ANALISIS DE PRUEBAS UNITARIAS

ControlGeneralTest

El test del control general del programa coordina cada uno de los métodos y el flujo general del programa.

Primero se importan las clases a las que el control tiene que llamar y junto con las herramientas de Junit.

Se cargan

```
@TempDir
  Path tempDir;
 @BeforeEach
  public void setUp() {
    controlGeneral = new ControlGeneral();
  }


  @AfterEach
  public void tearDown() {
        controlGeneral = null;
  }
```

Para crear un directorio temporal , crear una nueva instancia de ControlGeneral antes de cada test, garantizando que las pruebas sean independientes entre sí y liberar la referencia después de cada test para evitar contaminación entre casos.

Analizando cada uno de los métodos que tenemos

testCargarEquiposDesdeProperties_ArchivoValido.

Crea un archivo .properties temporal con dos equipos válidos y lo pasa al método cargarEquiposDesdeProperties().Simula cargar un archivo de configuración válido.

testCargarEquiposDesdeProperties_ArchivoNull.

Prueba qué pasa si le pasas `null` en lugar de un archivo.
Debe devolver un mensaje de error en lugar de romper el programa.

testCargarEquiposDesdeProperties_ArchivoInexistente.

Prueba un archivo que no existe en el sistema.

testIniciarPartida_EquiposValidos.

Carga equipos válidos, inicia una partida entre ellos y verifica que el mensaje indique éxito, confirma que ControlGeneral integra bien con ControlJuego.

testIniciarPartida_EquiposInexistentes.

Intenta iniciar una partida con equipos que no existen.

testSiguienteLanzamiento_SinPartidaIniciada.

Prueba que si no hay partida en curso, `siguienteLanzamiento()` no funcione y devuelva un error.

testSiguienteLanzamiento_ConPartidaIniciada.

Simula una partida activa y lanza un turno, comprueba que los valores generados (puntos, jugador, equipo) sean válidos.

testNuevaRonda.

Verifica que el método `nuevaRopnda()` devuelva algún texto (aunque no haya partida activa), comprueba que no falle ni devuelva null.

testGuardarEstado.

Asegura que `guardarEstado()` no lance excepciones, incluso sin datos cargados.

testObtenerHistorial.

Llama al método que lee el historial de partidas desde archivo RAF, asegura que siempre devuelva una lista válida (aunque esté vacía).

testFlujoCompleto.

Simula todo el flujo completo del programa:

Carga equipos. Inicia partida. Hace un lanzamiento. Guarda estado. Obtiene historial.

crearArchivoEquiposTest.

Genera un archivo de prueba temporal con dos equipos y varios jugadores.

ControlJuegoTest

Simula una partida con equipos precargados, para evaluar cada método de la clase y recibir su correcta ejecución en el proyecto final.

Importa las clases necesarias (Equipo, Jugador y las herramientas de JUnit).

Importa los métodos de aserción (assertTrue, assertFalse, assertEquals, etc.), que se usan para comprobar si el código funciona como se espera.

Primero se definen atributos, luego

@BeforeEach

void setUp() {

controlJuego = new ControlJuego();

con lo que se crean los 2 equipos que estarán precargados para usarlos en las pruebas.

testIniciarPartidaConEquiposValidos.

Una partida con equipos válidos inicia correctamente.

El mensaje de inicio contiene “Ronda 1 iniciada”.

No haya errores ni mensajes de selección inválida.

testIniciarPartidaConEquiposInvalidos.

Si se intenta iniciar una partida con equipos inválidos o iguales, el sistema debe mostrar un mensaje de error, esto comprueba que el control de validaciones en `ControlJuego` funciona correctamente.

testSiguienteLanzamientoSinPartidaIniciada.

Si se llama a `siguienteLanzamiento()` **sin haber iniciado una partida**, debe generar un error controlado.

testSiguienteLanzamientoConPartidaIniciada.

Tras iniciar una partida, un lanzamiento devuelve información válida, se verifica:

Que no haya errores.

Que haya un jugador y un equipo.

Que los puntos estén dentro del rango [0,8].

Que se asigne una jugada válida.

Que la ronda sea la primera y no haya muerte súbita.

testFlujoCompletoDeUnaMano.

Los jugadores de un equipo lancen en orden.

Al terminar la mano, cambie el turno al otro equipo.

testPuntajesYJugadas.

Evalúa que cada puntaje tenga su jugada asociada correctamente y que no existan valores de puntaje fuera del rango.

testNuevaRonda.

Que al llamar a `nuevaRonda()` después de iniciar una partida, se muestre un mensaje adecuado.

testNuevaRondaSinPartidaIniciada.

Si se intenta comenzar una nueva ronda sin tener partida activa, el sistema debe advertirlo.

testJugadoresEnOrdenCorrecto.

Que los jugadores sean tomados en el orden exacto en que fueron agregados al equipo.

testCambioDeEquipoDespuesDeMano.

Que después de una mano completa (4 lanzamientos), el turno pase correctamente al siguiente equipo.