego()));
    expect (find.text('Cartas: 7'), findsOneWidget);
    expect (find.text('Cartas: 8'), findsNothing);

    await tester.tap(find.text('Robar'));
    await tester.pumpAndSettle(Duration(seconds: 3));

    expect (find.text('Cartas: 7'), findsNothing);
    expect (find.text('Cartas: 8'), findsOneWidget);

});
```

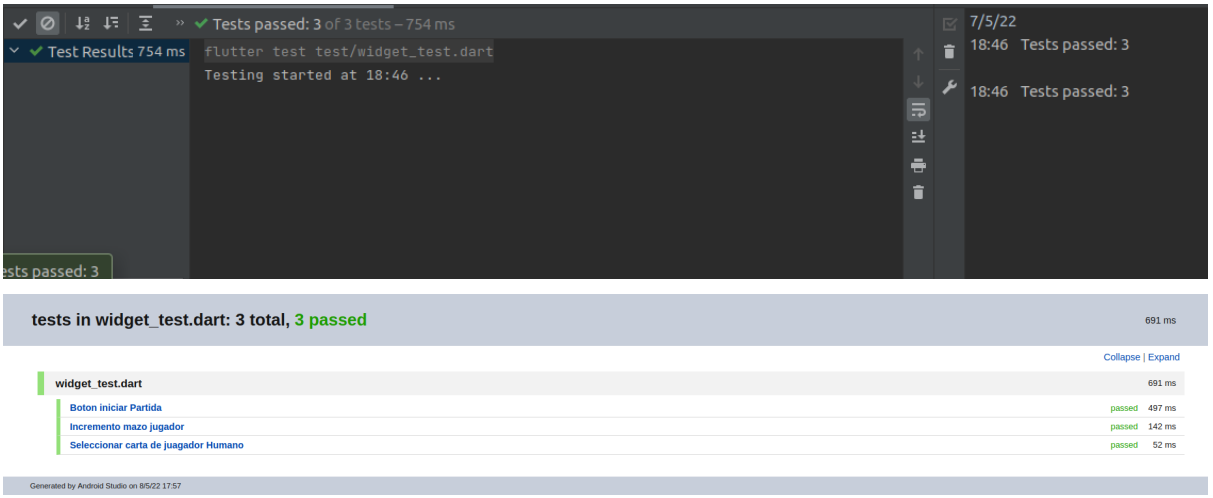
Test 3

En el último test se va a comprobar que el botón de echar carta del jugador persona cumple su cometido, para ello vamos a comprobar que antes de robar existen 25 cartas en el mazo, que son las que hay nada más empezar en el juego y loa cantidad de cartas existentes en el mazo después de que robe el jugador y juegue la IA.

```
testWidgets('Seleccionar carta de juagador Humano', (WidgetTester
tester) async{
    await tester.pumpWidget(MaterialApp(home: Juego()));

    expect(find.text('Mazo: 25'), findsOneWidget);

    await tester.tap(find.text('Robar'));
    await tester.pumpAndSettle(Duration(seconds: 3));

    expect(find.text('Mazo: 25'), findsOneWidget);
});
```

Resultados

Los 3 test explicados anteriormente los pasa nuestro programa correctamente, como se puede observar en las imágenes de abajo.



Cambios en el programa

Mientras ocurría la realización de esta práctica, nos hemos dado cuenta de que cuando el jugador persona (el jugador 0 del array) echaba una carta al tablero correcta, la carta que estaba anteriormente en el tablero no se añadía al mazo de las cartas del juego, del cuál se extraen las cartas cuando algunos de los jugadores roba una carta; esto sí ocurría cuando su adversario (el jugador IA) echa una carta válida al tablero. Para solucionar esto, ha bastado con poner lo mismo que se tenía en el método jugar de la clase jugadorIA en el código dónde se comprueba que la carta seleccionada por el jugador es correcta, antes de sobreescribir la carta que se encuentra en el tablero.

```
this.actual.tablero1.mazo.add(cartaTablero);
```

Con esto ha sido suficiente para solucionar este error.

También, hemos puesto en el juego, al lado de la carta que se encuentra en el tablero, la cantidad de cartas que se encuentran en el mazo de cartas del cuál se escogen aleatoriamente las cartas al robar.

```
child: Text('Mazo: ${this.actual.tablero1.mazo.length}',
```

Texto que muestra la longitud del mazo de cartas.

El otro error que se ha encontrado fue que podíamos, como jugador persona echar más de una carta válida al tablero o robar más de una de ellas antes de que la IA eche la carta correspondiente, aunque le contara el turno y después echara o rabara cartas, dependiendo la carta del tablero y las de su mazo. Esto se debía a que, para que se viera lo que hacía la IA, añadimos un retraso de 3 segundos, después del

cuál, la IA hacía el movimiento que le tocaba. Para solucionarlo, hemos creado una variable de tipo booleana llamada jugada, la cuál se inicializa a falso y cuando el jugador roba o echa una carta correcta, esta variable se cambia a verdadera, lo que hace que se introduzca el retraso de 3 segundos y que cuando se intente hacer un movimiento antes de que la IA haga su movimiento, aparezca un mensaje por pantalla.