REPORT_TQS_PRACTICA1

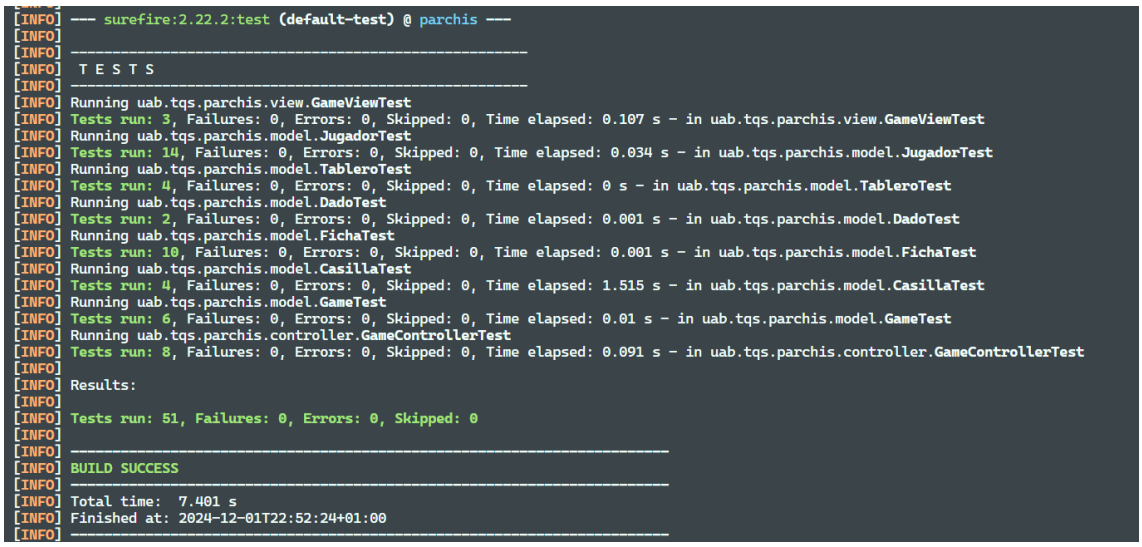
<Hugo Alonso Toledo>

<Gerard Garrido Sánchez>

<Dimecres 15:00-17:00>

<Parchis>

TEST MAVEN:



FUNCIONALITATS JOC PER CLASES:

DAU

Funcionalitat: <Llançament del dau>

Localització: <*Dado.java*, mètode *tirar()*>

Test: DatoTest.java, mètode testTirar:

- **Descripció:**
  - El test comprova que el mètode **tirar()** del dau retorna un número entre 1 i 6 inclosos. Aquest mètode garanteix que els valors possibles del dau són correctes segons dels regles del joc.
- **Tècniques utilitzades:**
  - Caixa negra: Per verificar que es generen els valors i aquets són correctes sense considerar la implementació i segons les regles del joc.
  - Path coverage: Per garantir que s'ha provat el path de la funció que genera el nombres aleatoris dintre del rang de valors (1-6).
  - Statement coverage: Es verifica que totes les línies de codi en el mètode tirar() s'executen correctament durant el test.
- **Comentaris:**
  - Els tests assegura que el comportament aleatori es manté dins dels límits establerts i que tots els valors possibles s'han verificat en repeticions.

- S'ha utilitzat una combinació de tècniques **caixa negra** i **caixa blanca** per cobrir totes les rutes possibles en la lògica de moviment i garantir el correcte funcionament del mètode.

Funcionalitat: <Resultat llançament del dau>

Localització: <Dado.java, mètode tirar(), getResultado()>

Test: DatoTest.java, mètode testSecuenciaTiradas:

- **Descripció:**
  - El test comprova que la crida al mètode **getResultado()** després de diverses invocacions a **tirar()** retorna el valor correcte, és a dir, l'últim resultat generat pel dau.
  - L'objectiu és verificar que el mètode **getResultado()** reflecteix el valor de la darrera tirada, mantenint la coherència interna de l'objecte.
- **Tècniques utilitzades:**
  - Caixa negra: Es valida el comportament extern del mètode, sense revisar la implementació interna. Es comprova que el valor retornat per **getResultado()** coincideix amb l'última tirada.
  - Caixa Blanca: Es verifica la lògica interna de l'objecte Dado, garantint que el valor de **getResultado()** s'actualitza correctament després de cada crida a tirar().
  - Statement coverage: Es comprova que totes les línies dels mètodes s'executin en cada crida.
- **Comentaris:**
  - Aquest test assegura que **getResultado()** sempre reflecteix correctament l'última tirada del dau, amb els valors generats actualitzant-se adequadament.
  - **Caixa Negra** es fa servir per comprovar que el comportament extern es compleix sense importar la implementació interna.
  - **Caixa Blanca** s'aplica per validar que la funció de recuperació del resultat actualitza i retorna correctament el valor de la tirada.
  - Els **tests per Dado** asseguren que el comportament aleatori es manté dins dels límits establerts i que tots els valors possibles (1-6) són verificats repetidament. S'ha utilitzat una combinació de tècniques **caixa negra** i **caixa blanca** per cobrir totes les rutes possibles en la lògica de moviment.

FITXA

Funcionalitat: <Moure la fitxa>

Localització: <Ficha.*java*, mètode *mover(int pasos)*>

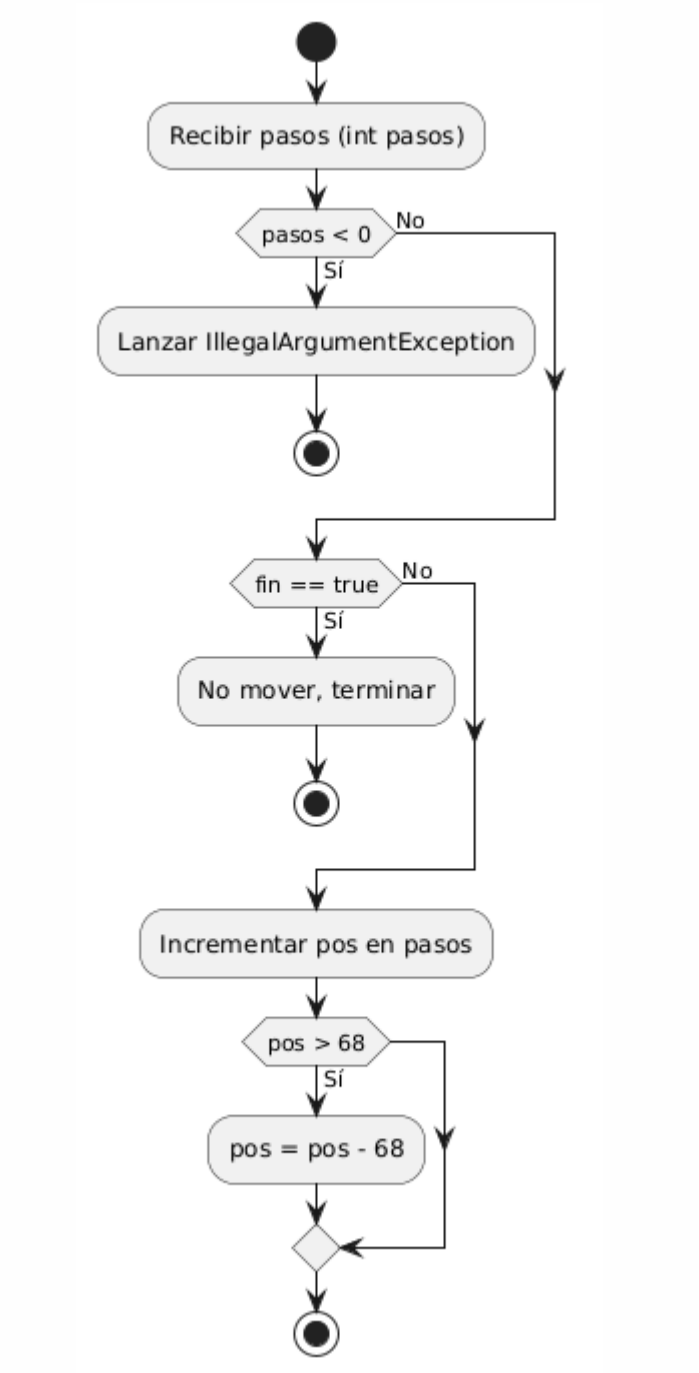
Test: FichaTest.java, mètodes testMoverCuandoNoEstaEnFin, testNoMoverCuandoEstaEnFin, , testMoverCeroPasos, testMoverAlPrincipio, testMoverPasosNegativosLanzaExcepcion, testMoverPasosNegativosLanzaExcepcionLimite, testMoverCercaDelFinal, testMoverMasDe68Pasos, testMoverCuandoEstaEnMeta:

- **Descripció:**
  1. **testMoverCuandoNoEstaEnFin**:
     - **Objectiu**: Verifica que la fitxa es mou correctament quan no està a la casa (home = false) ni a la meta (fin = false).
     - **Tècniques utilitzades**:

        - **Particions equivalents**: Comprova una situació representativa amb valors vàlids i sense restriccions.
        - **Path coverage**: Cobreix el camí on la fitxa es mou sense cap bloqueig.
2. **testNoMoverCuandoEstaEnFin**:
    - **Objectiu**: Valida que una fitxa que ha arribat a la meta (fin = true) no es mou més.
    - **Tècniques utilitzades**:
        - **Path coverage**: Cobreix el camí on el moviment està bloquejat per la condició fin.
        - **Valors límit**: Comprova el cas límit on la posició és 0 com a valor inicial, assegurant-se que la fitxa no es mogui després d'arribar a la meta.
3. **testMoverCeroPasos i testMoverAlPrincipio**:
    - **Objectiu**: Comprova que la posició de la fitxa no canvia quan es mou 0 passos.
    - **Tècniques utilitzades**:
        - **Particions equivalents/límits/frontera**: Comprova el límit del valor dels passos (pasos = 0) i també al primer pas (pasos = 1).
4. **testMoverPasosNegativosLanzaExcepcion-Limite**:
    - **Objectiu**: Assegura que el mètode llença una excepció (IllegalArgumentException) quan s'intenta moure un número negatiu de passos.
    - **Tècniques utilitzades**:
        - **Valors límit i frontera**: Prova un valor fora del rang permès (pasos = -1 i pasos = -3), assegurant-se que s'activa l'excepció.
        - **Path coverage**: Cobreix el camí on la validació dels passos negatius activa l'excepció.
5. **testMoverCercaDelFinal**:
    - **Objectiu**: Verifica que una fitxa es mou correctament quan es troba a la posició 67 i es mou 2 passos, hauria de passar a la posició 1, superant la posició 68.
    - **Tècniques utilitzades**:
        - **Caixa negra**: Assegura que el comportament de moviment es manté dins dels límits establerts.
        - **Path coverage**: Cobreix el camí de moviment on la fitxa supera la posició 68 i es retorna a la posició inicial.
6. **testMoverMasDe68Pasos**:
    - **Objectiu**: Comprova que la fitxa es mou correctament quan es mou més enllà de la posició 68, i s'ajusta circularment.
    - **Tècniques utilitzades**:
        - **Caixa negra**: Verifica el comportament esperat quan la fitxa supera els límits de la taula.
        - **Path coverage**: Cobreix el camí de moviment circular, que fa que la fitxa torni a la posició inicial després de passar la casella 68.

7. **testNoMoverCuandoEstaEnMeta**:
    - **Objectiu**: Assegura que una fitxa no es mogui quan ha arribat a la meta.
    - **Tècniques utilitzades**:
        - **Caixa blanca**: Verifica que la condició fin impedeix el moviment de la fitxa un cop ha arribat a la meta.
        - **Decision coverage**: Prova les rutes condicionals dins del mètode mover() per assegurar que el comportament es bloqueja correctament quan la fitxa és a la meta.

          Diagrama uml de mover(int pasos):



- **Comentaris:**
    1. Cobertura de camins:
        - Els tests cobreixen tots els camins d'execució dins del mètode mover(), incloent situacions on el moviment està bloquejat (home, fin) o és una entrada vàlida/no vàlida, per tant, també es compleix **statement coverage** per a tota la classe ficha.
    2. Tècniques aplicades:
        - **Caixa negra**: S'ha utilitzat per verificar el comportament esperat de la fitxa sense considerar la implementació interna del mètode mover().
        - **Caixa blanca**: Permet validar que les condicions internes, com home i fin, bloquegen el moviment quan és necessari.
        - **Particions equivalents**: Han ajudat a provar les rutes amb diferents valors de passos (positius, zero i negatius).
        - **Valors frontera i limits:** Ens ha permès verificar els comportaments als límits, com el moviment amb 0 passos o el moviment al final del tauler.
        - **Path coverage**: S'ha assegurat que tots els camins dins del mètode han estat explorats, cobrint tant les rutes normals com les excepcionals.
        - **Decision coverage**: S'ha utilitzat per verificar que les decisions de moviment de la fitxa (com el bloqueig en fin) són acomplides correctament.

Pairwise Testing Clase Fitxa:

Cas	*home*	*fin*	*pasos*	Escenari	Resultat esperat
1	true	false	0	La fitxa és a casa i no s'intenta moure.	La posició continua a 0.
2	ture	true	5	La fitxa és a casa i també a la meta, amb un intent de moure 5 passos.	La posició no canvia, ja que està bloquejada per home i fi.
3	false	false	70	La fitxa no és a casa ni a la meta, i s'intenta moure més enllà del límit del tauler.	La posició s'ajusta circularment a 2 després d'excedir-ne el màxim (68).
4	false	true	0	La fitxa no és a casa però ja és a la meta, amb un intent de moure 0 passos.	La posició no canvia, ja que està bloquejada per fi.
5	false	False	5	La fitxa no és a casa ni a la meta i s'intenta un moviment normal de 5 passos.	La posició avança correctament a 5.

Codi testPairwiseMover():

```java
@Test
void testPairwiseMover() {
    // Caso 1: home = true, fin = false, pasos = 0
    ficha.setHome(true);
    ficha.setFin(false);
    ficha.setPos(0);
    ficha.mover(0);
    assertEquals(0, ficha.getPos(), "La ficha no debería moverse si está en casa.");

    // Caso 2: home = true, fin = true, pasos = 5
    ficha.setHome(true);
    ficha.setFin(true);
    ficha.setPos(0);
    ficha.mover(5);
    assertEquals(0, ficha.getPos(), "La ficha no debería moverse si está en casa y en la meta.");

    // Caso 3: home = false, fin = false, pasos = 70
    ficha.setHome(false);
    ficha.setFin(false);
    ficha.setPos(0);
    ficha.mover(70);
    assertEquals(2, ficha.getPos(), "La posición debe ajustarse circularmente si supera el tablero.");

    // Caso 4: home = false, fin = true, pasos = 0
    ficha.setHome(false);
    ficha.setFin(true);
    ficha.setPos(10);
    ficha.mover(0);
    assertEquals(10, ficha.getPos(), "La posición no debe cambiar si la ficha está en la meta.");

    // Caso 5: home = false, fin = false, pasos = 5
    ficha.setHome(false);
    ficha.setFin(false);
    ficha.setPos(0);
    ficha.mover(5);
    assertEquals(5, ficha.getPos(), "La ficha debería moverse correctamente si no está en casa ni en la meta.");
}
```

JUGADOR

Funcionalitat: <Creació e inicialització del jugador>

Localització: <Jugador.*java*, mètode *Jugador(String nombre, String color)*>

Test: JugadorTest.java, mètodes testCreacionJugadorConNombreColor, testNombreFichas, testIniciarConCuatroFichas:

- **Descripció:**
  - Aquest conjunt de tests comprova que el jugador es crea correctament amb el nom, el color, i que inicialitza les 4 fitxes correctament. El test verifica que el constructor crea correctament un objecte Jugador assignant el nom i el color proporcionats.
- **Tècniques utilitzades:**
  - Caixa negra: Verifica el comportament extern del constructor, assegurant-se que els atributs nombre, color i fitxes són correctament inicialitzats.
  - Particions equivalents: Es verifica la creació de fitxes per a valors vàlids de nom i color, així com el seu emmagatzematge.
- **Comentaris:** Aquest conjunt de tests assegura que tant el jugador com les seves fitxes es creen de manera correcta quan es crida al constructor.

Funcionalitat: Canvi de torn del jugador>

Localització: <Jugador.*java*, mètode *setTurno(boolean turno)*>

Test: JugadorTest.java, mètode testCambiarTurno, testTurnoInicialFalse:

- **Descripció:**
  - Aquests test verifiquen que el jugador comença amb el torn desactiva (false) i que el mètode setTurno permet canviar correctament l'estat del torn del jugador. Es comprova que:
    - Si l'estat inicial del torn és false, es pot canviar a true.
    - Si l'estat inicial es true, es pot canviar a false.

    - El mètode no fa res si el nou estat és igual a l'actual.
- **Tècniques utilitzades:**
    - Caixa negra: Per validar que el mètode modifica correctament l'estat extern sense considerar la implementació interna.
    - Decision/Condition Coverage:
        - Condició: Es cobreixen els possibles valors de la condició this.turno != turno (true i false).
        - Decisió: Es verifica que el mètode executa el bloc intern només quan la condició és true.
    - Path coverage: Assegura que s'exploren tots els camins possibles, incloent-hi quan el torn no canvia perquè ja està en l'estat requerit.
- **Comentaris:** El test garanteix que la funcionalitat del canvi de torn és correcta, assegurant que els jugadors puguin tenir torns dinàmics durant el joc.

Funcionalitat: <Moviment de fitxes>

Localització: <Jugador.*java*, mètode *moverFicha(int indice, int pasos)*>

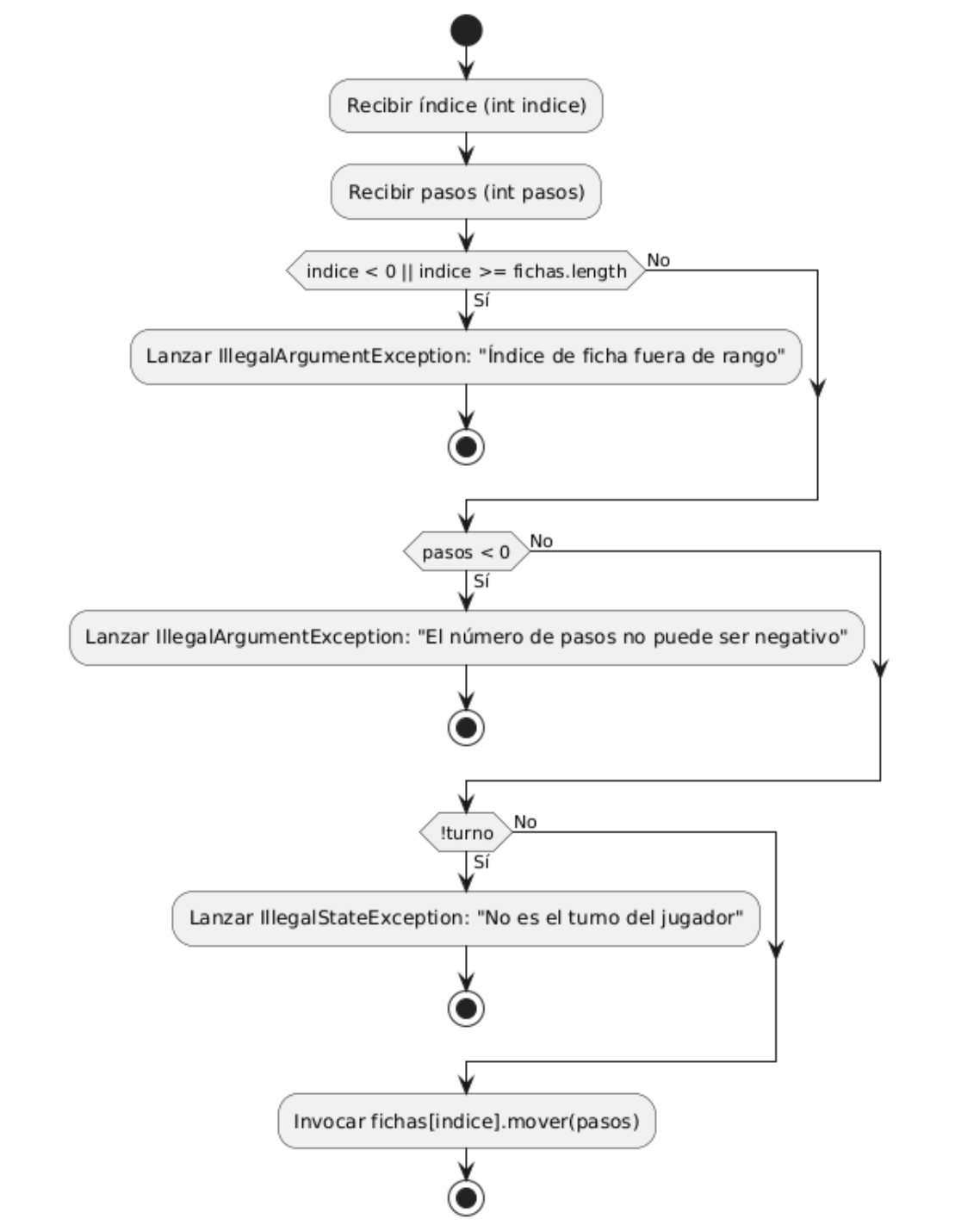
Test: JugadorTest.java, mètodes testMoverFichaEnTurno, testMoverFichaLanzaExcepcion, testMoverFichaFin:

- **Descripció:**
    - Aquest conjunt de tests verifica el comportament del mètode moverFicha() en diversos escenaris per garantir que el moviment de les fitxes segueix les regles del joc i que es gestionen correctament els errors en cas d'entrada no vàlida. Es comprova que:
        - El jugador pot moure la fitxa si el torn està activat (testMoverFichaEnTurno).
        - No es permet moure una fitxa que ja ha arribat a la meta (testMoverFichaFin).
        - Es cobreixen tots els camins possibles dins del mètode, incloent casos d'índex fora de rang i passos negatius (testMoverFichaLanzaExcepcion).
- **Tècniques utilitzades:**
    - Caixa negra: Per validar que el mètode compleix amb els requisits funcionals (p. ex., que no es permet moure fitxes fora de torn).
    - Caixa blanca:
        - Decision/Condition Coverage: Es verifica que cada condició (índex vàlid, passos positius, torn activat) funciona correctament i que les excepcions es llencen quan es compleixen les condicions d'error.
        - Path coverage: S'assegura que es cobreixen totes les rutes possibles dins del mètode, incloent-hi:
            - Casos d'índex fora de rang.
            - Intent de moure passos negatius.
            - Moviments vàlids i moviments bloquejats.
        - Statement coverage: s'executen totes les sentencies possibles.
    - Valors límit i frontera: Es prova amb valors límit per a l'índex (negatiu i més gran que 3) i per al nombre de passos (0 i negatiu).

- Partició equivalents: Es comprova el moviment dintre del joc (partició del tauler del rang).

- **Comentaris:** Aquest conjunt de tests reforça la robustesa de la funcionalitat de moviment de les fitxes. Amb el test afegit (testMoverFichaPathCoverage), es garanteix que:
  - Es llencen excepcions adequades per a entrades incorrectes.
  - El moviment de fitxes respecta les condicions del torn i la posició actual (no mou fitxes en meta).
  - Tots els camins interns del mètode són coberts, assegurant així un alt nivell de qualitat i prevenció d'errors en execució.

Diagrama uml de moverFicha(int indice, int pasos):



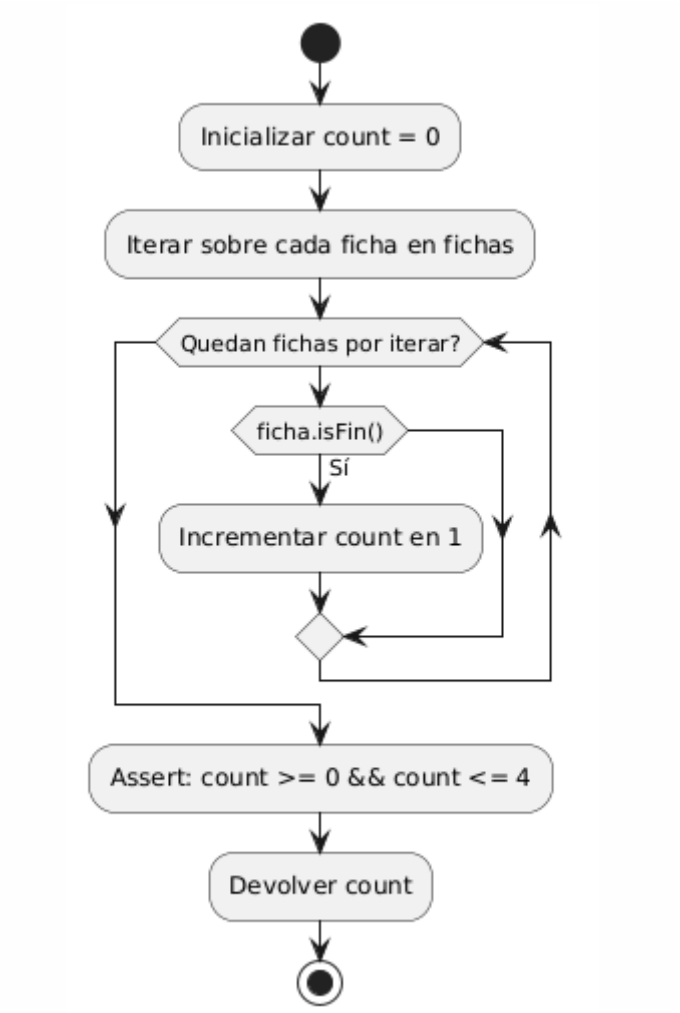
Funcionalitat: <Comptar fitxes en la meta>

Localització: <Jugador.*java*, mètode *contarFichasEnFin()*>

Test: JugadorTest.java, mètode testContarFichasEnFin:

- **Descripció:**
  - Aquest test verifica que el mètode compta correctament el nombre de fitxes que estan a la meta (fin=true).
  - El test comprova que el mètode contarFichasEnFin retorna el nombre correcte de fitxes que han arribat a la meta. Es comprova per diversos casos: cap fitxa, algunes fitxes i totes les fitxes a la meta.
- **Tècniques utilitzades:**
  - Caixa negra: Es verifica que el resultat retornat sigui correcte sense considerar la implementació interna.
  - Particions equivalents: es divideix les possibles situacions en grups on cap fitxa, algunes fitxes o totes les fitxes han arribat a la meta, per tant es proba totes les possibles combinacions.
  - Path coverage i statement coverage: es contemplant tots el possibles camins de la funció a més de verificar que s'executen totes les sentencies.
- **Comentaris:** Aquest test assegura que el comptador de fitxes a la meta funciona correctament i compta totes les fitxes que han arribat al final.

Diagrama uml de contarFichasEnFin():



Funcionalitat: <Excepcions per valors nuls en nom o color>

Localització: <Jugador.*java*, mètode Jugador(String nombre, String color)>

Test: JugadorTest.java, mètodes testNombreNuloLanzaExepcion, testColorNuloLanzaExcepcion:

- **Descripció:**
  - Comprova que es llança una excepció IllegalArgumentException quan el nom o el color són passats com a null al constructor del jugador.
- **Tècniques utilitzades:**
  - Caixa blanca: Es verifica que es llança l'excepció correctament en els casos on el nom o el color són nuls.
  - Path coverage i statement coverage: Comprova que es llança l'excepció correctament en els casos on el nom o el color són nuls i que s'executen totes les sentencies.
- **Comentaris:** El test garanteix que el sistema gestiona correctament els valors incorrectes per al nom i el color del jugador.

Funcionalitat: <Moviment de fitxes al final del tauler>

Localització: <Jugador.*java*, mètode moverFicha(int indice, int pasos)>

Test: JugadorTest.java, mètodes testMoverCercaDelFinal, testMoverMasDe68Pasos:

- **Descripció:**
  - El test verifica que la fitxa es mou correctament quan supera la posició 68 i torna a la posició inicial i també quan es mou una fitxa a les últimes posicions.
- **Tècniques utilitzades:**
  - Valors límit/frontera: provar els límits de la posició del tauler.
- **Comentaris:** Aquest test valida la lògica de moviment quan una fitxa supera el final del tauler.

CASELLA

Funcionalitat: <Número de cada casella>

Localització: <Casilla.java, mètode getNumero()>

Test: CasillaTest.java, mètode testNumero>

- **Descripció:**
  - El test comprova que cada tipus de casella retorna el número correcte. Es prova amb diferents instàncies de caselles (Casa, Normal, Segura i Final) per verificar que el número de cada una és el valor correcte que ha estat assignat al crear-les.
- **Tècniques utilitzades:**
  - Particions equivalents: Es prova per a cada tipus de casella (Casa, Normal, Segura i Final), ja que el número de la casella ha de ser un valor concret per a cada tipus.
  - Caixa negra: Per validar que el mètode getNumero() retorna els valors esperats par a cada tipus de casella
  - Path coverage: Cobreix el camí de cada tipus de casella, assegurant-se que es passa per tots els possibles valors per cada casella.

- Mock Object: el mock permet hardcodejar el retorn de les funcio getNumero() de l'objecte CasillaFinal.

Funcionalitat: <Comprovació de si una casella és segura>

Localització: <Casilla.java, mètode getSeguro()>

Test: CasillaTest.java, mètode testSeguro>

- **Descripció:**
  - El test verifica que la casella sigui segura o no, depenent del tipus de casella. Per exemple, les caselles de tipus Casa, Segura i Final són segures, mentre que les Normales no ho són.
- **Tècniques utilitzades:**
  - Particions equivalents: Per validar que el mètode getSeguro() retorna la seguretat de la casella de manera correcta.
  - Path coverage: Cobreix totes les rutes possibles de seguretat, tant per les caselles segures com per les no segures.
  - Decision coverage: Es comprova que s'executen totes les decisions possibles (segura i no segura) per a cada tipus de casella.
  - Mock Object: el mock permet hardcodejar el retorn de les funció getseguro() de l'objecte CasillaFinal.

Funcionalitat: <Afegir i treure fitxes de la casella>

Localització: <Casilla.java, mètode agregarFicha() i quitarFicha()>

Test: CasillaTest.java, mètode testAgregarQuitar>

- **Descripció:**
  - El test comprova que les fitxes es poden afegir i treure de les caselles de manera correcta. Això inclou verificar que les caselles contenen fitxes després d'afegir-les i que es buiden després de treure-les.
- **Tècniques utilitzades:**
  - Caixa negra: Per verificar que els mètodes agregarFicha() i quitarFicha() funcionen correctament.
  - Path coverage: Cobreix les rutes per afegir i treure fitxes de diferents tipus de casilla, i comprova els estats abans i després de la manipulació. (Veure imatge del test).
  - Condition/decision coverage: Es comprova que cada condició interna d'afegir o treure fitxes es cobreix adequadament (si la casilla està buida o no).

    Com podem veure a la captura, hem de comprovar si ficha es null (tant per true com false).

```java
/**
 * Agrega una ficha a la casilla.
 *
 * Precondiciones:
 * - `ficha` no debe ser `null`.
 *
 * Postcondiciones:
 * - La ficha se agrega a la lista `fichas`.
 * - Si el número de fichas en la casilla es exactamente 2, se establece `bloqueado = true`.
 * - Si hay más de 2 fichas, el estado de `bloqueado` permanece inalterado.
 *
 * @param ficha Ficha a agregar.
 */
@Override
public void agregarFicha(Ficha ficha) {
    if (ficha == null) {
        throw new IllegalArgumentException(s:"La ficha no puede ser nula.");
    }
    this.fichas.add(ficha);

    // Verificar bloqueo
    if (this.fichas.size() == 2) {
        this.bloqueado = true;
    }
}
```

Mitjançant el test, es comproven tots el escenaris possibles per cadascun del tipus de casella.

```java
@Test
void testAgregarQuitar() {
    // POSIBLE MOCK OBJECT AQUÍ
    Ficha ficha = new Ficha();

    // TEST CASILLA CASA
    casilla_casa.agregarFicha(ficha);

    assertFalse(casilla_casa.getFichas().isEmpty(), "Deberia haber alguna ficha en la casilla");

    casilla_casa.quitarFicha(ficha);

    assertTrue(casilla_casa.getFichas().isEmpty(), "No deberia haber ninguna ficha en la casilla");
```

Funcionalitat: <Estat de bloqueig de la casella>

Localització: <Casilla.java, mètode getBloqueo()>

Test: CasillaTest.java, mètode testBloqueo>

- **Descripció:**
  - El test verifica que la casella estigui bloquejada quan hi ha més d'una fitxa a la casella i desbloquejada quan es treu una fitxa. Es comprova el comportament de les caselles de tipus Normal i Segura, que tenen la capacitat de bloquejar-se quan estan ocupades per més d'una fitxa.
- **Tècniques utilitzades:**
  - Caixa negra: Per comprovar que el mètode getBloqueo() retorna l'estat correcte de la casella depenent del nombre de fitxes.
  - Decision/condition coverage: Es cobreixen totes les decisions de bloqueig i desbloqueig de la casella.
  - Path coverage: Es comprova que les rutes de bloqueig i desbloqueig de les caselles funcionen correctament.

- Valors límit i frontera: El test explora el límit entre 0 i 2 fitxes, que és el límit per activar el bloqueig (1 fitxa en una casella no la bloqueja, però 2 sí).

  Com podem veure al test es comproven tots els escenaris possibles, a més, de mirar si las condicions/decision agafen tots el valors possibles (true/false).

```java
@Test
void testBloqueo() {
    Ficha ficha1 = new Ficha();
    Ficha ficha2 = new Ficha();

    // TESTS CASILLA NORMAL
    casilla_normal.agregarFicha(ficha1);
    casilla_normal.agregarFicha(ficha2);

    assertTrue(casilla_normal.getBloqueo(), "La casilla deberia estar bloqueada");

    casilla_normal.quitarFicha(ficha2);

    assertFalse(casilla_normal.getBloqueo(), "La casilla deberia estar desbloqueada");

    // TEST CASILLA SEGURA
    casilla_segura.agregarFicha(ficha1);
    casilla_segura.agregarFicha(ficha2);

    assertTrue(casilla_segura.getBloqueo(), "La casilla deberia estar bloqueada");

    casilla_segura.quitarFicha(ficha2);

    assertFalse(casilla_segura.getBloqueo(), "La casilla deberia estar desbloqueada");
}
```

TAULER

Funcionalitat: <Inicialització del tauler>

Localització: <Tablero.*java*, mètode *inicializarTablero(), inicializarTableroFinal(), getTablero(), getTableroFinal()*>

Test: TableroTest.java, mètodes testTablero, testCasillas, testCasillasSeguras, testLoopTestingInicializarTablero:

- **Descripció:**
  1. **testTablero**:
     - **Objectiu**: Verifica que la fitxa es mou correctament quan no està a la casa (home = false) ni a la meta (fin = false). Comprova que el tauler principal té exactament 69 caselles (1 casa i 68 caselles normals/segures) i el tauler final conté exactament 8 caselles finals. També valida que, si es tracta d'accedir a una casella fora del rang, es llença una excepció (IndexOutOfBoundsException).
       - **Tècniques utilitzades**:
         - **Caixa negra**: Verifica que la longitud de les llistes de caselles sigui correcta.
         - **Particions equivalents:** Divideix l'espai d'entrada en dues categories: caselles vàlides (0-68) i caselles fora del rang (> 68).
         - **Valors límits i frontera/Particions:** Comprova les caselles límit (68 i 70) per assegurar que el comportament és correcte en els valors extrems. També amb valors negatius que estan fora de la partició.

2. **testCasillas**:
    - **Objectiu**: Verifica que:
        - La casella inicial (casa) té el número 0.
        - La casella 1 és una casella normal i no segura.
        - La casella 5 és una casella segura.
    - **Tècniques utilitzades**:
        - **Particions equivalents**: Comprova una situació representativa de cada casella i que es configuren correctament.
        - **Path coverage**: S'exploren les rutes possibles per comprovar que el sistema retorna la casella adequada en cada cas.
        - **Caixa negra**: Verifica que les propietats de les caselles siguin correctes.

      Es comproven que estiguin correctament assignades dintre del tauler:

```
@Test
void testCasillas() {
    // Casilla CASA
    assertEquals(0, tablero.getCasillaTablero(numero:0).getNumero(), "La casilla deberia ser la 0, casilla casa");

    // Casilla NORMAL
    assertEquals(1, tablero.getCasillaTablero(numero:1).getNumero(), "La casilla deberia ser la 1, casilla normal");
    assertFalse(tablero.getCasillaTablero(numero:1).getSeguro(), "La casilla deberia ser no segura");

    // Casilla SEGURA
    assertEquals(5, tablero.getCasillaTablero(numero:5).getNumero(), "La casilla deberia ser la 5, casilla segura");
}
```

3. **testCasillasSeguras**:
    - **Objectiu**: Comprova que les caselles segures (números predefinits) són identificades correctament com a segures.
    - **Tècniques utilitzades**:
        - **Decision coverage**: Cobertura de les rutes condicionals del mètode esCasillaSegura().

      A la imatge poden veure el codi de la funció i amb el test assegurem la cobertura:

```java
/**
 * Verifica si una casilla es segura según su número.
 *
 * Precondiciones:
 * - `numero` debe estar entre 1 y 68.
 *
 * Postcondiciones:
 * - Devuelve true si el número corresponde a una casilla segura.
 * - Devuelve false en caso contrario.
 *
 * @param numero Número de la casilla.
 * @return `true` si la casilla es segura, `false` en caso contrario.
 */
private boolean esCasillaSegura(int numero) {
    if (numero < 1 || numero > 68) {
        throw new IllegalArgumentException(s:"El número de casilla debe estar entre 1 y 68.");
    }

    int[] casillasSeguras = {5, 12, 17, 22, 29, 34, 39, 46, 51, 56, 63, 68};
    for (int segura : casillasSeguras) {
        if (numero == segura) {
            return true;
        }
    }
    return false;
}
```

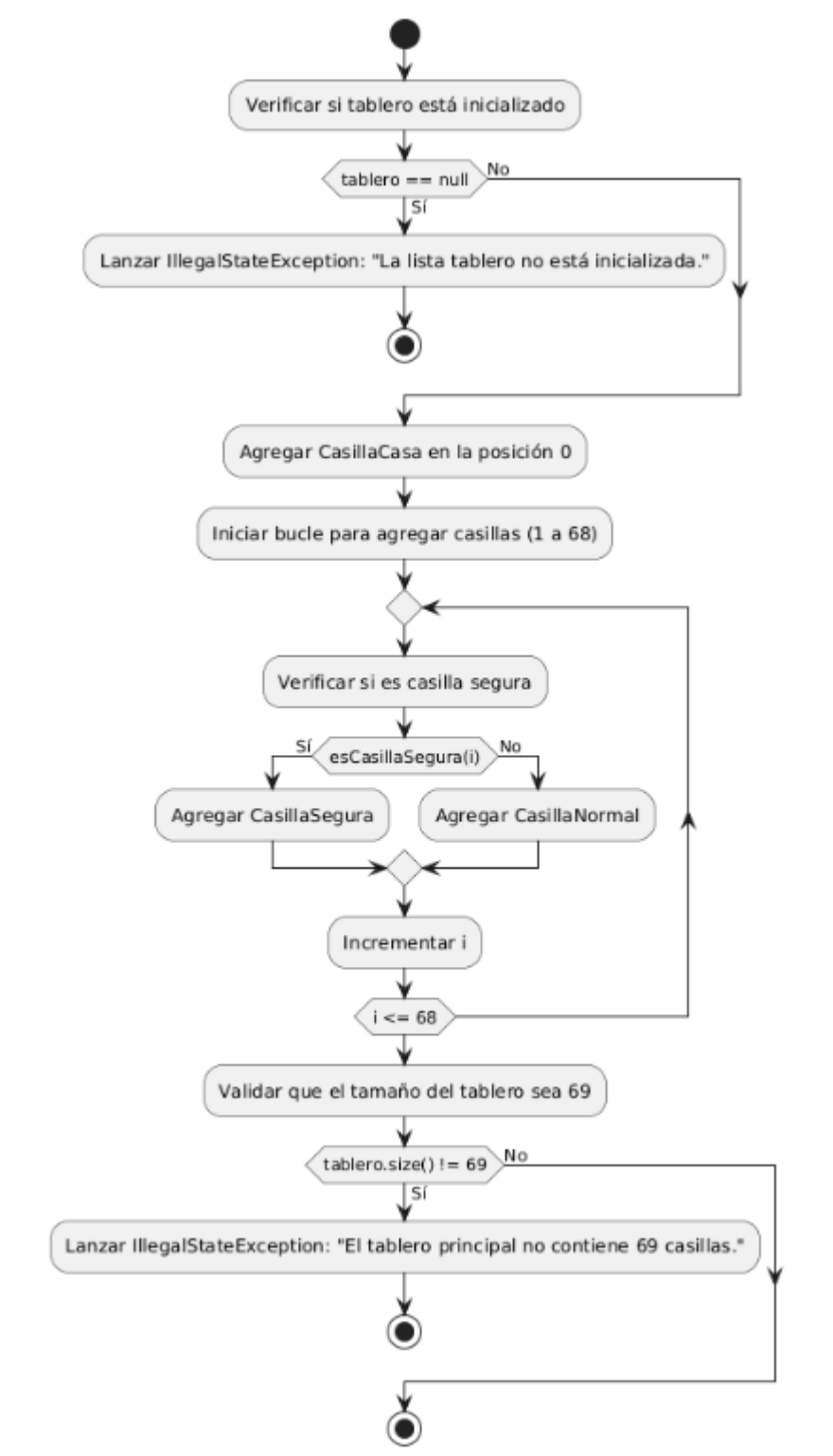
- **Path coverage**: Cobreix totes les rutes de validació de caselles segures i no segures.

Es comproven totes les possibles posicions de les caselles segures:

```java
@Test
void testCasillasSeguras() {
    int[] seguras = {5, 12, 17, 22, 29, 34, 39, 46, 51, 56, 63, 68};
    for (int numero : seguras) {
        assertTrue(tablero.getCasillaTablero(numero).getSeguro(), "La casella " + numero + " ha de ser segura.");
    }
}
```

- **Caixa negra**: Assegura que les caselles segures es marquen correctament al tauler.

Diagrama uml de inicializarTablero(): (Següent pagina)



4. **testLoopTestingInicializarTablero**:
    - **Objectiu:** Verifica que totes les caselles del tauler principal (0-68) i totes les caselles finals (1-8) existeixen després de la inicialització del tauler.

- **Tècniques utilitzades**:
  - **Loop testing**: Prova totes les iteracions dels bucles de la inicialització per garantir que totes les caselles s'inicialitzen correctament.
  - **Statement coverage i path coverage**: Cobertura de totes les línies de codi en els mètodes d'inicialització.

  Es comproven totes les possibles posicions de les caselles dintre del tauler:

```
@Test
void testLoopTestingInicializarTablero() {
    for (int i = 0; i < 69; i++) {
        assertNotNull(tablero.getCasillaTablero(i), "La casella " + i + " ha d'existir al tauler principal.");
    }
    for (int i = 1; i <= 8; i++) {
        assertNotNull(tablero.getCasillaTableroFinal(i - 1), "La casella " + i + " ha d'existir al tauler final.");
    }
}
```

  - **Particions equivalents**: Comprova valors els límits del bucle (primer i últim element de les llistes) a més de totes les posicions dintre del mateix.

- **Comentaris:**
  - Tècniques utilitzades en conjunt:
    - **Path coverage:** Valida totes les rutes possibles en els mètodes inicializarTablero(), getCasillaTablero(), i getCasillaTableroFinal().
    - **Caixa negra**: Comprova el comportament extern de la classe Tablero sense entrar en la seva implementació interna.
    - **Caixa blanca:** Verifica les condicions internes de la classe, com la correcta identificació de les caselles segures.
    - **Particions equivalents:** Utilitzades per provar els comportaments per a casos vàlids i no vàlids (com caselles fora del rang).
    - **Valors límit:** Es verifica el comportament als límits de les caselles del tauler.
    - **Loop testing:** Examina les iteracions dels bucles per garantir que totes les caselles són inicialitzades correctament.
  - Conclusió:
    - Els tests cobreixen exhaustivament la funcionalitat de la classe Tablero i garanteixen que les caselles es gestionen correctament, que el tauler s'inicialitza de forma precisa i que totes les condicions estan degudament cobertes.

GAME

Funcionalitat: <Afegir jugadors>

Localització: <*Game.java*, mètode *agregarJugadores(String nombre, String color)*>

Test: GameTest.java, mètode testAgregarJugadoresValidos I testAgregarJugadorExcedeLimite:

- **Descripció:**
  - El test testAgregarJugadoresValidos comprova que el sistema permet afegir jugadors amb noms i colors vàlids fins a un màxim de 4 jugadors. Per cada jugador afegit, es verifica que el nom i el color s'han assignat correctament.

- El test testAgregarJugadorExcedeLimite verifica que, si es prova d'afegir un cinquè jugador, el sistema llença una excepció IllegalStateException amb el missatge adequat.

- **Tècniques utilitzades:**
  - Caixa negra: Per verificar que els jugadors vàlids s'afegeixen correctament.
  - Caixa blanca(**Condition/Decision coverage**): Per cobrir la lògica de la condició que impedeix afegir més de 4 jugadors.

  Es comproven que els jugadors siguin vàlids i que siguin com a màxim 4 jugadors, del contrari llença excepció (Veure test i codi següents):

```java
/**
 * Agrega un nuevo jugador al juego.
 *
 * Precondiciones:
 * - El número de jugadores actuales debe ser menor que 4.
 *
 * Postcondiciones:
 * - El nuevo jugador se agrega a la lista de jugadores.
 * - Lanza una excepción si ya hay 4 jugadores.
 *
 * @param nombre Nombre del jugador.
 * @param color Color del jugador.
 * @throws IllegalStateException Si ya hay 4 jugadores en la lista.
 */
public void agregarJugador(String nombre, String color) {
    if (jugadores.size() >= 4) {
        throw new IllegalStateException(s:"El máximo número de jugadores es 4.");
    }
    jugadores.add(new Jugador(nombre, color));
}
```

```java
@Test
void testAgregarJugadoresValidos() {
    game.agregarJugador("Jugador 1", "amarillo");
    game.agregarJugador("Jugador 2", "azul");
    game.agregarJugador("Jugador 3", "rojo");
    game.agregarJugador("Jugador 4", "verde");

    assertEquals(4, game.getJugadores().size(), "Debería haber 4 jugadores en la lista.");
    assertEquals("Jugador 1", game.getJugadores().get(0).getNombre());
    assertEquals("amarillo", game.getJugadores().get(0).getColor());
}

@Test
void testAgregarJugadorExcedeLimite() {
    game.agregarJugador("Jugador 1", "amarillo");
    game.agregarJugador("Jugador 2", "azul");
    game.agregarJugador("Jugador 3", "rojo");
    game.agregarJugador("Jugador 4", "verde");

    Exception exception = assertThrows(IllegalStateException.class, () -> {
        game.agregarJugador("Jugador 5", "morado");
    });

    assertEquals("El máximo número de jugadores es 4.", exception.getMessage());
}
```

  - Particions equivalents: Per garantir que s'han provat entrades vàlides i entrades que excedeixen el límit.

- **Comentaris:** Aquest test verifica que l'afegiment de jugadors es fa de manera correcta i que la informació associada (nom i color) es manté fidel a l'entrada. També valida que el nombre de jugadors no superi el límit de 4.

Funcionalitat: <Avançar torn jugador>

Localització: <*Game.java*, mètode *avanzarTurno()*>

Test: GameTest.java, mètode testAvanzarTurno:

- **Descripció:**
  - Aquest test comprova que el torn avança seqüencialment entre els jugadors. Després que un jugador finalitzi el seu torn, el control passa al següent jugador. Quan l'últim jugador finalitza el torn, el torn torna al primer jugador, garantint un flux circular.
  - Es verifica que el nom del jugador actual s'actualitza correctament després de cada avançament del torn.
- **Tècniques utilitzades:**
  - Caixa negra: Per comprovar el funcionament del flux de torn entre els jugadors.
  - Caixa blanca (Loop testin: Per cobrir totes les rutes possibles del bucle circular.
  - Path coverage: Per assegurar que totes les possibles transicions de torn són provades.

Funcionalitat: <Detecció del fin del joc>

Localització: <*Game.java*, mètode *esFinDelJuego()*>

Test: GameTest.java, mètode testEsFinDelJuego:

- **Descripció:**
  - Aquest test valida que el joc es considera acabat quan un jugador té totes les seves 4 fitxes a la meta. Es prova inicialment que el joc no finalitza si cap jugador ha completat aquesta condició.
  - Es simula que un jugador té totes les fitxes en l'estat de meta (fin = true) i es comprova que el mètode retorna true.
- **Tècniques utilitzades:**
  - Caixa negra: Per verificar que el mètode detecta correctament l'estat de finalització del joc.
  - Path coverage: Per assegurar que es prova el camí on un jugador compleix la condició i el camí on no es compleix.
  - Statement coverage: Per assegurar que totes les línies de codi del mètode són executades.

Funcionalitat: <Obtenir guanyador del joc>

Localització: <*Game.java*, mètode *obtenerGanador()*>

Test: GameTest.java, mètode testObtenerGanador:

- **Descripció:**
  - Aquest test comprova que el mètode retorna el jugador correcte com a guanyador quan aquest té totes les seves 4 fitxes a la meta. Inicialment es comprova que el mètode retorna null si cap jugador ha guanyat.

- Es simula un escenari en què un jugador té les seves 4 fitxes marcades com fin = true i es comprova que el mètode retorna el jugador correcte.

- **Tècniques utilitzades:**
  - Caixa negra: Per verificar que el guanyador es retorna correctament només quan s'ha completat la condició de guanyar.
  - Statement coverage: Per assegurar que cada línia del mètode és executada, inclosa la part que retorna null quan no hi ha guanyador.

GAME_CONTROLLER

Funcionalitat: <Inicialització del joc>

Localització: <*GameController.java*, mètode *iniciarJuego(String[] jugadores, String[] colores)*>

Test: GameControllerTest.java, mètode testIniciarJuego, testIniciarJuegoMax:

- **Descripció:**
  1. **testIniciarJuego**:
     - Valida que el mètode iniciarJuego inicialitza correctament el joc amb jugadors vàlids i colors únics.
     - Es comprova que els jugadors s'afegeixen correctament amb els noms i colors proporcionats, i que el missatge d'èxit es mostra.
  2. **testIniciarJuegoMax**:
     - Comprova que el mètode iniciarJuego gestiona adequadament el cas en què es proporcionen més de 4 jugadors.
     - Es verifica que no s'inicialitzen jugadors, que la llista de jugadors roman buida i que es mostra el missatge d'error corresponent.
- **Tècniques utilitzades:**
  - **Caixa negra**:
    - Valida el comportament extern del mètode sense tenir en compte la seva implementació interna.
    - testIniciarJuego: Assegura que els jugadors es creen correctament amb les dades vàlides.
    - testIniciarJuegoMax: Valida la restricció del nombre màxim de jugadors.
  - **Mock Object**:
  - **Statement coverage**:
    - Cobreix els camins del mètode:
      - Quan el nombre de jugadors és vàlid.
      - Quan el nombre de jugadors excedeix el límit i es mostra un error.
  - **Decision coverage/Condition coverage**:
    - Es verifica que totes les condicions són avaluades (número màxim de jugadors, coincidència de longitud entre jugadores i colores).

Es comproven que els jugadors siguin vàlids i que siguin com a màxim 4 jugadors, del contrari llença excepció (Veure test i codi següents):

```java
@Test
void testIniciarJuego() {
    String[] jugadores = {"Jugador 1", "Jugador 2"};
    String[] colores = {"Azul", "Amarillo"};

    game_controller.iniciarJuego(jugadores, colores);

    assertEquals("Jugador 1", game.getJugadores().get(0).getNombre());
    assertEquals("Azul", game.getJugadores().get(0).getColor());

    verify(game_view).mostrarMensaje("Juego inicializado correctamente.");
}

void testIniciarJuegoMax() {
    // Más de 4 jugadores: caso inválido
    String[] jugadores = {"Jugador 1", "Jugador 2", "Jugador 3", "Jugador 4", "Jugador 5"};
    String[] colores = {"Azul", "Amarillo", "Verde", "Rojo", "Violeta"};

    game_controller.iniciarJuego(jugadores, colores);

    // Verificar que no se agregan jugadores y se muestra un error
    assertEquals(0, game.getJugadores().size(), "No se deben inicializar jugadores si exceden el límite.");
    verify(game_view).mostrarError("Debe haber entre 2 y 4 jugadores con colores únicos.");
}
```

```java
/**
 * Inicializa el juego con los jugadores y colores proporcionados.
 * Precondiciones:
 * - `jugadores` y `colores` no deben ser nulos.
 * - `jugadores.length` debe ser igual a `colores.length`.
 * - Debe haber entre 2 y 4 jugadores.
 *
 * Postcondiciones:
 * - Los jugadores y sus fichas son inicializados correctamente.
 * - Las fichas se colocan en la casilla inicial (casa).
 */
public void iniciarJuego(String[] jugadores, String[] colores) {
    if (jugadores.length != colores.length || jugadores.length > 4) {
        game_view.mostrarError("Debe haber entre 2 y 4 jugadores con colores únicos.");
        return;
    }

    for (int i = 0; i < jugadores.length; i++) {
        this.game.agregarJugador(jugadores[i], colores[i]);
    }

    Casilla casilla_casa = game.getTablero().getCasillaTablero(0);
    for (Jugador jugador : game.getJugadores()) {
        for (Ficha ficha : jugador.getFichas()) {
            casilla_casa.agregarFicha(ficha);
        }
    }

    game_view.mostrarMensaje("Juego inicializado correctamente.");
    actualizarVista();
}
```

Funcionalitat: <Llençar dau>

Localització: <*GameController.java*, mètode *lanzarDado()*>

Test: GameControllerTest.java, mètode testLanzarDado:

- **Descripció:**
  - Comprova que el mètode lanzarDado() retorna un valor vàlid en el rang de 1 a 6. Aquest test no necessita una comprovació específica ja que el valor del dau es gestiona internament.
- **Tècniques utilitzades:**

- Caixa negra: Comprova que el comportament extern del dau és correcte (l'usuari no necessita veure la implementació interna).

Funcionalitat: <Avançar el torn>

Localització: <*GameController.java*, mètode avanzarTurno()>

Test: GameControllerTest.java, mètode testAvanzarTurno:

- **Descripció:**
  - Comprova que el mètode avanzarTurno() funciona correctament, avançant al següent jugador i després tornant al primer jugador quan s'ha completat el cicle.
- **Tècniques utilitzades:**
  - Caixa blanca: Verifica que el canvi de torn s'implementa correctament internament seguint la lògica de la classe Game.
  - Decision coverage: Cobreix les condicions de la lògica de l'avanç de torn per assegurar-se que es realitza el canvi de torn adequadament.
- **Comentaris:** Aquest test valida que el sistema gestiona correctament el canvi de torn entre jugadors, avançant al següent jugador i reiniciant-se un cop arriba al final.

Funcionalitat: <Moviment inicial de la fitxa>

Localització: <*GameController.java*, mètode *movimientoInicial(Ficha ficha)*>

Test: GameControllerTest.java, mètode testMovimientoInicial:

- **Descripció:**
  - Comprova que la fitxa es mou correctament des de la casella de casa fins a la seva casella inicial, segons el color del jugador. Es verifica que la fitxa s'ha mogut i que el seu estat es correspon amb la posició correcta.
- **Tècniques utilitzades:**
  - Caixa negra: Verifica que el comportament extern del mètode és correcte, sense considerar la implementació interna.

Funcionalitat: <Moure una fitxa>

Localització: <*GameController.java*, mètode moverFicha(Ficha ficha, int cantidad)>

Test: GameControllerTest.java, mètode testMoverFicha:

- **Descripció:**
  - Comprova que el mètode moverFicha() mou la fitxa correctament la quantitat de passos especificada. Es verifica que el moviment es realitza bé des de la posició inicial de la fitxa.
- **Tècniques utilitzades:**
  - Caixa negra: Verifica que el comportament extern del mètode funciona correctament (el moviment de la fitxa es fa adequadament).
- **Comentaris:** Aquest test valida que el sistema gestiona correctament el canvi de torn entre jugadors, avançant al següent jugador i reiniciant-se un cop arriba al final.

Funcionalitat: <Bloqueig del moviment>

Localització: <*GameController.java*, mètode *moverFicha(Ficha ficha, int cantidad)*>

Test: GameControllerTest.java, mètode testMoverFichaBloqueo:

- **Descripció:**
  - Comprova que el mètode moverFicha() funciona correctament quan una fitxa es topa amb una altra fitxa a la mateixa casella (bloqueig). Es verifica que la fitxa es quedi bloquejada i no es mogui més.
- **Tècniques utilitzades:**
  - Caixa negra: Comprova que el comportament extern funciona correctament quan la casella està bloquejada per una fitxa.
- **Comentaris:** Aquest test valida que la lògica de bloqueig de fitxes funciona correctament quan una fitxa ocupa una casella amb una fitxa enemiga.

Funcionalitat: <Moviment final del tauler>

Localització: <*GameController.java*, mètode *moverFicha(Ficha ficha, int cantidad)*>

Test: GameControllerTest.java, mètode testFinalInicioTablero:

- **Descripció:**
  - Comprova que quan una fitxa supera la casella final (68), torna a la casella inicial del tauler (posició 1).
- **Tècniques utilitzades:**
  - Caixa negra: Verifica el comportament extern del mètode sense conèixer la seva implementació interna.
  - Statement coverage/Path coverage: Cobreix el camí que recorre la fitxa quan supera la casella final i torna a començar.
- **Comentaris:** Aquest test verifica que el moviment de la fitxa funciona correctament quan supera la casella final.

Funcionalitat: <Arribar a la meta>

Localització: <*GameController.java*, mètode *moverFicha(Ficha ficha, int cantidad)*>

Test: GameControllerTest.java, mètode testMoverFichaFinal:

- **Descripció:**
  - Comprova que quan una fitxa arriba a la meta, s'actualitza el seu estat i es marca com finalitzada. Es valida que el mètode moverFicha() actualitza l'estat fin de la fitxa correctament.
- **Tècniques utilitzades:**
  - Caixa negra: Verifica que la fitxa es marca correctament com a finalitzada quan arriba a la meta.
- **Comentaris:** Aquest test assegura que les fitxes són correctament marcades com a finalitzades quan arriben a la meta del tauler.

GAME_VIEW

Funcionalitat: <Mostrar missatge>

Localització: <GameView.*java*, mètode mostrarMensaje(String mensaje)>

Test: GameViewTest.java, mètode testMostrarMensaje:

- **Descripció:**
  - Comprova que el mètode mostrarMensaje() imprimeix correctament el missatge al flux de sortida de la consola amb el format adequat.

- **Tècniques utilitzades:**
  - Caixa negra: Valida que el comportament extern del mètode és correcte, sense considerar la implementació interna.
- **Comentaris:** Aquest test valida que el sistema pot mostrar missatges a l'usuari amb el format correcte.

Funcionalitat: <Mostrar missatge d'error>

Localització: <GameView.*java*, mètode mostrarError(String mensaje)>

Test: GameViewTest.java, mètode testMostrarError:

- **Descripció:**
  - Comprova que el mètode mostrarError() imprimeix correctament un missatge d'error al flux de sortida de la consola amb el format adequat.
- **Tècniques utilitzades:**
  - Caixa negra: Valida que el comportament extern del mètode és correcte, imprimint els missatges d'error amb el format adequat.
- **Comentaris:** Aquest test assegura que els missatges d'error es mostren de manera adequada a l'usuari en cas d'incidències o errors en el joc.

Funcionalitat: <Mostrar el tauler>

Localització: <GameView.*java*, mètode mostrarTablero(Tablero tablero)>

Test: GameViewTest.java, mètode testMostrarTablero:

- **Descripció:**
  - Comprova que el mètode mostrarTablero() imprimeix el tauler complet (tant les caselles del tauler principal com les caselles finals) amb els detalls de les caselles.
- **Tècniques utilitzades:**
  - Caixa negra: Verifica que el comportament extern del mètode imprimeix la informació del tauler de manera correcta.
  - Path coverage: Cobreix els camins on el tauler es mostra completament, incloent les caselles de tipus CasillaCasa, CasillaNormal, CasillaSegura i CasillaFinal.
  - Decision coverage Verifica que totes les condicions del mètode (com els tipus de caselles) es compleixen correctament durant l'execució.:

Funcionalitat: <Esperar a la resposta de l'usuari>

Localització: <GameView.*java*, mètode esperarRespuesta()>

Test: GameViewTest.java, mètode testEsperarRespuesta:

- **Descripció:**
  - Comprova que el mètode esperarRespuesta() llegeix correctament l'entrada de l'usuari i la retorna com un valor enter. Aquest test simula l'entrada d'un número per l'usuari.
- **Tècniques utilitzades:**
  - Caixa negra: Verifica que el comportament extern del mètode funciona correctament, llegint i processant l'entrada de l'usuari.
  - Statement coverage: Cobreix la línia d'execució en què el mètode espera i processa la resposta de l'usuari.

- **Comentaris:** Aquest test valida que el mètode per llegir l'entrada de l'usuari funciona adequadament, gestionant correctament la conversió de l'entrada a un valor enter.